Bachelor's Thesis (TUAS)

Information Technology

2011

Aaron Pratt

RICH INTERNET APPLICATION DEVELOPMENT WITH THE VAADIN JAVA FRAMEWORK

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree Program in Information Technology

15.8.2011 | 39 pages

Instructor: Patric Granholm

Aaron Pratt

RICH INTERNET APPLICATION DEVELOPMENT WITH THE VAADIN JAVA FRAMEWORK

The purpose of this work was to design and create a new, custom web application to assist the client in the planning of musical events involving organizing groups of people and set lists of songs specific to each event. Future versions should handle communicating with people that are scheduled as well as managing a song database with music-related editing features.

This thesis documents the planning and implementing of this application and describes the various the technologies and tools which made it a reality. The project was implemented using the Eclipse programming environment on a Windows 7 machine utilizing the Java programming language together with the Vaadin framework to provide the client-server communication handling as well as user interface components.

The completed application utilized several libraries and packages in cooperation with Vaadin to enable this Java-based Rich Internet Application to meet the demands set forth for it by the client. The finished product will be configured in a production environment on a web server to enable global access to the client.

KEYWORDS:

Programming, web application, Java, Vaadin, database

TABLE OF CONTENTS

1 PROJECT OVERVIEW 1
1.1 Introduction 1
1.2 Background 1
1.3 Project Requirements 2
1.4 Technologies Used 3
1.5 Design 5
1.6 Testing and Validation 5
2 SYSTEM AND DEVELOPMENT TOOLS 7
2.1 Operating System 7
2.2 Preparation and Mockups 7
2.3 Java IDE and Tools 11
2.3.1 Eclipse IDE 11
2.3.2 Vaadin Plugin 11
2.3.3 Apache Tomcat 12
2.3.4 Gliffy 13
2.4 Database 13
3 IMPLEMENTATION 15
3.1 Programming Tasks 15
3.1.1 Business Logic 15
3.1.2 User Interface 18
3.1.3 Services 29
3.2 Database 34
3.2.1 Database Connectivity 34
3.2.2 Database Structure 36
4 CONCLUSIONS AND SUMMARY 38
REFERENCES 41
APPENDICES 42
1 Source Code 42
2 Database Configurations 105

ACRONYMS, ABBREVIATIONS AND SYMBOLS

AJAX	Asynchronous Javascript and XML
API	Application Protocol Interface
CRUD	Create / Read / Update / Delete
DB	Database
GUI	Graphical user interface
GWT	Google Web Toolkit
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IRC	Internet Relay Chat
JPA	Java Persistence API
JSF	JavaServer Faces
JVM	Java Virtual Machine
MVC	Model-View-Controller. An architectural pattern used to isolate “business logic” from user interface.
ORM	Object Relational Mapper
OS	Operating System
PDF	Portable Document Format
PHP	PHP: Hypertext Preprocessor
POJO	Plain Old Java Object
RIA	Rich Internet Application
SQL	Structured Query Language
MySQL	An SQL database technology. A subsidiary of Oracle

UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
WAMP	Windows, Apache, MySQL, PHP
XHTML	eXtensible HyperText Markup Language

1 Project Overview

1.1 Introduction

The goal of this project was to provide a system for a local church organization to assist their music teams and leadership in planning and arranging their weekly events. This system should be an all-in-one place to organize a variety of tasks such as: song selection, music editing, personnel scheduling, and communication. This would save time and confusion by having one single tool to provide all the functionality that previously took multiple environments to handle. Not only that, but automation of tasks involving music theory, such as automatic transposition of key, would increase productivity and decrease planning time on a weekly basis. This also takes away the need for storing multiple copies of a single song in the database, one for each desired key, as the single entry can be altered with a single click of the mouse.

It was decided to deploy this system as a web application rather than a local desktop program. This makes it possible for multiple users to access the information without the need to download or install any piece of software or plugins on their local machines, as all that is needed is a connection to the Internet. Many frameworks and tools exist to build such an application, but Vaadin (developed by Vaadin Ltd in Turku, Finland) suits the project requirements perfectly. Section 1.4 (Technologies Used) contains a thorough description of Vaadin and what it provides.

1.2 Background

Being a staff member of Calvary Chapel Turku, the church for which this application is being developed, made understanding the system in place, and the needs of this new application clear and simple. The problem points with the current team planning method included:

- Too many methods of communication
    - (Email, phone calls, SMS, Facebook…)
- Emailing attachments back and forth for any small change in the music plan

- Too much work by hand that could be automated and streamlined

Previous experience with the Java programming language was somewhat limited to only a few small projects as well as courses in the basics of Java and Swing. Swing is Java's primary Graphical User Interface (GUI) widget toolkit which can be used to build, for example, a desktop application. An introduction to Vaadin came through a one week intensive course, held at Åbo Akademi, where attendants were taught the very basics of how to create a web application from scratch using the Vaadin framework. Teams of two were then formed and given three to four days to create their own simple web applications. The fact that my own limited experience with Java and Vaadin demonstrates the usefulness of a framework like Vaadin in that it is not too difficult to learn. Vaadin is easy to pick up because it scales well to both small projects to learn the basics with, as well as large, professional corporate applications.

1.3 Project Requirements

The initial demands for the new application were reasonably limited as the actual needs and functionalities were few. A main request was for a sense of straightforwardness and that the program would be easy to use, rather than overcomplicated with excess components. The requirements for the first version were:

- Availability from any internet connected computer
- An archive of uploadable song sheets and tablature
- Creating events with: a date, band members assigned, a set list of songs
- Automatic transposing of a song's key by selecting the desired one (*2nd version*)
- Communication to/with team members (some email system) (*2nd version*)

Although the minimum requirements are few, ideas and possibilities for further development were discussed and could eventually lead to a much more comprehensive and feature rich environment. This could result in further development in the future which could perhaps make the application marketable as either a free or a paid system available to multiple organizations on the internet. This extension of the application could be very interesting and open up many possibilities. The author and the staff at Calvary Chapel Turku like the idea of providing others with a free, open source tool for anyone to use and benefit from. At the same time, if it is decided to

make some sort of “premium services”, this could provide a beneficial source of income.

1.4 Technologies Used

Java

The main programming language to be used for this project is Java. Java was developed by James Gosling of Sun Microsystems (now a subsidiary of Oracle) in 1995 as a truly object-oriented programming language. It shares some similarities with other languages, such as C++ in its object oriented approach and even syntax, but has perhaps a simpler object model. Java source code is saved as a `.java` file type and is usually compiled to a class file with a `.class` extension which can run on any Java Virtual Machine (JVM). Running class files on a JVM allows Java programs to be supported and run on almost any OS architecture including Windows, Linux, Mac OS, and UNIX. Whether developing a desktop or a web application, having fewer implementation dependencies can vastly improve productivity by allowing the same code to run a program on any one of these architectures.

Java is essentially a set of libraries that defines its syntax and functionality. Outside of its own libraries, Java can be accompanied by other libraries with additional software which can provide resources such as graphics, communication over networks, and database interaction. This allows Java to be easily extendable to fit the specific needs of a software project. In fact, libraries specifically for graphics and database interaction were added in order for this project function the way it should. (Lewis and Loftus 2001)

Vaadin

Alongside of Java itself, Vaadin was one of these libraries just mentioned and was the main technology that made this application possible. Vaadin is a Java framework for creating modern Rich Internet Applications (RIA). As it is a server-side Ajax web application development framework, it allows developers to build powerful web applications in the same manner as traditional desktop frameworks like AWT or Swing. The server-driven model of Vaadin means that the application code is run on the

server, but the user interaction is handled by a client-side engine running in the browser. This model provides several advantages. Firstly, the developer can focus on writing application logic instead of worrying about client-side technologies like HTML and Javascript. Since the client-side engine is built on GWT-based widgets and runs as Javascript in the web browser, there is no need to install plug-ins. Vaadin applications run out of the box on all Ajax-capable browsers. Second, the server-side architecture allows developers to use the power and flexibility of Java in web applications. Using Java's object-oriented programming also allows for creating easily extendable and maintainable applications. Running the code on the server also provides a layer of security and hides the application code from being seen. (Grönroos 2010, 2)

Not only does Vaadin handle all of the client-server communications and free the developer to focus on the application itself, but it is also a vast library of UI components (just as Swing provides in building desktop applications). These components, such as Buttons, TextFields, Tables, and Labels, comprise the user interface which utilizes event listeners and data bindings to communicate both with each other, as well as the actual Java application logic. Having a clear separation between the UI and "domain logic" allows a developer to implement a proper Model-View-Controller (MVC) design pattern as shown in Figure 1.1. Using an MVC architecture allows the program to be divided into separate, complete sections that can each be developed on their own. Perhaps the greatest advantage is that the application logic can be written and unchanged, while the physical view of the application can be altered to change the look and feel of the program.

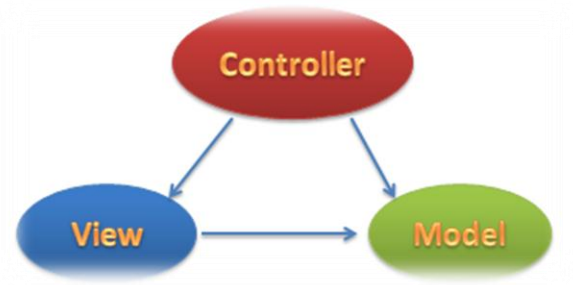

Figure 1.1. Simple MVC architecture design.

The focus of the Vaadin method can be simplified in three concepts. First, through the use of the built-in, out-of-the-box components and themes, Vaadin applications can have a great look and feel. However, developers also have access to add-on

components and themes, as well as the ability to create their own custom themes, to give even more options. Second, Vaadin Ltd strives to make their applications perform well and scale well to any size project. Performance is always important, especially with all the processing power in computers and smartphones today, as consumers want a product to work fast and reliably. Finally, productivity in application development is a major goal within the company. With a system like Vaadin that allows the programmer to concentrate the program itself and not all the client-server interactions, skilled programmers should be able to create entire projects from start to finish in a much smaller amount of time than was previously possible. Productivity and speed of development is actually one of the major goals for even more improvement at Vaadin in 2011.

The beauty of programming with Vaadin is that very few other tools than writing Java code is necessary. Section 2 (System and Development Tools) describes in more detail the other tools used, such as a Java IDE, server, and database system.

1.5 Design

The direction for the layout and design of the application was to achieve a clean, structured, and easy to understand interface. In planning for the design, research and comparisons were made with similar concept sites and applications, some of which contained powerful tools and options. However, the design and confusing layout of these applications deterred people at times from using their services due to a sense of frustration at the apparent complexity. Therefore, the overall design strategy should seek to achieve a simple, professional, and clean interface that is inviting to users. Mockups and design examples are shown later in Section 2.2 (Preparation and Mockups).

1.6 Testing and Validation

The testing phase was one area of this project that suffered, primarily due to a lack of time for proper testing. Naturally, during the process of developing an application like this, debugging and error correction occur every step of the way. All components and

aspects related to the application's use were tested during development and after completion of the project to ensure functionality.

2 System and Development Tools

2.1 Operating System

The programming tasks were carried out in a Microsoft Windows environment on the personal computer of the author. The particular version used was the currently newest version, Windows 7 64-bit. Vaadin is available and can be used on a number of different operating systems. They officially support the following OS types:

- Windows
- Linux
- Mac OS X Tiger or Leopard
- Other UNIX based operating systems, such as Sun Solaris

Regardless of OS type, Vaadin is supported on a wide array of all major web browsers. Since Vaadin is used to specifically create web applications, it is important that users are not limited to only one or two browsers that can access the applications made with it. As of the newest release of Vaadin, 6.6.1, the following web browser technologies are supported for application viewing:

- Mozilla Firefox 3 and 4
- Safari 4 and 5
- Google Chrome (latest version)
- Opera 10 and 11
- Internet Explorer 6, 7, 8, and 9

2.2 Preparation and Mockups

The Preparation phase was a larger and more integral part of this project than it would perhaps normally be. This is primarily due to the fact that, as previously stated, the author had little experience in Java programming techniques. Furthermore direct experience with Vaadin was limited to only a one week introductory project. Therefore,

considerable time was spent examining documentation on writing web apps with Vaadin.

Fortunately, their website (vaadin.com) is very well built and contains several key resources for developers. There are sample projects and tutorials varying from both small to medium sized projects which illustrate proper techniques. Of course, the entire API (Application Protocol Interface) is archived for referencing all Vaadin components and their classes and methods in the same way that any Java library API is indexed. Vaadin also hosts its own forum where questions, comments, and even bugs can be reported and discussed by anyone with a valid user-id and password (accounts are free). Perhaps the two most beneficial resources they have are the *Book of Vaadin* and their two chat channels. The book is currently in version 6.4 and was written by their own developer, Marko Grönroos, to provide an overview of Vaadin and explain all the integrated UI components, architecture, how to write and deploy a web application, and the Vaadin data model. This book is available in printed form and is freely available online or in PDF format. The website even contains functional examples with source code and working components that further demonstrate examples from the *Book of Vaadin*. They have both an IRC (Internet Relay Chat) channel as well as a Skype group where users can ask questions and get immediate feedback and assistance.

After reviewing the many resources from Vaadin's website and other real world example projects, the next step in preparation was to plan out the application architecture in an effort to maintain a type of MVC structure that can be built upon and expanded with little complications. Although the final structure did differ in small ways from the original design, planning these steps did help keep the code organized and the coding process clear. For example, it was decided that the coding procedure should be methodical by working on tasks in this order:

1. Create the "business logic"
    a. Models
2. Design layouts
    a. Main layout structure
    b. Views for adding and showing various components
3. Write the services and database classes

Taking the time to plan out the UI of the application benefited the coding process by having actual images of the design to refer to. Balsamiq (www.balsamiq.com, 2011) is an excellent tool for creating mockups and wireframe designs. It has both an online and a desktop application for quickly making mockups of UIs that can be specific for desktop, web app, or even mobile applications. These mockups can be made in a matter of minutes and they look great, which helps inspire good design and implementation. Figure 2.1 shows an example of what the main layout and UI structure would look like. The left side of the screen would have the logo and space for a calendar and possible other components, while the main layout contains a tabSheet where the bulk of all work is done. Figure 2.1 shows the People tab selected were team members can be added, edited, and deleted. Figure 2.2 displays the Songs tab with an open modal window with which to add and edit songs.

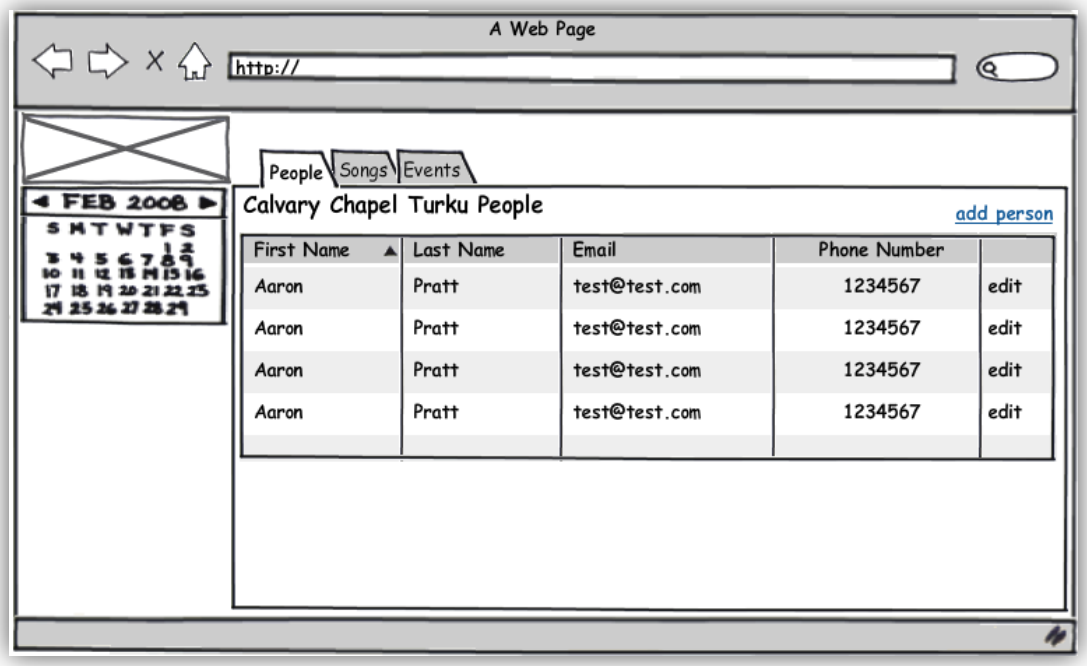

Figure 2.1. A mockup design of the main layout and “People View”.

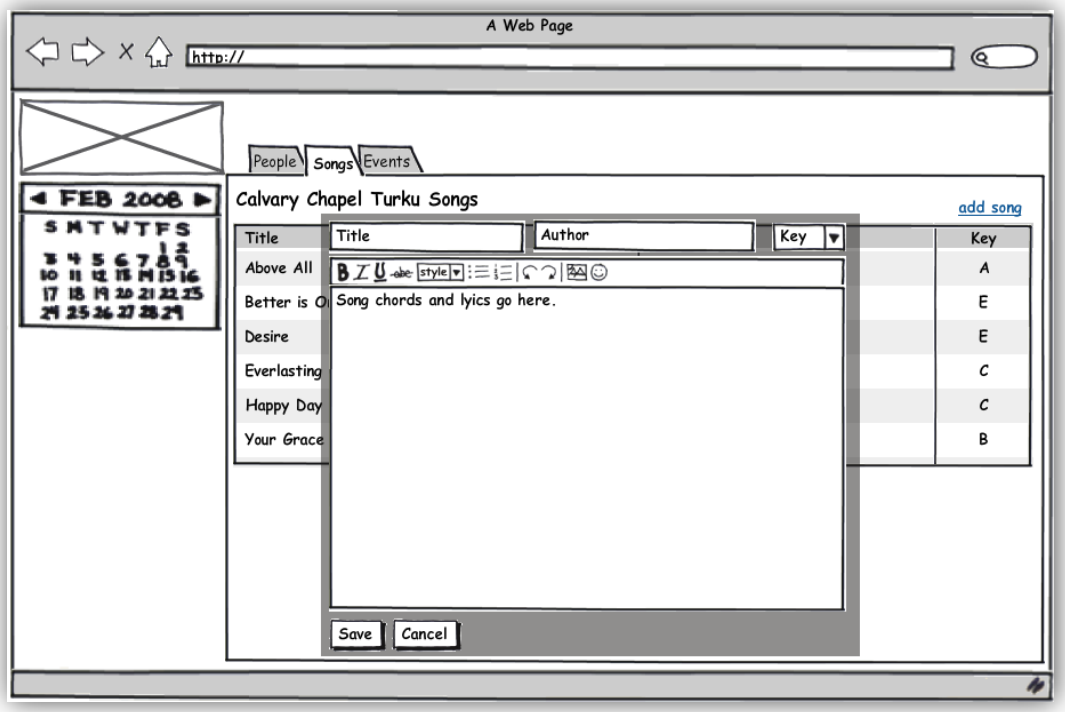

Figure 2.2. “Song View” with modal window for adding a new song.

Finally, Figure 2.3 displays a simple use case scenario in order to have a clear goal for what services and functionalities would be provided.

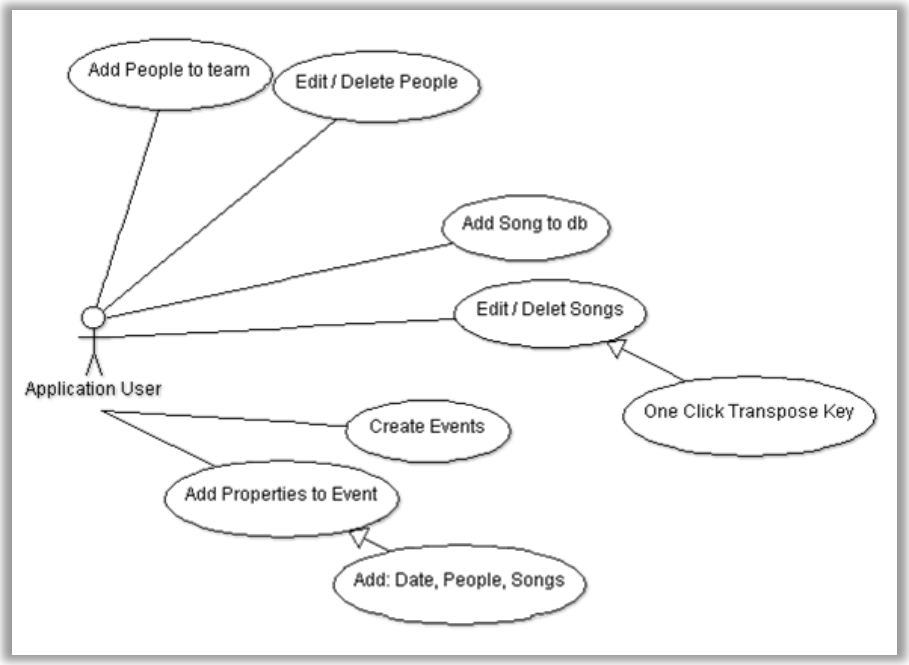

Figure 2.3. Use Case Diagram of functionality.

2.3 Java IDE and Tools

2.3.1 Eclipse IDE

The environment chosen for all programming tasks on this project was the open source tool suit Eclipse, specifically Eclipse IDE for Java EE Developers (Helios Version). Eclipse is a standalone application that, on Windows, does not require installation by an .exe file but is by itself a runnable program. The Enterprise Edition is a tool for Java EE and web application developers with a Java IDE (Integrated Development Environment) and tools for JPA (Java Persistence API), JSF (JavaServer Faces), and others.

Before being able to run Eclipse, or to run a Vaadin application inside of it, a Java runtime environment (JRE) must be installed on the development machine. If there is no JRE found, the Vaadin application will fail to start as the batch file will fail and close immediately (Grönroos 2009, 13). At this point, it is very important to make sure that the JRE and Eclipse versions are compatible with each other, namely that they are either both 32-bit versions, or both 64-bit versions. In this instance, the JDK 6.0 64-bit and Eclipse JEE 64-bit versions were chosen. It could be noted that in a 64-bit Windows 7 environment, either a 32 or 64-bit option could be chosen, as long as the same is used for both pieces of software.

2.3.2 Vaadin Plugin

Installing Vaadin is the next step to be able to create the application in Eclipse, and this is very simple and easy to do. Vaadin has created a plugin for both the Eclipse and NetBeans IDEs; therefore, it is easily integrated in either environment. Simply searching for Vaadin in Eclipse's "Install New Software" wizard will yield a window where all that is needed is to check the boxes and click install as shown in Figure 2.4. At the time this project began, 6.5.7 was the current stable release of Vaadin.

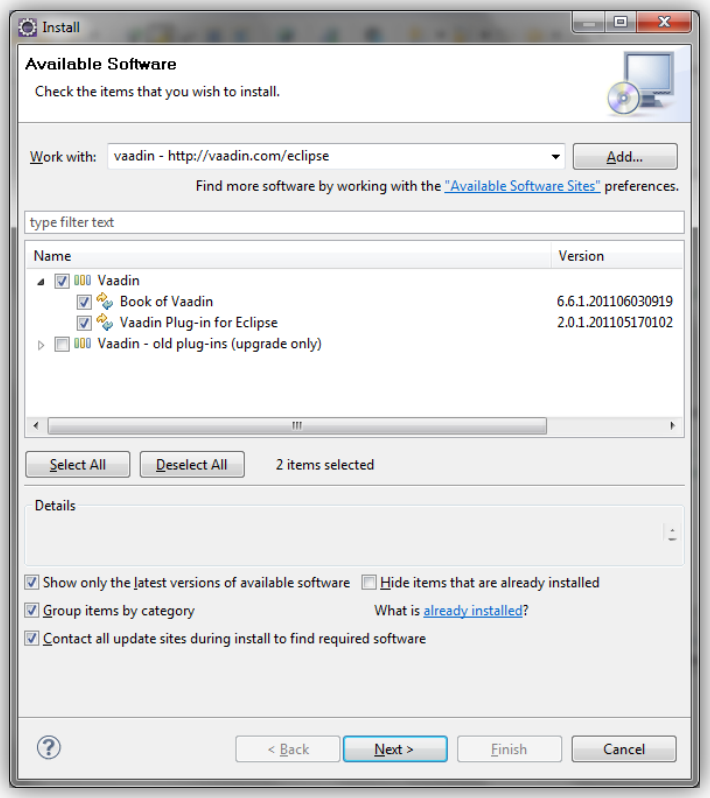

Figure 2.4. Installing the Vaadin Eclipse Plugin.

2.3.3 Apache Tomcat

In order to run the web application, the files must be served somehow. Though far from the only compatible option, perhaps the most popular server to use in Vaadin application development is Apache Tomcat. Tomcat is a lightweight Java server that can be used in both development and production environments. Tomcat is developed in an open and participatory environment and even powers many large-scale, mission-critical web applications despite being lightweight. (tomcat.apache.org, 2011)

For this project, Apache Tomcat version 6.0.32 was chosen as the development server. Prior use and familiarity showed that Tomcat was a dependable and powerful choice. Vaadin officially supports Tomcat version 4.1 or later.

2.3.4 Gliffy

A tool called gliffy was used to create all of the UML and Use Case diagrams for the project. Gliffy is an online design tool that is growing in popularity. It has customers in over 40 countries that use it for personal use, small businesses, academic studies, and even large global corporations, such as Cisco, Dell, Pandora, Pixar, Apple, Adobe, and even MIT. It is a versatile system that supports numerous diagram and chart types such as: flow charts, network diagrams, site map software, UML diagrams, and business process modeling software. (www.gliffy.com, 2011)

Some tools for drawing UML and database entity diagrams are over-complicated and difficult to understand, much less to implement. Gliffy is very straightforward and yields quick results that can be customized quite a bit to look great.

2.4 Database

Depending on the needs and functionalities of a web app, most production applications require the ability to store and retrieve data and information. That is why these types of apps can be referred to as CRUD applications (Create/Read/Update/Delete). Although several storage options exist for Vaadin applications, even a simple file system can be utilized; most often a database is the preferred method to fulfill this requirement.

As previously mentioned, Vaadin is currently making strides to make development even more streamlined and faster. Databases and data persistence is one of these areas. Most Vaadin professionals use some sort of Object Relational Mapper (ORM) such as Hibernate or JPA as a Java persistence framework. Using these tools simplifies database connectivity compared to writing traditional SQL queries, and can be a more productive method. As of 2011, Vaadin has now released a plugin for Spring Roo so that developers can take advantage of both Vaadin and Roo simultaneously which makes it possible to create a fully functioning, attractive, and data persistent application in a matter of hours or minutes. Explaining Roo is beyond the scope of this paper and will not be further mentioned.

It was decided that for this project, these newer methods would not be used relating to the database configuration. This decision was two-fold. First, as an academic assignment, writing traditional SQL was more reflective on the author's actual training

at TUAS (Turku University of Applied Sciences). The second and larger reason, however, was that learning a new technology like Hibernate or Roo did not fit into the timeline allowed by this project. It should be mentioned though that these technologies, especially Roo, show tremendous signs of inspiring developers as they open up a world of possibilities in application development with so much ease, once the technology has been learned.

MySQL was chosen as the database management system as it is widely used and documented and is reliable. During development, it was specifically used from inside WAMP (Windows, Apache, MySQL, PHP) due to the fact that WAMP was already installed from other software projects on the local machine and was ready to deploy. The only other requirement from WAMP was to create a new database in phpMyAdmin for this project. Deciding to go with these technologies also made sense as the final production server environment supplies MySQL databases with phpMyAdmin. Therefore, moving the database over to the production server would be compatible and straightforward.

3 Implementation

The implementation stage is where the bulk time and effort on this project was spent. Once Eclipse, along with the Vaadin plugin as well as the Tomcat server, is installed and set up, work could begin on building the actual application. At this point the theoretical information would become practical assignments deriving physical code and real world results.

As previously stated, a design method was in place in order to systematically approach the separate logical pieces of the application. This section will follow that chronological structure as close as possible in order to be a proper documentation of the coding process. Although the project attempted to maintain a proper MVC architecture, some structures may have overlapped to a degree, perhaps mostly due to lack of experience with Vaadin and GUI application development.

It should be stated that this document is not meant to be a full “tutorial” on how to create a Vaadin application from the ground up. The purpose is, however, to document and report the general procedures and results of this software development project. Therefore, small details or certain specific steps might not be mentioned here, as they will not be beneficial within the scope of this paper. Many tutorials and full instructions on writing Vaadin application can be found on their website, as was discussed in Section 2.2 on Preparation and Mockups.

3.1 Programming Tasks

3.1.1 Business Logic

Following the MVC pattern for software architecture, the first classes that need to be written are the “domain logic” components. This represents the **model** classes that will handle the behavior and data of the application. The models will respond to requests for an item’s, or POJO’s (Plain Old Java Object), state as well as requests to change that state. These requests come from the views (to display the state) and the controllers (to change the state).

Writing the model classes was a rather quick process, especially when compared with designing the UI and Controller classes. There is only a total of 3 entities needed to

fulfill the Use Case determined for this project: Happening, Person, and Song. In the event that further development will occur and the program will be extended to add more functionality, several more entities will be needed such as: Group and Instrument.

Java is defined as an object-oriented programming language. Although not the only "objects" used in this application, the model classes can be referred to as objects (or POJOs). As an example, each team member added to the application is stored as an instance of the Person class. Each Person is an object which can be viewed, edited, updated, and deleted. These interactions take place by means of the Person class' methods, such as constructors and the getters and setters. The same holds true for the Happening and Song classes.

```
// Constructor
public Person(int personId, String fName, String lName, String email,
String phoneNumber){
    this.personId = personId;
    this.fName = fName;
    this.lName = lName;
    this.email = email;
    this.phoneNumber = phoneNumber;
}
 //Getters and setters
public void setfName(String fName) {
    this.fName = fName;
}
public String getfName() {
    return fName;
}
```

From the above example, there is a getter and setter for a person's first name. In this case, the `setfName()` method is called upon by a controller class when creating a person, to store the name given in the database. The `getfName()` method is used by a view to retrieve the name and insert it into a table which displays all properties of all the people in the database (Fig 3.1).

FIRST NAME	LAST NAME	EMAIL	PHONE NUMBER		
instant	test	email		edit	delete
first name	last name	email	phone number	edit	delete

Figure 3.1. Example of table retrieving properties from getter methods.

The Song class is structured similarly with variable for a song's title, author, key, and lyrics with chords. There are two features of this entity that should be pointed out. First, the `songText` variable should be able to hold enough information comparable to around one to two pages of a typical rich text formatted document (MS Word for example). A `String` was used in the Java class and the data type used in the MySQL database for storing this data was a `LONGTEXT`.

```
private String songText;    // the mysql data type used is LONGTEXT
```

```
CREATE TABLE `songs` (
...
`song_text` LONGTEXT
)
```

The second important part of the Song class is `key` variable. This is the one property of a song that a user will not add any new data for. The `key` is set as a String variable which is set later in the SongView class as a predefined set of options that a user can choose from. This list of options provides all of the possible keys in which music can be played. The user has only to choose which key he/she will set the song in.

```
String[] available_keys = new String[]
{"C", "Db", "D", "Eb", "E", "F", "F#", "G", "Ab", "A", "Bb", "B"};
```

The demands from the model entities in this project were quite light, as it is a simple set up. Most of the "magic" and interesting features take place in the UI components and the controllers which tie the UI and models together. It is important though, no matter how large or small a project might be, to take special care in writing clean, error free model objects, as they are the building blocks so to speak of the entire application. One way of keeping track of the Java classes is through the use of UML (Unified Modeling Language) diagrams. Depending on the size of the project and the experience of the programmer, taking the time to draw-up UML diagrams might not always be beneficial. However, many developers are quite keen on always planning and tracking their work the proper way, and UMLs are an excellent way of documenting and doing just that. Figure 3.2 displays a UML diagram of the data package (`com.aaronpratt.teamplanner.data`) for the Team Planner Application.

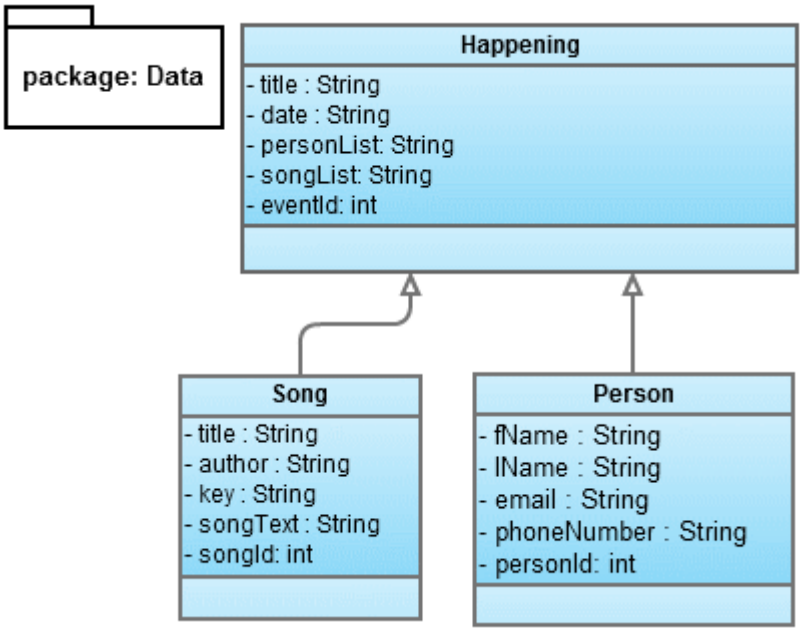

Figure 3.2. UML Diagram of the data package.

3.1.2 User Interface

The second aspect from the MVC architecture that needs to be implemented is the **View**. The view provides the user interface that is the only direct part of the program that end users will see and directly interact with. The models provided the “application logic” which is the foundation for a functional program, and the controllers will provide the interaction of those models together with the database. Not only that, but the controllers tie these functionalities together with the UI to make it a working graphical system. Therefore, the sole purpose of the view classes are to, as implied, provide an interface with which to view the data and resources inside the program. This means that the view should be separate from any “logic” and database transactions in order to maintain a true MVC structure. The reasons and advantage of programming this way is that no matter what the user interface looks like, the underlying structure never changes, but continues to work as it should. This means that changes can be made, or even an entirely new design of the GUI can be made which then ties together with the models and controllers, but should not require modifications to them.

The view classes do, however, contain some similarities to “logical components”, but these are not considered part of the “business logic” or controllers themselves and

should be understood correctly. The views implement methods from the models and controllers in order to retrieve data to be displayed by the UI. This is partially done using listeners, such as ClickEvents or ValueChangeEvents, which are actually methods of the UI components themselves. These methods provide functionality for the graphical components, instead of having a button that does nothing.

This section is where Vaadin really comes into place. The programming approach that was taken in this project, in practice meant that only Java itself was needed for the model classes; and Java together with SQL statements and JDBC connectors were utilized in the controllers. In essence, the only aspect Vaadin was directly used for was in implementing the GUI with its own graphical components, as well as a few add-ons from the Vaadin directory. In reality, however, Vaadin does so much more for the programmer. The difference between a desktop and web application is that a web app is run in a browser over the Internet. This means that there are many other aspects than just the UI that must be handled in order for the application to function remotely, such as AJAX communication between the browser and the server. Vaadin also removes the need for developers to learn and debug browser technologies like HTML and JavaScript. (Marko Grönroos 2009, 1-2)

Vaadin has always had a fairly powerful set of UI components for developers to utilize in application development. As anyone with experience in GUI development would probably know, sometimes getting the design, layout, and alignment set properly can be a cumbersome process. This is why Vaadin has introduced a tool to assist in the design and implementation of Vaadin applications called the Visual Editor. This is now a part of the package that comes with the Eclipse plugin. The visual editor is itself, in fact, a Vaadin application that allows graphical components and layouts to be arranged and edited in a drag-and-drop type manner. Almost all component attributes such as name, caption, alignment, size, and expand settings can be configured easily. The visual editor automatically compiles the actual code corresponding to the changes made in the “design view”. This code can be then altered by hand in the “code view”; however, that can lead to strange behavior when later attempting to use the design view again. The display window has gridlines and allows components to snap in place to keep them all properly aligned as shown in Figure 3.3.

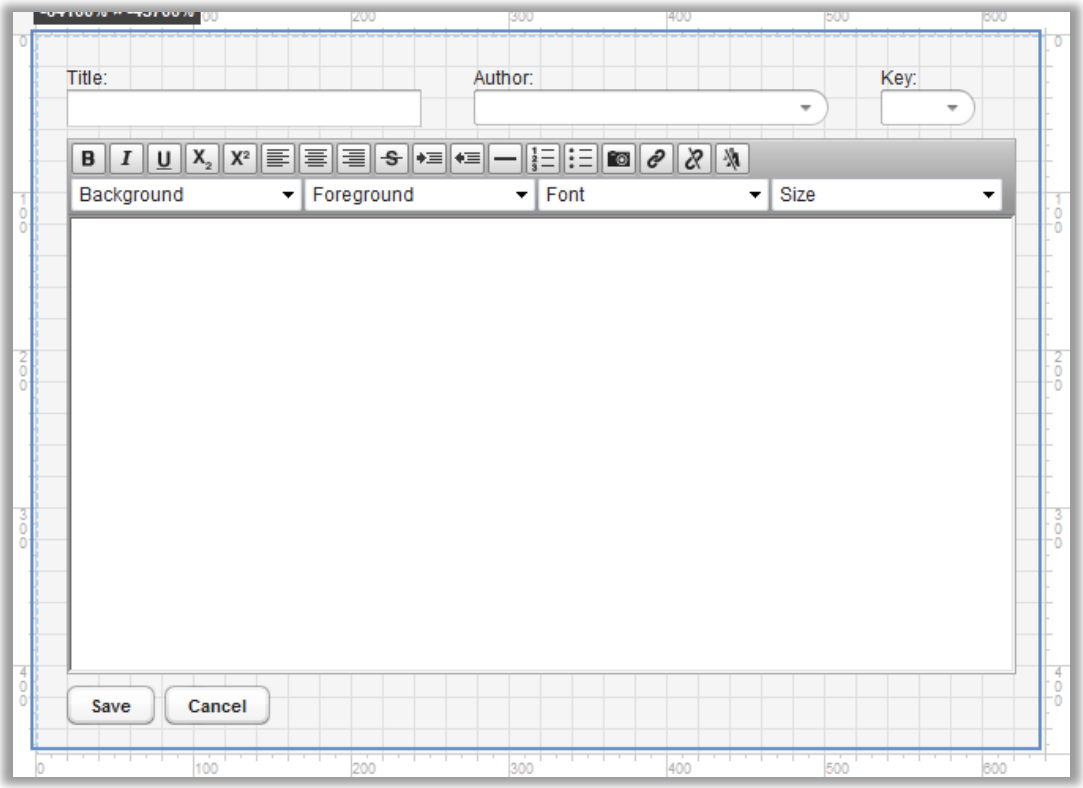

Figure 3.3. Layout editor of the Vaadin Visual Editor.

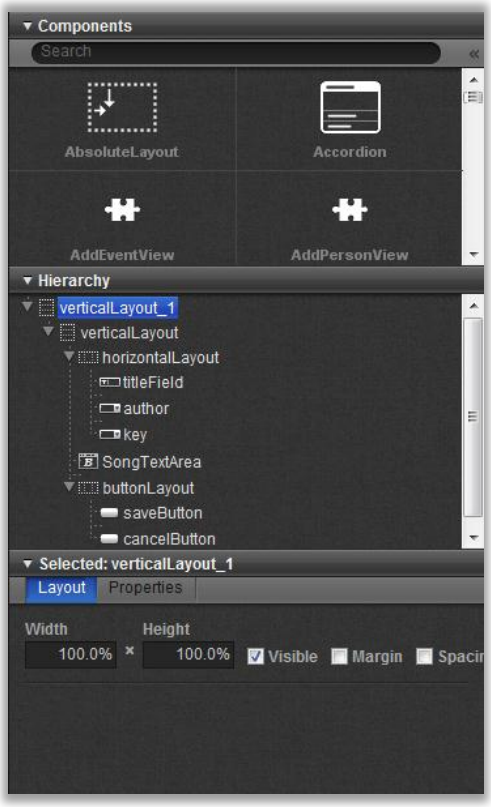

Figure 3.4. Component tree and property editor.

The starting point in any Vaadin application is the main application class itself. This is where everything begins and where content is loaded from. It is just like any other class with the exception that it must extend the `Application` class.

```
public class TeamPlannerApp extends Application
```

As far as the View is concerned, the only action that happens in this class is the loading of the main application layout. The `init` method accomplishes essentially two tasks. First, it sets the theme used in the application. It is very easy to change the overall theme used in a Vaadin application using one of the built in themes, an add-on theme, or even writing your own. Second, the main window is created and the content is set to load the `MainLayout` and the `HeaderLayout`. This could be done inside the `init` method, but a separate method is used so that in the next version of this application (which will have multiple user login capabilities) the login page will be force loaded first; and only after a successful login will the `loadProtectedResources` method be called.

```
public void init() {
    setTheme("reindeer");

    // Set main window
    mainWindow = new Window("Worship Team Planner");
    setMainWindow(mainWindow);
    loadProtectedResources();
}
// load the main application in the window
public void loadProtectedResources (){
    //layout to hold header and main layout
    mainLayoutHolder = new VerticalLayout();
    mainLayoutHolder.setStyleName(Reindeer.LAYOUT_WHITE);

    //create new header and main layout
    mainLayout = new MainLayout(this);
    headerLayout = new HeaderLayout(this);

    //add components to layout
    mainLayoutHolder.addComponent(headerLayout);
    mainLayoutHolder.addComponent(mainLayout);
    mainLayoutHolder.setSizeFull();
    mainLayoutHolder.setExpandRatio(mainLayout, 1);

    //set new window content
    mainWindow.setContent(mainLayoutHolder);
}
```

The main application layout consists of two classes, the MainLayout and the HeaderLayout. The header, as displayed in Figure 3.5, is a simple layout displayed as a thin bar on the top of the window containing the application logo and a label. In future versions that support user login, there will also be buttons such as “my account” and “logout”.

Figure 3.5. Header Layout

The rest of the application lies inside two classes, the MainLayout and the SideLayout. The SideLayout holds space for a calendar and will have more context menu options in the next release of this application. These additional options will be for adding detailed info about team members and songs. For example, additional information to help sort through songs such as: tempo, beat-per-minute, themes, and genre will be added. The MainLayout and SideLayout are divided using a fixed-in-place split panel as shown in the example code below (Note: this is a snippet example and therefore, does not contain all the parameters actually used).

```
// Add a horizontal split panel in the bottom area
splitPanel = new HorizontalSplitPanel();
splitPanel.setLocked(true);
addComponent(splitPanel);

// Add some content to the left side of split panel
sideBar = new SideLayout();
splitPanel.addComponent(sideBar);

// Add content to Right side of Split Panel
t = new TabSheet();
splitPanel.addComponent(t);
```

The MainLayout is essentially a tabSheet that provides navigation to separate views specific to dealing with people, songs, or happenings (events). Each of these views is a class on its own which has an instance created and added to the tabSheet. The MainLayout is displayed in Figure 3.6.

```
// Create component containers for tabs
peopleTab = new PeopleView();
songTab = new SongView();
happeningTab = new HappeningView();

// add tabs. call the 3 main views
t.addTab(peopleTab, "People", null);
```

```
t.addTab(songTab, "Songs", null);
t.addTab(happeningTab, "Events", null);
t.addListener(this);
```

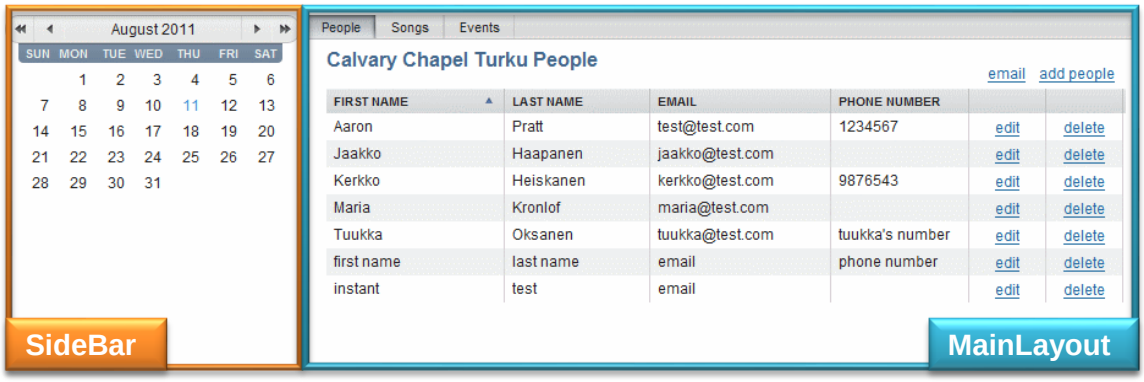

Figure 3.6. The main layout of the application

By separating the various components that make up the user interface as a whole it is easier to keep the code clean and organized, as well as easier to debug because it is less cluttered and confusing. This creates a larger number of Java files to deal with, but each one has fewer lines of code. The fewer lines to scroll through in any individual class the easier it becomes to locate specific functions and understand what is happening in the code. The structure of the Views section is shown in Figure 3.7. Two packages comprise the View architecture, the layouts and the views. The three classes that make up the layouts have just been explained; the views package will be examined next.

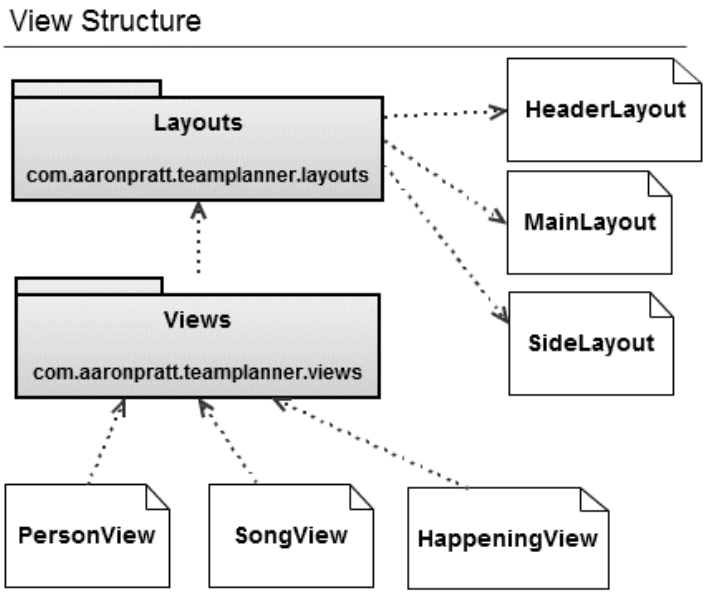

Figure 3.7. Layouts and View packages which form the UI

Each of the three views (PersonView, SongView, and HappeningView) shares the same basic design and functions. They provide four main operations: the ability to add content, view content, edit content, and delete content. The specific way that those functions are carried out varies slightly in each tab depending on the type of content and the UI components needed for the data.

The first tab, which is also the view displayed upon loading the application, displays the PeopleView content. The layout is simple so that it won't take hours of toiling with the application to figure out how to use it. There are basically three component areas in the view: a header label with a title for the menu page selected, buttons for initial actions, and a table to display the people currently saved in the system (Figure 3.8). The label is set to display content formatted as standard XHTML. So a standard header tag is used to give the label its formatting according to the current theme of the application.

```
headerLable = new Label("Calvary Chapel Turku People");
headerLable.setStyleName("h2");
```

Buttons can be styled in several ways in Vaadin; in this case a link style is used. Only two buttons are initially shown: the add people and email button. Additionally, the user can right click on any row and get a menu with options like deleting a person. The email button is not in this menu, but is displayed above because multiple rows can be selected and therefore all selected people can be sent an email to.

```
emailButton = new Button("email");
emailButton.setStyleName("link");

addPeopleButton = new Button("add people");
addPeopleButton.setStyleName("link");
```

Lastly, the table for displaying the people is set with a few qualifications. It is set to a maximum height of ten rows. When more than ten rows of data exist, it will automatically enable scrolling. The table is styled to be without a border to give it a sleeker look and feel. Selection and selecting multiple rows is enables as well as the ability to reorder the columns. Finally customized header names are given; otherwise the variable names would be used, such as `fName` instead of "First Name".

```
// peopleTable
peopleTable = new Table("", personBean);
peopleTable.setWidth("97.0%");
peopleTable.setPageLength(10);
peopleTable.setStyleName(Reindeer.TABLE_BORDERLESS);
```

```
peopleTable.setSelectable(true);
peopleTable.setMultiSelect(true);
peopleTable.setImmediate(true);
peopleTable.setColumnReorderingAllowed(true);
peopleTable.setSortAscending(true);
peopleTable.setSortContainerPropertyId("fName");

//column headers
peopleTable.setVisibleColumns(new Object[]{"fName", "lName", "email",
"phoneNumber", "edit", "delete"});
peopleTable.setColumnHeader("fName", "First Name");
peopleTable.setColumnHeader("lName", "Last Name");
peopleTable.setColumnHeader("phoneNumber", "Phone Number");
peopleTable.setColumnHeader("edit", "");
peopleTable.setColumnHeader("delete", "");
peopleTable.setColumnWidth("phoneNumber", 100);
peopleTable.setColumnWidth("edit", 50);
peopleTable.setColumnWidth("delete", 50);
peopleTable.setColumnAlignment("edit", Table.ALIGN_CENTER);
peopleTable.setColumnAlignment("delete", Table.ALIGN_CENTER);
```

The last two columns in the `peopleTable` are “generated columns”. These columns provide buttons with the ability to edit and delete a single table entry.

```
//edit button column
peopleTable.addGeneratedColumn("edit", new Table.ColumnGenerator() {
    private static final long serialVersionUID = 7190293891473776387L;
    public Component generateCell(Table source, Object itemId, Object
columnId) {
        //Item item = songTable.getItem(itemId);
        tableEditButton = new Button("edit");
        tableEditButton.setData(itemId);
        tableEditButton.setStyleName("link");
        tableEditButton.setImmediate(true);
        tableEditButton.addListener(new EditButtonListener());

        return tableEditButton;
    }
});
```

In order to get the data to be displayed in the table, a few simple steps must be taken. Since multiple items will be displayed, an `ArrayList` is created to hold these items and this list calls the controller method which selects all rows in the database `people` table. Then a `BeanItemContainer` is used as a container for the multiple beans (or POJOs) received from the database. This bean container is run inside a loop which adds one bean at a time to the container, complete with its person data values from the database. When the container has been filled from the DB, it is added to the table as the container data source.

```
db = new PersonService();
list = new ArrayList<Person>();
list=db.getAllPeople();

// Create a container for beans
personBean = new BeanItemContainer<Person>(Person.class);

// Add some beans to it
for(Person person:list){
    personBean.addBean(new Person( person.getPersonId(),
    person.getfName(), person.getlName(),
    person.getEmail(), person.getPhoneNumber()));
}

// peopleTable
peopleTable = new Table("", personBean);
```

Figure 3.8 shows these elements in the finished “People” tab, together with test data in the table.

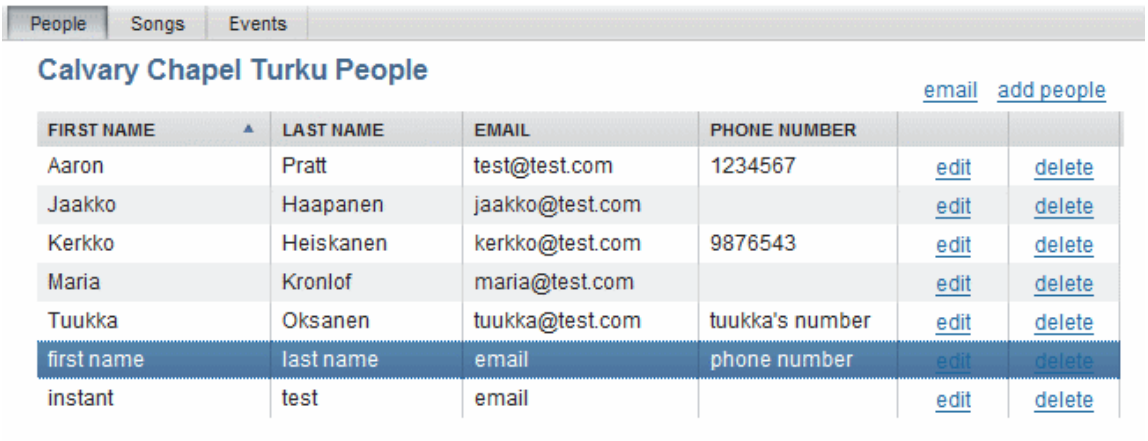

Figure 3.8. PeopleView basic layout.

When the user clicks on the “add people” button, a modal window pops up in the center of the screen with a form for creating a new person. The window can be closed by saving the information, clicking the cancel button, or by clicking the “x” in the top right corner (just as any standard application has a close icon). The input data is saved to the database by calling a controller method (similarly to how data was added to the table) from within the save button’s ClickListener. Finally, the modal window is closed by removing it from the main window.

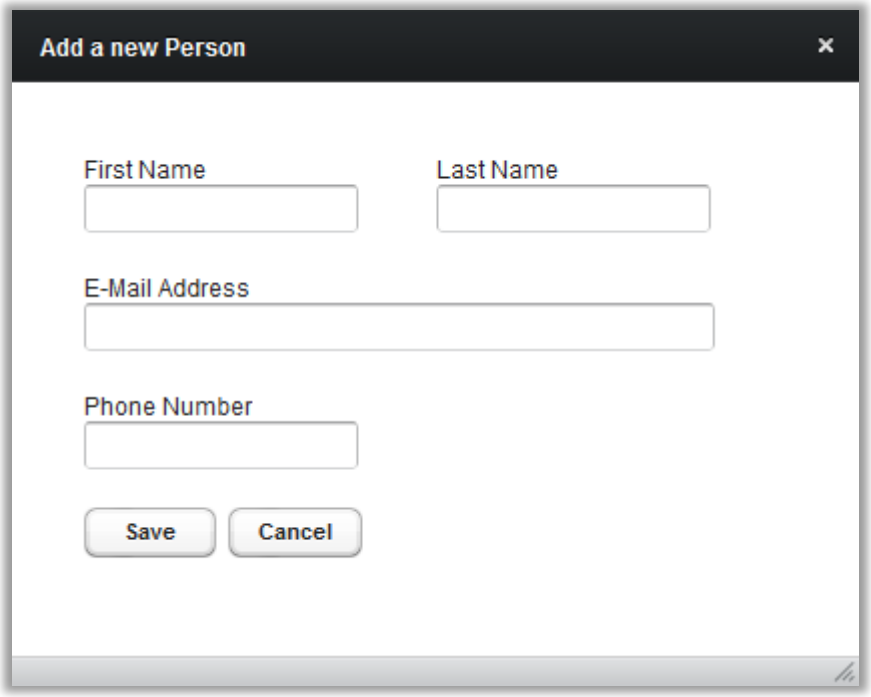

Figure 3.9. Modal window to add a person.

The “Songs” tab is almost identical in design to the “People” tab with a label, a few buttons, and a table to view the songs. A table is used to sort through the archived songs. This table shows the title, author, and key of a song, along with generated buttons for deleting or editing a song. A modal window is used again as an area for adding and editing songs. This window contains one of the more advanced built in Vaadin components, a fully functioning rich text editor. When the save button is clicked, its listener calls a controller method, passing the correct parameters to be stored in the database. The listener then removes the modal window from the display as shown in the code below.

```
db.createSong(title.getValue().toString(),
              author.getValue().toString(),
              keyBox.getValue().toString(),
              songTextArea.getValue().toString());

addSongSubwindow.getParent().removeWindow(addSongSubwindow);
```

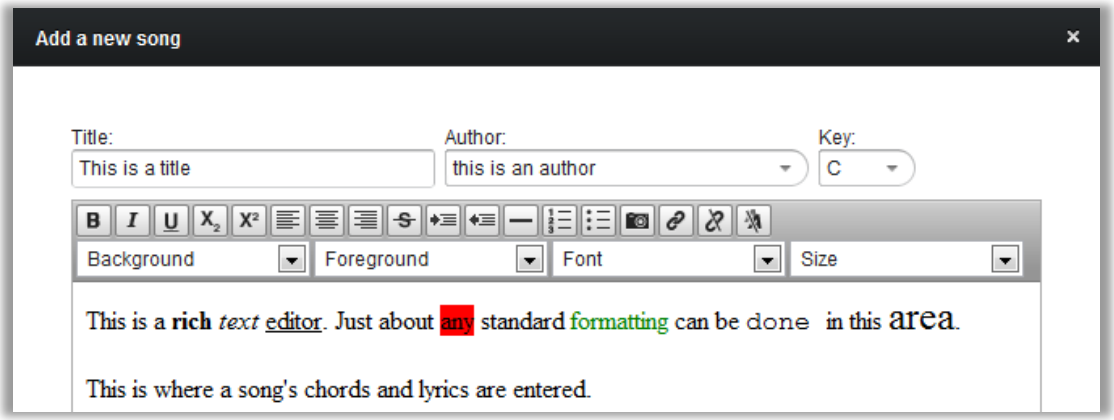

Figure 3.10. Modal window demonstrating how to add/edit a song.

The HappeningView shares the same basic approach as that of the PersonView and SongView; however, it has one main functional difference. The general format of a definition label, a button to add a new event, and a container of some sort to display events is used. Since an event has not only multiple properties (like a song has a title, author, key, and text), but some of these properties are themselves lists of objects (multiple songs and multiple people assigned to a single event). A normal table does not provide the type of functionality needed to display all this information. There could be several methods of solving the problem of how to display this information in a clean, organized fashion without a single event requiring a very large portion of the screen.

The choice used to solve this particular situation came in the form on a UI add-on component called a `TreeTable`. Although this component was developed by Vaadin and not a 3rd party developer, it is not yet implemented as a built-in component but is available as a free add-on. This is, exactly as it sounds, a combination of the tree structure and a table component. The table aspect provides a rich visual experience with a more defined separation of items, while the tree provides a hierarchical structure that can be minimized to show only the parent row (with the title and date, for example) or maximized to display all the songs and people currently added to that event. This component works nicely for the Happening table as a single event can have a subgroup of people and songs. These subgroups can be collapsed or expanded to take up as little or much space as needed as shown in Figure 3.11.

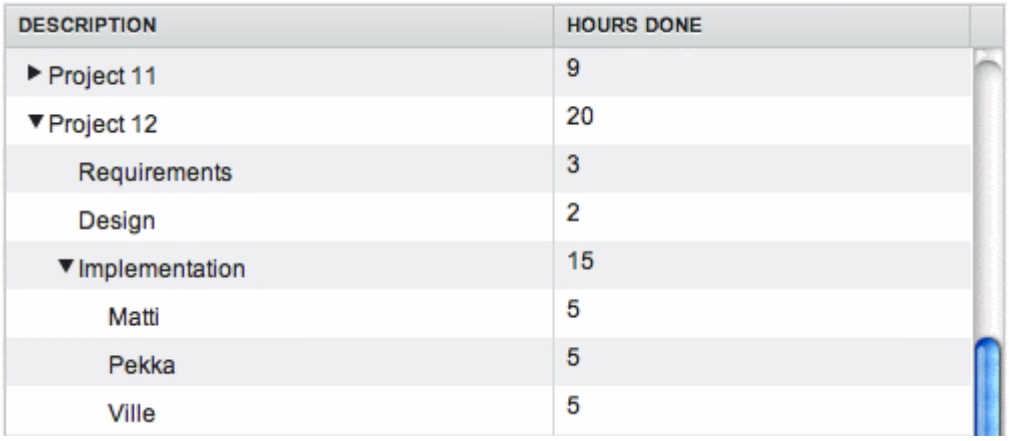

Figure 3.11. Example of a `TreeTable` component.

3.1.3 Services

The models have been written to provide the objects and base upon which to build the application, and the views have built the overall user interface. Now the final aspect of the software that is needed is the **controller**. These controller classes tie the models and views together, as well as provide the connection to the database where all application data will be stored.

The overall method design for the controllers is quite simple, and can be compared to the functionality of the “getters and setters” from the model classes. There is one controller class for each model entity. Each class has a series of methods which contain SQL queries for communicating with the database. Some of the methods add data to the database by utilizing `INSERT` or `UPDATE` commands. These methods can be thought of as “setters”, since they set values of certain properties which the DB then stores in its tables. In practice, this will be the method called from a form, for example, to add a new person instance to the team. Other methods request or “get” information back from the database through the use of `SELECT` statements. This data can then be called on from any UI component that can handle the specific content. For example, the table shown previously in Figure 3.1 uses one of these methods to request all the properties for all people instances in the DB and displays the people found in rows. This demonstrates the power and flexibility of using the Model-View-Controller architecture, as the actual queries only need to be written once and can be called multiple times from anywhere in the application interface.

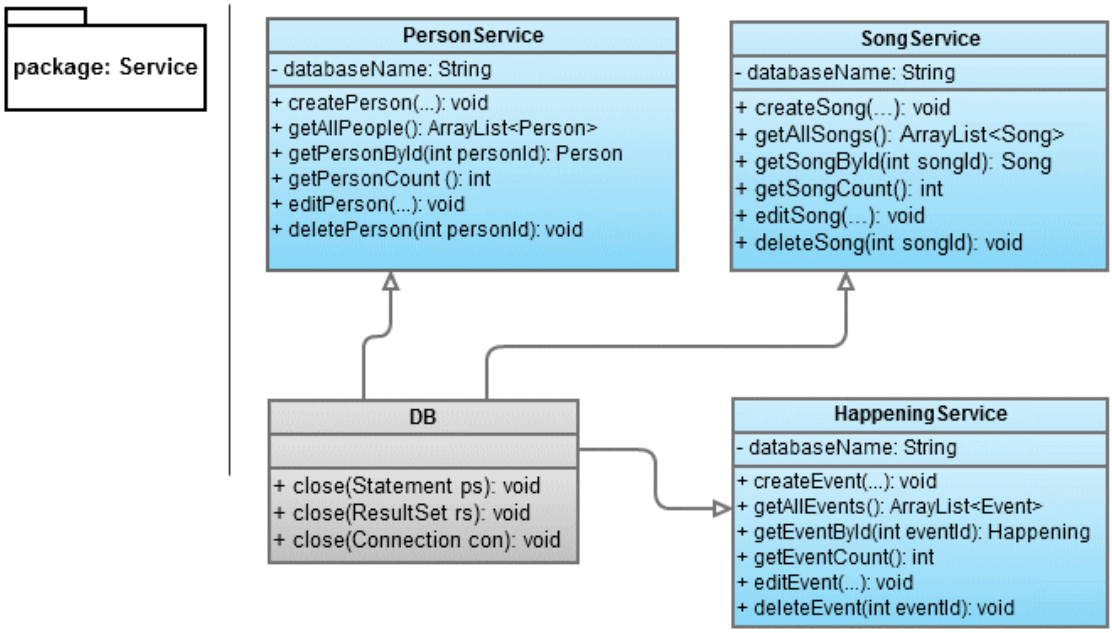

Figure 3.12. UML diagram of the service package.

The package for the controllers, `com.aaronpratt.teamplanner.services`, is structured as shown above. The first class written is the `DB.java` class which contains three methods. This class and its methods are called on by all the other three classes in the package. The methods provide a streamlined way to disconnect from the database. The reason for doing this is because every method that requests or inserts data from the database needs to first connect to the DB, execute the statement, and then disconnect. Instead of rewriting the code to sever the DB connection in every one of these methods, that logic can be stored in the `DB` class methods and simply called on whenever disconnection is needed.

```
public static void close(Statement ps){
    if(ps!=null){
        try{
            ps.close();
        }catch (SQLException ignore){
        }
    }
}
```

The code above is one such method. This method is used to close a prepared statement, which can be thought of similar to a `String` variable which holds some amount of written data. All queries used in the service package use PreparedStatements as a type of wrapper for the actual SQL query (a detailed explanation on PreparedStatements and JDBC is in section 3.2.1 on Database Connectivity). Statements are used to execute queries, ResultSets are used to retrieve

data (in the case of a `SELECT` statement), and all database transactions take place within a `Connection` to the DB. All three of these must be properly closed, which is done by calling these methods. ResultSet and Connection closing methods are as follows:

```
public static void close(ResultSet rs){
    if(rs!=null){
        try{
            rs.close();
        }catch (SQLException ignore){
        }
    }
}
public static void close(Connection connection) {
    if (connection!=null){
        try {
          connection.close();
        } catch (SQLException ignore) {
        }
    }
}
```

The PersonService class provides all the controllers related to interactions with people instances. This, as well as the SongService and HappeningService classes, has only one local variable which performs a similar functionality as the DB class did. This String variable is merely for storing the database connection data (URL, user-id, password) to help simplify all the database transactions. Instead of writing out the full URL with the database username and password within every method which makes a transaction, the variable name, in this case: `databaseName`, can be called instead.

```
private final static String databaseName =
    "jdbc:mysql://localhost:3306/team_planner?user=root&password=";
```

The next example shows the method for adding a new person to the database. The parameters in this method are all the properties of a Person instance which the user has entered into the form, in this case those properties are first name, last name, email, and phone number. The first task to be done is to access the JDBC driver (the JAR file library add-on that provides connection to the database). Next, a Connection variable is made which creates the DB connection, calling the String `databaseName` with connection data. Now that a connection is made, the SQL query can be written and stored inside a PreparedStatement. Since the actual values are unknown until the user inputs them in the UI, question marks are used which are then filled, in order, with user

input data as defined by the `setString()` statements. At this point, all the information has been stored in the PreparedStatement, but is not actually executed in the database until it is specified to do so by the `executeUpdate()` statement. The final task is to properly close the database connection, which process was just described. The query statement is closed first, followed by the actual connection to the database.

```
//create a person
public void createPerson(String fName, String lName, String email,
String phone){
    try {
        //Access driver from JAR file
        Class.forName("com.mysql.jdbc.Driver").newInstance();

        //create variable for db connection
        Connection con = DriverManager.getConnection(databaseName);

        //create a query
        PreparedStatement st = con.prepareStatement
("INSERT INTO people (fName, lName, email, phone) VALUES(?,?,?,?);");

        //set input data to values
        st.setString(1, fName);
        st.setString(2, lName);
        st.setString(3, email);
        st.setString(4, phone);

        //execute the statement
        st.executeUpdate();

        //close transaction
        DB.close(st);
        DB.close(con);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Although persistence frameworks, such as Hibernate, assist developers by demanding less for code in order to obtain the same result, this process is still quite straightforward and easy to understand. At least in a smaller project, such as this one, where the number of different DB transactions is somewhat limited, using the approach shown here is not very time-demanding.

Retrieving data back from the database in order to display it in the UI is done in a very similar manner as writing to the DB, with only a few minor adjustments. The method should not be `void` in this case; but it should have a return type, as something is being retrieved in order to be shown. If only a single specific table entry is being

requested, there will still be a method parameter in order to distinguish which entry is being requested; but in the example below we are asking for all the entries from the 'people' database table, therefore no such specification is needed. An `ArrayList` is created to hold all the people instances. The database connection and SQL statement are written the same way as the previous example; followed by a ResultSet which executes the query and then runs inside a `while` loop, adding the data to the person instances. When the loop has completed the connections must be closed again, this time starting with the ResultSet and then proceeding as before.

```
//get all people
public ArrayList<Person> getAllPeople(){
    ArrayList<Person> people = new ArrayList<Person>();
    try {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection(databaseName);
        PreparedStatement st = con.prepareStatement("SELECT * FROM
people");
        ResultSet rs = st.executeQuery();
        while (rs.next()){
            people.add(new Person(rs.getInt(1), rs.getString(2),
                                  rs.getString(3), rs.getString(4),
                                  rs.getString(5)));
        }
        DB.close(rs);
        DB.close(st);
        DB.close(con);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return people;
}
```

The HappeningService and SongService classes are structured the exact same way as these examples from the PersonService class have displayed. Once the database itself is set up and the connection has been successfully made from the Java files, the controllers themselves are very simple to implement. The power and effectiveness of the Controllers and the Models, while themselves quite simple to implement, displays the beauty and advantages of Java's object oriented approach.

3.2 Database

3.2.1 Database Connectivity

Since it was decided not to implement Hibernate or another similar ORM, direct SQL statements were used together with JDBC (Java Database Connectivity) as the connection driver. JDBC was developed in the 1990s by Sun Microsystems with the intent of being a "standard for data access on the Java Platform" (Jesse Davis 2010, 1). At that time it was merely a thin API that ran on top of an ODBC (Open Database Connectivity) driver. Today it has been extended and rebuilt to be a fully featured standalone data access standard and has virtually replaced the need for ODBC at all. Through the years, JDBC has become much easier to deploy and utilize despite its growth in power and features, because it is but a single JAR file to be added in the IDE. Figure 3.13 illustrates these features and all that is under the hood of the JDBC driver.

Figure 3.13. 'Anatomy of the JDBC driver' (Jesse Davis 2010, 4)

If a persistence layer like Hibernate or JPA is not used in tandem with JDBC, it is still possible, of course, to implement a more direct SQL statement approach. One way to execute SQL queries with this approach is to use PreparedStatements. There are,

naturally, several other options when using Java and JDBC, but a PreparedStatement is often the most common choice for two reasons. First, if there are optional parameters which need to be specified to the SQL statement or values that do not convert easily to a string, such as a `BLOB`, a PreparedStatement handles these scenarios best. Also, it can help defend against attacks such as SQL injection when working with string values. Both of these benefited this particular project, as was described in the section on services and controllers (3.1.3). (Jesse Davis 2010, 1)

Once the database is configured, which will be covered in the next section, setting up the connection requires only a few simple steps. The first step in configuring JDBC is to download the driver which is freely available on MySQL's website under 'MySQL Connector/J'. The JAR file (in this case: `mysql-connector-java-5.1.16-bin.jar`) needs to be added to the project's build path in Eclipse as shown in figure 3.14. At this point Vaadin requires the widget-set to be recompiled, which should be prompted for automatically or can also be forced manually by simply clicking the Vaadin plugin button.

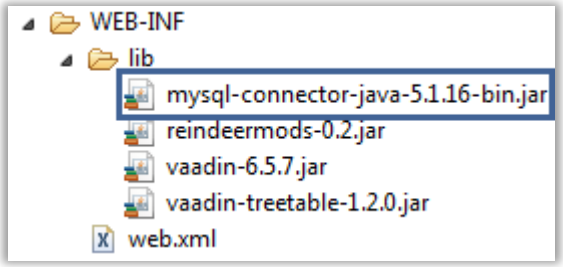

Figure 3.14. JAR libraries added in this project. JDBC driver is highlighted.

Once these steps are complete, the only task left is to initialize the connection in the java source code as was shown in configuring the controllers. It should be noted that the URL used in this example to connect to the database is pointing to the local machine (denoted by `localhost`); hence this is still during the development phase of the project. Once this application is finished and in a production environment, this URL should be changed to reflect the location of the actual production database.

```
//Access driver from JAR file
Class.forName("com.mysql.jdbc.Driver").newInstance();

// create variable for db connection
Connection con = DriverManager.getConnection(
     "jdbc:mysql://localhost:3306/testdb", "root", "root");
```

3.2.2 Database Structure

The structure of the database needed for the application reflects very closely the data package and its model entities. The design approach was found by asking a few simple questions. What are the requirements of the project? What does it need to do? What 'objects' are actually dealt with? This led to the same three main results as the models: Happenings, People, and Songs should be the core elements that need to be dealt with. Therefore, the database was separated into three tables as displayed in Figure 3.15.

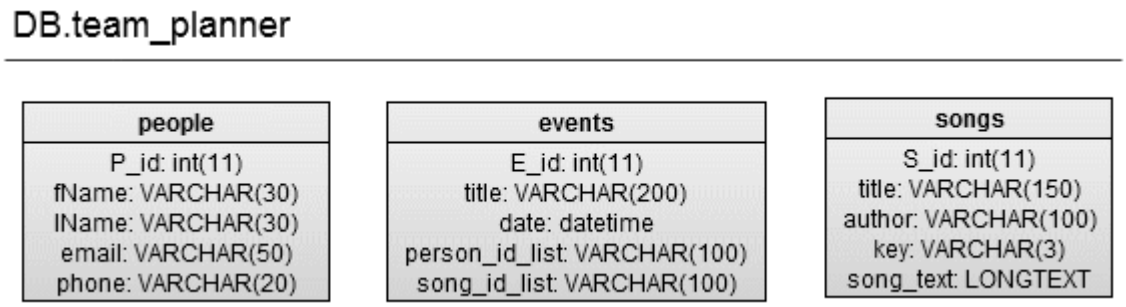

Figure 3.15. UML diagram for the team_planner application's database.

Following good practice, data types were chosen as well as reasonable max values in order to control and maintain the size of the tables. Each table contains fields that match up to the variables of the corresponding model. For example, a person instance built from the Person class has an id, first and last name, an email address, and a phone number. Therefore, the 'people' table has fields for those five attributes as well; the same is true for the 'events' and 'songs' tables.

The events table in the database contains two fields that provide the ability to link people and songs to a specific event. The `person_id_list` and `song_id_list` store a list of numbers as simple String values. These numbers are the ids associated to people and song objects. When an event is created which contains people and songs, a list of the ids is created and stored in these instead of the actual names of people and songs. When the event is called and displayed in the Happenings table, those ids are read from the database and the proper names and song titles are shown.

The 'songs' table has a few characteristics that are worth mentioning. The title and author fields are standard and self-explanatory. The 'key' field has a data type of

VARCHAR with a max length of 3. Since the content for what keys can be used are set as a `final static String`, no actual modification to this variable will occur but will be hard written in to the program. From a musical standpoint, there are seven notes in any given scale; and there are seven major keys to choose from (C, D, E, F, G, A, B). If each of these keys is thought of as having a full step between them, with the exception of after E and after B, there is then a half step between each one. For example, instead of actually counting: G, A, B… it would look like this: G, Ab, A, Bb, B… where the Ab and Bb (pronounced 'A flat') are the half steps. This means that the maximum length of any 'key' option will only be two characters long. An actual max length of '3' was chosen because it can be good practice to allow slightly more space than the absolute minimum theoretical requirement. The Last field, 'song_text', is where the lyrics and chord structures are stored in the database. A type of LONGTEXT is used to make sure that enough space is available as the length of any given song can vary. On average, most songs' lyrics should fill between one to two pages of a typical rich text style document, and at max require three to even four pages. The LONGTEXT data type should meet these requirements.

4 Conclusions and Summary

Looking back over the course of this project and the three main phases it took (planning/design, implementing, documenting), there are several things I have learned about working on software projects as well as about programming technologies. Although implementation is typically the most time demanding and largest part of a project, taking the time for proper R&D and planning of the application is a highly valuable procedure. The process of developing any application will always require some amount of learning and working out new methods during the programming itself; however, planning out exactly what the needs of the application will be and how they might be solved with what technologies can help make the coding process less confusing and flow more smoothly.

Coding with Java can be very methodical and practical in task-solving. A strong grasp of the utilized technologies always make a project run smoother; however, few developers know everything and will need some source of input and ideas. As my experience prior to this project with Java was very limited, I would strongly put myself in this category. Especially in the beginning of the project, tasks took me much longer than they would take even now, as I have a better grasp on the principles. It is safe to say that with a little more experience, I would expect to be able to solve programming tasks and future applications, for example, with Vaadin, much faster. The best way to learn is by doing.

As of the publication of this thesis document, the application failed to meet all of the needs and goals set out by the requirements and requested features. Most of the main structure and functionalities of the application are intact, with a few features not yet fully functional. Of the three main sections of the application, the People and Songs tabs are complete and fully functional. The Event tab has still some bug-fixing to do as the `TreeTable` component proved to be fairly complicated to deploy. Because of the time that was required to debug and complete the main functions of the application and the time frame allowed for this project, the two extra utilities, the transpose and email utilities were not able to be included in the first version of this application.

Based on the problems mentioned above, the program could be considered as a failed project. Because the demands set forth by the church organization were open to

flexibility, together with the likelihood of continued development past the bounds of this thesis project, the application can be seen as not a complete failure but as still in development.

As of the current configuration at the date of this publication, the entire web application consists of 3648 lines of code throughout 5 packages and 15 separate class files. Other than the native Vaadin UI components, two add-ons were used: the JDBC driver for the MySQL connection and the TreeTable add-on component.

After working on this project, I have decided that in future applications and even in further development of this application, it will be highly beneficial to learn a proper ORM like Hibernate to handle the actual data CRUD actions. Even better than Hibernate would be to learn how to utilize Spring Roo in a web application as it streamlines the object relational mapping so well that it can literally take only minutes to have a fully functioning simple application.

There are a few aspects of Vaadin I would point out based on my experience. The Visual Editor is a great tool that helps to quickly put together and have a clear view in mind (literally) of the desired layout of the application. In my personal experience, it seems that the visual editor is not yet perfected, but maybe has a few bugs in it yet. This is, of course, natural that any complex software would not be completely bug-free. Outside of a few strange behaviors, the editor did help speed up the UI development, which is exactly what it is meant for.

Another focus point would be Vaadin’s data model. Since almost any advanced application will deal with data structures, the developer will need to utilize the data model (properties, items, containers). Although much of it is simple to understand and use, for someone with little experience, parts of it can be difficult to grasp. Careful study through the Book of Vaadin and the many example and forums available from vaadin.com is a valuable use of time.

This project has served a useful purpose as I seek to learn more and enter into the IT field, specifically application development. It has provided me with more experience than any other project so far and will hopefully be a springboard into future projects and opportunities.

REFERENCES

Davis J. 2010. Unbreakable Data Access for Any Application: Performance, Functionality, and Reliability for Enterprise Applications. NC, USA: DZone, Inc. Also available at http://refcardz.dzone.com.

Grönroos M. 2009. Book of Vaadin. 6.2 Edition. Turku: OY IT Mill Ltd.

Grönroos M. 2010. Getting Started with Vaadin. NC, USA: DZone, Inc. Also available at http://refcardz.dzone.com.

Lewis J.; Loftus W. 2001. Java Software Solutions: Foundations of Program Design. Boston, Massachusetts: Addison-Wesley.

2001. Balsamiq Studios, LLC. Consulted 3.6.2011 http://balsamiq.com/.

2011. Gliffy, Inc. Consulted 8.6.2011 http://www.gliffy.com/.

1999-2011. The Apache Tomcat Foundation. Consulted 6.6.2011 http://tomcat.apache.org/.

APPENDICES

1 Source Code

```
//TeamPlannerApp.java

package com.aaronpratt.teamplanner;

import com.aaronpratt.teamplanner.layouts.HeaderLayout;
import com.aaronpratt.teamplanner.layouts.LoginLayout;
import com.aaronpratt.teamplanner.layouts.MainLayout;
import com.aaronpratt.teamplanner.layouts.RegistrationLayout;
import com.vaadin.Application;
import com.vaadin.service.ApplicationContext;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.Window;
import com.vaadin.ui.themes.Reindeer;

/**
 * The Application's "main" class
 */
public class TeamPlannerApp extends Application implements
ApplicationContext.TransactionListener {

    private static final long serialVersionUID = -
6342939979093918711L;
    private static ThreadLocal<TeamPlannerApp> currentApplication =
new ThreadLocal<TeamPlannerApp>();
    Window mainWindow;
    String currentUser;
    private LoginLayout loginLayout;
    private RegistrationLayout regLayout;
    private HeaderLayout headerLayout;
    private MainLayout mainLayout;
    private VerticalLayout mainLayoutHolder;


    @Override
    public void init() {
        //setTheme("runo");
        setTheme("reindeer");

        setCurrent(this);
        if (getContext() != null) {
            getContext().addTransactionListener(this);
        }

        // Set main window
        mainWindow = new Window("Worship Team Planner");
        setMainWindow(mainWindow);

        loadProtectedResources();
        // Login required, set login layout for main window
```

```
        //mainWindow.setContent(new LoginLayout(this));

    }

    /***
     * load the main application in the window
     */
    public void loadProtectedResources (){
        //layout to hold header and main layout
        mainLayoutHolder = new VerticalLayout();
        mainLayoutHolder.setStyleName(Reindeer.LAYOUT_WHITE);
        //create new header and main layout
        mainLayout = new MainLayout(this);
        headerLayout = new HeaderLayout(this);

        //add components to layout
        mainLayoutHolder.addComponent(headerLayout);
        mainLayoutHolder.addComponent(mainLayout);
        mainLayoutHolder.setSizeFull();
        mainLayoutHolder.setExpandRatio(mainLayout, 1);

        //set new window content
        mainWindow.setContent(mainLayoutHolder);
          //addListener(mainLayout.getHeader());
//.getUserChangeListener());
          //setUser(user);
    }

    /***
     * load the registration layout in the window
     */
    public void loadRegistration(){
        regLayout = new RegistrationLayout(this);
        mainWindow.setContent(regLayout);
    }

    /***
     * load the login layout in the window
     */
    public void loadLogin(){
        loginLayout = new LoginLayout(this);
        mainWindow.setContent(loginLayout);
    }

    /**
     * @return the current application instance
     */
    public static TeamPlannerApp getCurrent() {
        return currentApplication.get();
    }

    /**
     * Set the current application
     */
    public static void setCurrent(TeamPlannerApp application) {
        if (getCurrent() == null) {
            currentApplication.set(application);
        }
    }
```

```
    public void removeCurrent() {
        currentApplication.remove();
    }
}
```

```
//Happening.java

package com.aaronpratt.teamplanner.data;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

/**
 * Model class
 * @author Aaron Pratt
 */
public class Happening implements Serializable{

    private static final long serialVersionUID = -
1821842964693261145L;
    private String title;
    private String date;
    private String personList;
    private String songList;
    private List<PersonHappening> peopleInEvent = new
ArrayList<PersonHappening>();
    private int eventId;

    /**
     * Constructors
     */
    public Happening(int eventId, String title, String date, String
personList, String songList){
        this.eventId = eventId;
        this.title = title;
        this.date = date;
        this.personList = personList;
        this.songList = songList;
        //addPeopleToEvent(people);
    }
    public Happening(int eventId, String title, String date){
        this.eventId = eventId;
        this.title = title;
        this.date = date;
    }
    public Happening(){

    }

    // method to add multiple people to single event instance
```

```
    private void addPeopleToEvent(List<Person> people) {
        for(Person person : people){
            peopleInEvent.add(new PersonHappening(person));
        }
    }

    // getters and setters
    public String getDate() {
        return date;
    }
    public void setDate(String date){
        this.date = date;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title){
        this.title = title;
    }
    public int getEventId(){
        return eventId;
    }
    public void setEventId(int eventId){
        this.eventId = eventId;
    }
    public String getPersonList(){
        return personList;
    }
    public void setPersonList(String personList){
        this.personList = personList;
    }
    public String getSongList(){
        return songList;
    }
    public void setSongList(String songList){
        this.songList = songList;
    }


    public List<Person> getPerson() {
        List<Person> people = new ArrayList<Person>();
        for(PersonHappening personEvent : peopleInEvent){
            people.add(personEvent.getPerson());
        }
        return people;
    }

    public boolean isValid() {
        for(PersonHappening personEvent : peopleInEvent){
            if(!personEvent.isValid()){
                System.out.println("Error: PersonEvent for person " +
personEvent.getPerson().getfName()+"
"+personEvent.getPerson().getlName() + " was invalid!");
                return false;
            }
        }
        return true;
    }
}
```

```
//Person.java

package com.aaronpratt.teamplanner.data;

import java.io.Serializable;

/**
 * Model class
 * @author Aaron Pratt
 *
 */
public class Person implements Serializable {

    private static final long serialVersionUID = 4428060321751014797L;
    String fName;
    String lName;
    String email;
    String phoneNumber;
    int personId;
    String fullName;

    /**
     * Constructor
     * @param fName
     * @param lName
     * @param email
     * @param phoneNumber
     */
    public Person(int personId, String fName, String lName, String
email, String phoneNumber){
        this.personId = personId;
        this.fName = fName;
        this.lName = lName;
        this.email = email;
        this.phoneNumber = phoneNumber;
    }
    public Person(int personId, String fName, String lName, String
email){
        this.personId = personId;
        this.fName = fName;
        this.lName = lName;
        this.email = email;
    }
    public Person(){

    }

    public Person(String fName) {
        this.fName = fName;
    }
    //Getters and setters
```

```
    public void setfName(String fName) {
        this.fName = fName;
    }
    public String getfName() {
        return fName;
    }
    public void setlName(String lName) {
        this.lName = lName;
    }
    public String getlName() {
        return lName;
    }
    public void setEmail(String email){
        this.email = email;
    }
    public String getEmail(){
        return email;
    }
    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }
    public String getPhoneNumber() {
        return phoneNumber;
    }
    public int getPersonId(){
        return personId;
    }
    public String getFullName(){
        fullName = fName + " " + lName;
        return fullName;
    }

}
```

```
//PersonHappening.java

package com.aaronpratt.teamplanner.data;

public class PersonHappening {

        private Person person;

        public PersonHappening(Person person) {
            this.person = person;
        }
        public Person getPerson(){
            return person;
        }
        public boolean isValid() {
            // TODO Auto-generated method stub
            return false;
        }
```

```
}
```

```
//Song.java

package com.aaronpratt.teamplanner.data;

import java.io.Serializable;

public class Song implements Serializable {

    private static final long serialVersionUID = -
4636644268111161376L;
    String title;
    String author;
    String key;
    String songText;    // the mysql data type used is LONGTEXT
    int songId;


    /***
     * Constructors
     */
    public Song (int songId, String title, String author, String key,
String songText){
        this.title = title;
        this.author = author;
        this.key = key;
        this.songText = songText;
        this.songId = songId;
    }
    public Song (int songId, String title, String author, String key){
        this.title = title;
        this.author = author;
        this.key = key;
        this.songId = songId;
    }
    public Song() {
        // TODO Auto-generated constructor stub
    }


    //Getters and Setters
    public void setTitle(String title) {
        this.title = title;
    }
    public String getTitle() {
        return title;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public String getAuthor() {
```

```
        return author;
    }
    public void setKey(String key) {
        this.key = key;
    }
    public String getKey() {
        return key;
    }
    public void setSongText(String songText) {
        this.songText = songText;
    }
    public String getSongText() {
        return songText;
    }
    public int getSongId(){
        return songId;
    }
    public void setSongId(int songId){
        this.songId = songId;
    }

}
```

```
//HeaderLayout.java

package com.aaronpratt.teamplanner.layouts;

import com.aaronpratt.teamplanner.TeamPlannerApp;
import com.vaadin.annotations.AutoGenerated;
import com.vaadin.terminal.ThemeResource;
import com.vaadin.ui.AbsoluteLayout;
import com.vaadin.ui.Alignment;
import com.vaadin.ui.Button;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.CustomComponent;
import com.vaadin.ui.Embedded;
import com.vaadin.ui.HorizontalLayout;
import com.vaadin.ui.Label;

public class HeaderLayout extends CustomComponent {

    @AutoGenerated
    private AbsoluteLayout mainLayout;
    @AutoGenerated
    private Label headerLabel;
    @AutoGenerated
    private HorizontalLayout horizontalLayout;
    @AutoGenerated
    private Button logoutButton;
    @AutoGenerated
    private Button myAccountButton;
    @AutoGenerated
```

```
    private Embedded logoImage;
    private static final long serialVersionUID = 731143019429023080L;
    private TeamPlannerApp app;


    /**
     * The constructor should first build the main layout, set the
     * composition root and then do any custom initialization.
     *
     * The constructor will not be automatically regenerated by the
     * visual editor.
     */
    public HeaderLayout(TeamPlannerApp app) {
        buildMainLayout();
        setCompositionRoot(mainLayout);

        // TODO add user code here
        this.app = app;
    }

    @AutoGenerated
    private AbsoluteLayout buildMainLayout() {
        // common part: create layout
        mainLayout = new AbsoluteLayout();

        // top-level component properties
        setWidth("100.0%");
        setHeight("75px");

        // logoImage
        logoImage = new Embedded();
        logoImage.setWidth("242px");
        logoImage.setHeight("75px");
        logoImage.setCaption("Logo");
        logoImage.setImmediate(false);
        logoImage.setSource(new ThemeResource("common/img/logo.png"));
        mainLayout.addComponent(logoImage, "top:0.0px;left:0.0px;");

        // horizontalLayout
        //horizontalLayout = buildHorizontalLayout();
        //mainLayout.addComponent(horizontalLayout,
"right:9.0px;bottom:5.0px;");

        // headerLabel
        headerLabel = new Label();
        headerLabel.setWidth("-1px");
        headerLabel.setHeight("-1px");
        headerLabel.setStyleName("h2");
        headerLabel.setValue("CALVARY CHAPEL FINLAND");
        headerLabel.setContentMode(3);
        headerLabel.setImmediate(false);
        mainLayout.addComponent(headerLabel,
"top:5.0px;right:9.0px;");

        return mainLayout;
    }

    @AutoGenerated
    private HorizontalLayout buildHorizontalLayout() {
```

```
        // common part: create layout
        horizontalLayout = new HorizontalLayout();
        horizontalLayout.setWidth("-1px");
        horizontalLayout.setHeight("26px");
        horizontalLayout.setImmediate(false);
        horizontalLayout.setMargin(false);
        horizontalLayout.setSpacing(true);

        // myAccountButton
        myAccountButton = new Button();
        myAccountButton.setWidth("-1px");
        myAccountButton.setHeight("-1px");
        myAccountButton.setCaption("my account");
        myAccountButton.setStyleName("small");
        myAccountButton.setImmediate(true);
        horizontalLayout.addComponent(myAccountButton);
        horizontalLayout.setComponentAlignment(myAccountButton, new
Alignment(
                48));

        // logoutButton
        logoutButton = new Button();
        logoutButton.setWidth("-1px");
        logoutButton.setHeight("-1px");
        logoutButton.setCaption("logout");
        logoutButton.setStyleName("small");
        logoutButton.setImmediate(true);
        horizontalLayout.addComponent(logoutButton);
        horizontalLayout.setComponentAlignment(logoutButton, new
Alignment(34));

        return horizontalLayout;
    }
}
```

```
//MainLayout.java
package com.aaronpratt.teamplanner.layouts;

import com.aaronpratt.teamplanner.TeamPlannerApp;
import com.aaronpratt.teamplanner.views.HappeningView;
import com.aaronpratt.teamplanner.views.PeopleView;
import com.aaronpratt.teamplanner.views.SongView;
import com.vaadin.terminal.Sizeable;
import com.vaadin.ui.HorizontalSplitPanel;
import com.vaadin.ui.TabSheet;
import com.vaadin.ui.TabSheet.SelectedTabChangeEvent;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.themes.Reindeer;

@SuppressWarnings("serial")
public class MainLayout extends VerticalLayout implements
TabSheet.SelectedTabChangeListener {
```

```
    private HorizontalSplitPanel splitPanel;
    private SideLayout sideBar;
    private PeopleView peopleTab;
    private SongView songTab;
    private HappeningView eventTab;
    private TabSheet t;
    private TeamPlannerApp app;

    public MainLayout(TeamPlannerApp app) {
        this.app = app;

        this.setSizeFull();
        setMargin(false);

        // Add a horizontal split panel in the bottom area
        splitPanel = new HorizontalSplitPanel();
        splitPanel.setStyleName(Reindeer.SPLITPANEL_SMALL);
        splitPanel.setSplitPosition(242, Sizeable.UNITS_PIXELS);
        splitPanel.setSizeFull();
        splitPanel.setLocked(true);
        addComponent(splitPanel);

        // Add some content to the left side of split panel
        sideBar = new SideLayout();
        splitPanel.addComponent(sideBar);

        // Create component containers for tabs
        peopleTab = new PeopleView();
        songTab = new SongView();
        eventTab = new HappeningView();

        // create tabsheet
        t = new TabSheet();
        t.setStyleName(Reindeer.TABSHEET_SMALL);
        t.setSizeFull();

        // add tabs. call the 3 main views
        t.addTab(peopleTab, "People", null);
        t.addTab(songTab, "Songs", null);
        t.addTab(eventTab, "Events", null);
        t.addListener(this);

        // Add content to Right side of Split Panel
        //rightArea.setSizeFull();
        splitPanel.addComponent(t);

    }

    // tabsheet event listener
    public void selectedTabChange(SelectedTabChangeEvent event) {
      /*TabSheet tabsheet = event.getTabSheet();
        Tab tab = tabsheet.getTab(tabsheet.getSelectedTab());
        if (tab != null) {
            getWindow().showNotification("Selected tab: " +
tab.getCaption());
        }
      */
    }
```

```
}
```

```
//SideLayout.java
package com.aaronpratt.teamplanner.layouts;

import java.text.DateFormat;

import com.vaadin.annotations.AutoGenerated;
import com.vaadin.data.Property;
import com.vaadin.data.Property.ValueChangeEvent;
import com.vaadin.ui.CustomComponent;
import com.vaadin.ui.InlineDateField;
import com.vaadin.ui.VerticalLayout;


public class SideLayout extends CustomComponent implements
Property.ValueChangeListener {


    @AutoGenerated
    private VerticalLayout mainLayout;
    @AutoGenerated
    private InlineDateField calendar;
    private static final long serialVersionUID = 2243788466153753382L;
    /**
     * The constructor should first build the main layout, set the
     * composition root and then do any custom initialization.
     *
     * The constructor will not be automatically regenerated by the
     * visual editor.
     */
    public SideLayout() {
        buildMainLayout();
        setCompositionRoot(mainLayout);

        // TODO add user code here
    }

    @AutoGenerated
    private VerticalLayout buildMainLayout() {
        // common part: create layout
        mainLayout = new VerticalLayout();

        // top-level component properties
        setWidth("242px");
        setHeight("100.0%");

        // calendar
        calendar = new InlineDateField();
        calendar.setWidth("-1px");
        calendar.setHeight("-1px");
```

```
        calendar.setImmediate(true);
        calendar.addListener(this);

        // Set the value of the PopupDateField to current date
        //calendar.setValue(new java.util.Date());

        // Set the correct resolution
        calendar.setResolution(InlineDateField.RESOLUTION_DAY);

        mainLayout.addComponent(calendar);

        return mainLayout;
    }

    public void valueChange(ValueChangeEvent event) {
        // Get the new value and format it to the current locale
        DateFormat dateFormatter =
DateFormat.getDateInstance(DateFormat.SHORT);
        String dateOut =
dateFormatter.format(event.getProperty().getValue());
        // Show notification
        getWindow().showNotification("Selected date: " + dateOut);
    }

}
```

```
//DB.java

package com.aaronpratt.teamplanner.service;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * database connector class:
 * @author aaronandsarap
 */
public class DB {


    /**
     * close the current statement
     * @param ps Statement
     */
    public static void close(Statement ps){
        if(ps!=null){
            try{
                ps.close();
```

```
            }catch (SQLException ignore){
            }
        }
    }

    /**
     * close the current resultset
     * @param rs
     */
    public static void close(ResultSet rs){
        if(rs!=null){
            try{
                rs.close();
            }catch (SQLException ignore){
            }
        }
    }

    /**
     * Close the current database connection
     * @param connection
    */
    public static void close(Connection connection) {
        if (connection!=null){
            try {
              connection.close();
            } catch (SQLException ignore) {
            }
        }
    }

}
```

```
//HappeningService.java

package com.aaronpratt.teamplanner.service;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;

import com.aaronpratt.teamplanner.data.Happening;

public class HappeningService {

    private final static String databaseName =

"jdbc:mysql://localhost:3306/team_planner?user=root&password=";

    //create an event happening
```

```
    public void createEvent(String title, String date, String
personList, String songList) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("INSERT INTO
events (title, date, person_id_list, song_id_list) VALUES(?,?,?,?)");
            st.setString(1, title);
            st.setString(2, date);
            st.setString(3, personList);
            st.setString(4, songList);
            st.executeUpdate();
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    //get all event happenings
    public ArrayList<Happening> getAllEvents(){
        ArrayList<Happening> events = new ArrayList<Happening>();
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("SELECT * FROM
events");
            ResultSet rs = st.executeQuery();
            while (rs.next()){
                events.add(new Happening(rs.getInt(1),
rs.getString(2), rs.getTimestamp(3).toString()));    //,
rs.getString(4), rs.getString(5)));
            }
            DB.close(rs);
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return events;
    }

    public Happening getEventById(int eventId){
        Happening theEvent = new Happening();
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("SELECT * FROM
events WHERE E_id = ?");
            st.setInt(1, eventId);
            ResultSet rs = st.executeQuery();
            rs.next();
            theEvent = new Happening(rs.getInt(1), rs.getString(2),
rs.getTimestamp(3).toString(), rs.getString(4), rs.getString(5));
```

```
            DB.close(rs);
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return theEvent;
    }

    public int getEventCount(){
        int count = 0;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("SELECT
MAX(E_id) FROM events");
            ResultSet rs = st.executeQuery();
            rs.next();
            count = rs.getInt(1);
            DB.close(rs);
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return count;
    }

    //update an event happening
    public void editEvent(String eventId, String title, String date,
String personList, String songList){
        try{
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("UPDATE events
SET title = ?, date = ?, person_id_list = ?, song_id_list = ? WHERE
E_id = ?");
            st.setString(1, title);
            st.setString(2, date);
            st.setString(3, personList);
            st.setString(4, songList);
            st.setString(5, eventId);
            st.executeUpdate();
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    //delete an event happening
    public void deleteEvent(int eventId){
        try{
            Class.forName("com.mysql.jdbc.Driver");
```

```
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("DELETE FROM
events WHERE E_id = ?");
            st.setInt(1, eventId);
            st.executeUpdate();
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

```
//PersonService.java

package com.aaronpratt.teamplanner.service;

import java.sql.*;
import java.util.ArrayList;
import com.aaronpratt.teamplanner.data.Person;

/**
 * database connector class for person model
 * @author aaronandsarap
 */
public class PersonService {

    private final static String databaseName =

"jdbc:mysql://localhost:3306/team_planner?user=root&password=";


    //create a person
    public void createPerson(String fName, String lName, String email,
String phone){
        try {
            //Access driver from JAR file
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            //create variable for db connection
            Connection con =
DriverManager.getConnection(databaseName);
            //create a query
            PreparedStatement st = con.prepareStatement("INSERT INTO
people (fName, lName, email, phone) VALUES(?,?,?,?)");
            //set input data to values
            st.setString(1, fName);
            st.setString(2, lName);
            st.setString(3, email);
            st.setString(4, phone);
            //execute the statement
```

```
            st.executeUpdate();
            //close transaction
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    //get all people
    public ArrayList<Person> getAllPeople(){
        ArrayList<Person> people = new ArrayList<Person>();
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("SELECT * FROM
people");
            ResultSet rs = st.executeQuery();
            while (rs.next()){
                //people.addAll(people);
                people.add(new Person(rs.getInt(1), rs.getString(2),
rs.getString(3), rs.getString(4), rs.getString(5)));
            }
            DB.close(rs);
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return people;
    }

    // get a single person
    public Person getPersonById(int personId){
        Person thePerson = new Person();
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("SELECT * FROM
people WHERE P_id = ?");
            st.setInt(1, personId);
            ResultSet rs = st.executeQuery();
            rs.next();
            thePerson = new Person(rs.getInt(1), rs.getString(2),
rs.getString(3), rs.getString(4), rs.getString(5));
            DB.close(rs);
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return thePerson;
    }
```

```
    public int getPersonCount(){
        int count = 0;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("SELECT
MAX(P_id) FROM people");
            ResultSet rs = st.executeQuery();
            rs.next();
            count = rs.getInt(1);
            DB.close(rs);
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return count;
    }

    //update a person's info
    public void editPerson(String personId, String fName, String
lName, String email, String phone){
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("UPDATE people
SET fName = ?, lName = ?, email = ?, phone = ? WHERE P_id = ?");
            st.setString(1, fName);
            st.setString(2, lName);
            st.setString(3, email);
            st.setString(4, phone);
            st.setString(5, personId);
            st.executeUpdate();
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    //delete a person
    public void deletePerson(int personId){
        try{
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("DELETE FROM
people WHERE P_id = ?");
            st.setInt(1, personId);
            st.executeUpdate();
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
```

```
        }
    }
}
```

```
//SongService.java

package com.aaronpratt.teamplanner.service;

import java.sql.*;
import java.util.ArrayList;
import com.aaronpratt.teamplanner.data.Song;

public class SongService {

    private final static String databaseName =

"jdbc:mysql://localhost:3306/team_planner?user=root&password=";


    //create a song
    public void createSong(String title, String author, String key,
String songText){
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("INSERT INTO
songs (title, author, songKey, songText) VALUES(?,?,?,?)");
            st.setString(1, title);
            st.setString(2, author);
            st.setString(3, key);
            st.setString(4, songText);
            st.executeUpdate();
            DB.close(st);
            DB.close(con);
        }catch (Exception e){
            e.printStackTrace();
        }
    }

    //get all songs for table
    public ArrayList<Song> getAllSongs(){
        ArrayList<Song> songs = new ArrayList<Song>();
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("SELECT * FROM
songs ORDER BY title");
            ResultSet rs = st.executeQuery();
            while (rs.next()){
```

```
                songs.add(new Song(rs.getInt(1), rs.getString(2),
rs.getString(3), rs.getString(4)));
            }
            DB.close(rs);
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return songs;
    }

    public Song getSongById(int songId){
        Song theSong = new Song();
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("SELECT * FROM
songs WHERE S_id = ?");
            st.setInt(1, songId);
            ResultSet rs = st.executeQuery();
            rs.next();
            theSong = new Song(rs.getInt(1), rs.getString(2),
rs.getString(3), rs.getString(4), rs.getString(5));
            DB.close(rs);
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return theSong;
    }

    public int getSongCount(){
        int count = 0;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("SELECT
MAX(S_id) FROM songs");
            ResultSet rs = st.executeQuery();
            rs.next();
            count = rs.getInt(1);
            DB.close(rs);
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return count;
    }

    //update a song
```

```
    public void editSong(String songId, String title, String author,
String key, String songText){
        try{
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("UPDATE songs
SET title = ?, author = ?, songKey = ?, songText = ? WHERE S_id = ?");
            st.setString(1, title);
            st.setString(2, author);
            st.setString(3, key);
            st.setString(4, songText);
            st.setString(5, songId);
            st.executeUpdate();
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    //delete a song
    public void deleteSong(int songId){
        try{
            Class.forName("com.mysql.jdbc.Driver");
            Connection con =
DriverManager.getConnection(databaseName);
            PreparedStatement st = con.prepareStatement("DELETE FROM
songs WHERE S_id = ?");
            st.setInt(1, songId);
            st.executeUpdate();
            DB.close(st);
            DB.close(con);
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

```
//HappeningView.java

package com.aaronpratt.teamplanner.views;

import java.text.DateFormat;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;
```

```
import com.aaronpratt.teamplanner.data.Person;
import com.aaronpratt.teamplanner.data.Song;
import com.aaronpratt.teamplanner.data.Happening;
import com.aaronpratt.teamplanner.service.HappeningService;
import com.aaronpratt.teamplanner.service.PersonService;
import com.aaronpratt.teamplanner.service.SongService;

import com.vaadin.annotations.AutoGenerated;
import com.vaadin.data.Container;
import com.vaadin.data.Property;
import com.vaadin.data.Property.ValueChangeEvent;
import com.vaadin.data.util.BeanItem;
import com.vaadin.data.util.BeanItemContainer;
import com.vaadin.event.DataBoundTransferable;
import com.vaadin.event.dd.DragAndDropEvent;
import com.vaadin.event.dd.DropHandler;
import com.vaadin.event.dd.acceptcriteria.AcceptAll;
import com.vaadin.event.dd.acceptcriteria.AcceptCriterion;
import com.vaadin.event.dd.acceptcriteria.And;
import com.vaadin.event.dd.acceptcriteria.ClientSideCriterion;
import com.vaadin.event.dd.acceptcriteria.Not;
import com.vaadin.event.dd.acceptcriteria.SourceIs;
import com.vaadin.terminal.gwt.client.ui.dd.VerticalDropLocation;
import com.vaadin.ui.AbsoluteLayout;
import com.vaadin.ui.AbstractSelect.AbstractSelectTargetDetails;
import com.vaadin.ui.AbstractSelect.VerticalLocationIs;
import com.vaadin.ui.Alignment;
import com.vaadin.ui.Button;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.Component;
import com.vaadin.ui.CustomComponent;
import com.vaadin.ui.HorizontalLayout;
import com.vaadin.ui.Label;
import com.vaadin.ui.PopupDateField;
import com.vaadin.ui.Table;
import com.vaadin.ui.Table.TableDragMode;
import com.vaadin.ui.TextField;
import com.vaadin.ui.TwinColSelect;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.Window;
import com.vaadin.ui.Window.Notification;
import com.vaadin.ui.themes.Reindeer;

public class HappeningView extends CustomComponent implements
Property.ValueChangeListener  {


    @AutoGenerated
    private AbsoluteLayout mainLayout;
    @AutoGenerated
    private Button addEventButton;
    @AutoGenerated
    private Label headerLable;

    private static final long serialVersionUID = -
2508818236617842605L;

    //layouts
    private HorizontalLayout addEventLayout;
```

```
    private HorizontalLayout buttons;
    private HorizontalLayout deleteLayout;

    //components
    Window subwindow;
    public Window editEventSubwindow;
    public Window deleteEventSubwindow;
    private Window addEventSubwindow;
    private Table eventSongSelector;
    private Table eventSongSelected;
    private Table eventTable;
    private Button saveEvent;
    private Button cancelEvent;
    protected Button tableEditButton;
    protected Button tableDeleteButton;
    private TextField eventTitle;
    private PopupDateField eventDate;
    private TwinColSelect eventPeopleSelector;

    private SongService songDB;
    private HappeningService eventDB;
    private PersonService personDB;
    private BeanItemContainer<Song> songBeanSelected;
    private BeanItemContainer<Happening> eventBean;
    private BeanItemContainer<Person> personBean;
    private HorizontalLayout editEventLayout;
    private TextField editEventTitle;
    private PopupDateField editEventDate;
    private TwinColSelect editPeopleSelector;
    private Table editSongsDrag;
    private Table editSongsDrop;
    private Button saveEdit;
    private Button cancelEdit;



    /**
     * The constructor should first build the main layout, set the
     * composition root and then do any custom initialization.
     *
     * The constructor will not be automatically regenerated by the
     * visual editor.
     */
    public HappeningView() {
        buildMainLayout();
        setCompositionRoot(mainLayout);

        // TODO add user code here

    }


    @AutoGenerated
    private AbsoluteLayout buildMainLayout() {
        // common part: create layout
        mainLayout = new AbsoluteLayout();

        // top-level component properties
        setWidth("100.0%");
```

```
        setHeight("100.0%");

        // headerLable
        headerLable = new Label();
        headerLable.setWidth("-1px");
        headerLable.setHeight("-1px");
        headerLable.setStyleName("h2");
        headerLable.setValue("Calvary Chapel Turku Events");
        headerLable.setContentMode(3);
        mainLayout.addComponent(headerLable,
"top:5.0px;left:20.0px;");

        // addEventButton
        addEventButton = new Button("add event");
        addEventButton.setStyleName("link");
        addEventButton.setImmediate(true);
        addEventButton.addListener(new AddEventListener());
        mainLayout.addComponent(addEventButton,
"top:10.0px;right:36.0px;");

        //mainLayout.addComponent(buttonLayout,
"top:10.0px;right:36.0px;");

        // treeTable
        eventTable = buildEventTable();
        mainLayout.addComponent(eventTable,
"top:40.0px;left:20.0px;");

        return mainLayout;
    }

    private Table buildEventTable(){
        eventDB = new HappeningService();
        ArrayList<Happening> eventList = new ArrayList<Happening>();
        eventList=eventDB.getAllEvents();

        // Create a container for beans
        eventBean = new BeanItemContainer<Happening>(Happening.class);
        for(Happening happening:eventList){
            eventBean.addBean(new Happening(happening.getEventId(),
happening.getTitle(), happening.getDate(), happening.getPersonList(),
happening.getSongList()));
        }

        // Table
        eventTable = new Table("", eventBean);
        eventTable.setWidth("97.0%");
        eventTable.setPageLength(10);
        eventTable.setStyleName(Reindeer.TABLE_BORDERLESS);
        eventTable.setSelectable(true);
        eventTable.setSortAscending(true);
        eventTable.setSortContainerPropertyId("date");

        //edit button column
        eventTable.addGeneratedColumn("edit", new
Table.ColumnGenerator() {
            private static final long serialVersionUID =
6709348760708076615L;
```

```
            public Component generateCell(Table source, Object itemId,
Object columnId) {
                //Item item = eventTable.getItem(itemId);
                tableEditButton = new Button("edit");
                tableEditButton.setData(itemId);
                tableEditButton.setStyleName("link");
                tableEditButton.setImmediate(true);
                tableEditButton.addListener(new EditButtonListener());

                return tableEditButton;
            }
        });

        //delete button column
        eventTable.addGeneratedColumn("delete", new
Table.ColumnGenerator() {
            private static final long serialVersionUID =
6709348760708076615L;
            public Component generateCell(Table source, Object itemId,
Object columnId) {
                //Item item = eventTable.getItem(itemId);
                tableDeleteButton = new Button("delete");
                tableDeleteButton.setData(itemId);
                tableDeleteButton.setStyleName("link");
                tableDeleteButton.setImmediate(true);
                tableDeleteButton.addListener(new
DeleteButtonListener());

                return tableDeleteButton;
            }
        });

        //column headers
        eventTable.setVisibleColumns(new Object[]{"title", "date",
"edit", "delete"});
        eventTable.setColumnHeader("edit", "");
        eventTable.setColumnHeader("delete", "");
        eventTable.setColumnWidth("edit", 50);
        eventTable.setColumnWidth("delete", 50);
        eventTable.setColumnAlignment("edit", Table.ALIGN_CENTER);
        eventTable.setColumnAlignment("delete", Table.ALIGN_CENTER);
        //eventTable.setColumnExpandRatio("title", 1);

        return eventTable;
    }


    /***
     * add event window
     * @return
     */
    private HorizontalLayout buildAddEventLayout() {
        // common part: create layout
        addEventLayout = new HorizontalLayout();
        addEventLayout.setSpacing(true);
        addEventLayout.setMargin(true);

        // top-level component properties
        setWidth("100%");
```

```
        setHeight("100%");

        // Left side
        VerticalLayout left = new VerticalLayout();
        left.setSpacing(true);

        // eventTitle
        eventTitle = new TextField("Event Title:");
        eventTitle.setWidth("282px");
        eventTitle.setHeight("-1px");
        left.addComponent(eventTitle);

        // eventDate
        eventDate = new PopupDateField("Date and Time:");
        eventDate.setWidth("-1px");
        eventDate.setHeight("-1px");
        eventDate.setValue(new java.util.Date());
        eventDate.setResolution(PopupDateField.RESOLUTION_MIN);
        eventDate.addListener(this);
        eventDate.setImmediate(true);
        left.addComponent(eventDate);

        // eventPeopleSelector
        personDB = new PersonService();
        ArrayList<Person> listPeople = new ArrayList<Person>();
        listPeople=personDB.getAllPeople();

         // Create a container for beans
        personBean = new BeanItemContainer<Person>(Person.class);
        for(Person person:listPeople){
            personBean.addBean(new Person(person.getPersonId(),
person.getfName(), person.getlName(), person.getEmail(),
person.getPhoneNumber()));
        }

        eventPeopleSelector = new TwinColSelect("Add people to the
event:", personBean);

eventPeopleSelector.setItemCaptionMode(TwinColSelect.ITEM_CAPTION_MODE
_ID);
        eventPeopleSelector.setItemCaptionPropertyId("fullName");
        eventPeopleSelector.setWidth("-1px");
        eventPeopleSelector.setRows(8);
        eventPeopleSelector.setNullSelectionAllowed(true);
        eventPeopleSelector.setMultiSelect(true);
        eventPeopleSelector.setImmediate(true);
        eventPeopleSelector.addListener(this);
        left.addComponent(eventPeopleSelector);

        // Right side
        VerticalLayout right = new VerticalLayout();
        right.setSpacing(true);

        // eventSongSelector
        eventSongSelector = eventSongSelector();
        right.addComponent(eventSongSelector);

        // eventSongSelected
        eventSongSelected = new Table("Drop songs here:");
```

```
        initializeSelected(new SourceIs(eventSongSelector));
        right.addComponent(eventSongSelected);

        // buttons
        buttons = new HorizontalLayout();
        buttons.setWidth("-1px");
        buttons.setHeight("-1px");
        buttons.setMargin(false);
        buttons.setSpacing(true);

        // saveEvent
        saveEvent = new Button("Save");
        saveEvent.setImmediate(true);
        saveEvent.addListener(new SaveButtonListener());
        buttons.addComponent(saveEvent);

        // cancelEvent
        cancelEvent = new Button("Cancel");
        cancelEvent.setImmediate(true);
        cancelEvent.addListener(new Button.ClickListener() {
            private static final long serialVersionUID = -
3845382681013653261L;
            public void buttonClick(ClickEvent event) {
                // close the window by removing it from the parent
window

(addEventSubwindow.getParent()).removeWindow(addEventSubwindow);
            }
        });
        buttons.addComponent(cancelEvent);

        right.addComponent(buttons);
        right.setComponentAlignment(buttons, Alignment.BOTTOM_RIGHT);

        addEventLayout.addComponent(left);
        addEventLayout.addComponent(right);

        return addEventLayout;
    }


    /***
     * song table source...drop from
     * @return
     */
    private Table eventSongSelector(){
        songDB = new SongService();
        ArrayList<Song> list = new ArrayList<Song>();
        list=songDB.getAllSongs();

        // Create a container for beans
        BeanItemContainer<Song> songBean = new
BeanItemContainer<Song>(Song.class);

        // Add some beans to it
        for(Song song:list){
            songBean.addBean(new Song(song.getSongId(),
song.getTitle(), song.getAuthor(), song.getKey()));
        }
```

```
        // Bind a table to it
        eventSongSelector = new Table("Drag and drop a song to the
second table to add it to the event:", songBean);
        eventSongSelector.setWidth("700px");
        eventSongSelector.setPageLength(8);
        eventSongSelector.setColumnExpandRatio(songBean.firstItemId(),
1);
        eventSongSelector.setImmediate(true);
        eventSongSelector.setSelectable(true);
        eventSongSelector.setNullSelectionAllowed(true);
        eventSongSelector.setMultiSelect(true);
        eventSongSelector.setSortAscending(true);
        eventSongSelector.setSortContainerPropertyId("title");
        eventSongSelector.setVisibleColumns(new Object[]{"title",
"author", "key"});
        eventSongSelector.setColumnWidth("key", 35);
        eventSongSelector.setColumnAlignment("key",
Table.ALIGN_CENTER);
        eventSongSelector.setColumnExpandRatio("title", 1);

        // enable drag and drop options
        eventSongSelector.setDragMode(TableDragMode.ROW);
        eventSongSelector.setDragMode(TableDragMode.MULTIROW);

        return eventSongSelector;
    }

    /***
     * song table destination...drop to
     * @return
     */
    @SuppressWarnings({ "static-access" })
    private void initializeSelected(final ClientSideCriterion
acceptCriterion){

        // Create a container for beans
        songBeanSelected = new BeanItemContainer<Song>(Song.class);
        songBeanSelected.addItem(new Song(0, "title", "author",
"key"));

        // table properties
        eventSongSelected.setContainerDataSource(songBeanSelected);
        eventSongSelected.setWidth("700px");
        eventSongSelected.setPageLength(8);
        eventSongSelected.setStyleName(Reindeer.TABLE_BORDERLESS);
        //eventSongSelected.setImmediate(true);
        //eventSongSelected.setSelectable(true);
        //eventSongSelected.setNullSelectionAllowed(true);
        //eventSongSelected.setMultiSelect(true);
        eventSongSelected.setVisibleColumns(new Object[]{"title",
"author", "key"});
        eventSongSelected.setColumnWidth("key", 35);
        eventSongSelected.setColumnAlignment("key",
Table.ALIGN_CENTER);

eventSongSelected.setRowHeaderMode(eventSongSelected.ROW_HEADER_MODE_I
NDEX);
        eventSongSelected.setColumnExpandRatio("title", 1);
```

```
        eventSongSelected.removeAllItems();

        // enable drag and drop features
        eventSongSelected.setDragMode(TableDragMode.ROW);
        eventSongSelected.setDragMode(TableDragMode.MULTIROW);

        // drop handler
        eventSongSelected.setDropHandler(new DropHandler() {
            private static final long serialVersionUID = -
7005945929912511061L;
            public AcceptCriterion getAcceptCriterion() {
                //return new And(acceptCriterion, AcceptItem.ALL);
                //return new Not(VerticalLocationIs.MIDDLE);
                return new And(new SourceIs(eventSongSelected),
VerticalLocationIs.MIDDLE);
                //return AcceptAll.get();
            }

            public void drop(DragAndDropEvent event) {
                // Wrapper for the object that is dragged
                DataBoundTransferable t =
(DataBoundTransferable)event.getTransferable();

                // Make sure the drag source is one of the two tables
                if (t.getSourceComponent() != eventSongSelector &&
                    t.getSourceComponent() != eventSongSelected )
                    return;

                AbstractSelectTargetDetails target =
(AbstractSelectTargetDetails) event.getTargetDetails();

                // Get ids of the dragged item and the target item
                Object sourceItemId = t.getData("itemId");
                Object targetItemId = target.getItemIdOver();

                // Do not allow drop on the item itself
                if (sourceItemId.equals(targetItemId))
                    return;

                Song bean = null;
                if (sourceItemId instanceof BeanItem<?>) {
                    bean = (Song)
                        ((BeanItem<?>) sourceItemId).getBean();
                }
                else if (sourceItemId instanceof Song) {
                    bean = (Song) sourceItemId;
                }

                // On which side of the target the item was dropped
                VerticalDropLocation location =
target.getDropLocation();

                // The table was empty or otherwise not on an item
                if (targetItemId == null) {
                    songBeanSelected.addItem(bean); // Add to the end
                }

                // Drop at the top of a subtree -> make it previous
                else if (location == VerticalDropLocation.TOP) {
```

```
songBeanSelected.addItemAt(songBeanSelected.indexOfId(targetItemId),
bean);
                }

                // Drop below another item -> make it next
                else if (location == VerticalDropLocation.BOTTOM) {
                    songBeanSelected.addItemAfter(targetItemId, bean);
                }

            }

        });
    }


    /***
     * edit event window
     */
    @SuppressWarnings("serial")
    private HorizontalLayout EditEventWindow(Happening editable) {

        int eventId = editable.getEventId();
        //getWindow().showNotification("" + songId);
        HappeningService db = new HappeningService();
        editable = db.getEventById(eventId);
        //getWindow().showNotification("" + editable.getSongId());

        // common part: create layout
        editEventLayout = new HorizontalLayout();
        editEventLayout.setSpacing(true);
        editEventLayout.setMargin(true);

        // top-level component properties
        setWidth("100%");
        setHeight("100%");

        // Left side
        VerticalLayout left = new VerticalLayout();
        left.setSpacing(true);

        // eventTitle
        editEventTitle = new TextField("Event Title:",
editable.getTitle());
        editEventTitle.setWidth("282px");
        editEventTitle.setHeight("-1px");
        left.addComponent(editEventTitle);

        // eventDate
        editEventDate = new PopupDateField("Date and Time:");
        //editEventDate.getDateFormat()
        editEventDate.setValue(editable.getDate());
        editEventDate.setWidth("-1px");
        editEventDate.setHeight("-1px");
        editEventDate.setResolution(3);
        left.addComponent(editEventDate);

        // eventPeopleSelector
        personDB = new PersonService();
```

```
        ArrayList<Person> listPeople = new ArrayList<Person>();
        listPeople=personDB.getAllPeople();

         // Create a container for beans
        personBean = new BeanItemContainer<Person>(Person.class);
        for(Person person:listPeople){
            personBean.addBean(new Person(person.getPersonId(),
person.getfName(), person.getlName(), person.getEmail(),
person.getPhoneNumber()));
        }

        editPeopleSelector = new TwinColSelect("Add people to the
event:", personBean);

editPeopleSelector.setItemCaptionMode(TwinColSelect.ITEM_CAPTION_MODE_
ID);
        editPeopleSelector.setItemCaptionPropertyId("fullName");
        editPeopleSelector.setWidth("-1px");
        editPeopleSelector.setRows(8);
        editPeopleSelector.setNullSelectionAllowed(true);
        editPeopleSelector.addListener(new
Property.ValueChangeListener() {
            public void valueChange(ValueChangeEvent event) {

System.out.println(event.getProperty().getType().getName());
                if (event.getProperty().getValue() != null)

System.out.println(event.getProperty().getValue().getClass().getName()
);
            }
        });
        editPeopleSelector.setImmediate(true);
        left.addComponent(editPeopleSelector);

        // Right side
        VerticalLayout right = new VerticalLayout();
        right.setSpacing(true);

        // eventSongSelector
        editSongsDrag = editSongsDrag();
        right.addComponent(editSongsDrag);

        // eventSongSelected
        editSongsDrop = editSongsDrop();    //(new
SourceIs(eventSongSelected));
        right.addComponent(editSongsDrop);

        // buttons
        buttons = new HorizontalLayout();
        buttons.setWidth("-1px");
        buttons.setHeight("-1px");
        buttons.setMargin(false);
        buttons.setSpacing(true);

        // saveEvent
        saveEdit = new Button("Save");
        saveEdit.setImmediate(true);
        saveEdit.addListener(new SaveButtonListener());
        buttons.addComponent(saveEdit);
```

```
        // cancelEvent
        cancelEdit = new Button("Cancel");
        cancelEdit.setImmediate(true);
        cancelEdit.addListener(new Button.ClickListener() {
            private static final long serialVersionUID = -
3845382681013653261L;
            public void buttonClick(ClickEvent event) {
                // close the window by removing it from the parent
window

(editEventSubwindow.getParent()).removeWindow(editEventSubwindow);
            }
        });
        buttons.addComponent(cancelEdit);

        right.addComponent(buttons);
        right.setComponentAlignment(buttons, Alignment.BOTTOM_RIGHT);

        editEventLayout.addComponent(left);
        editEventLayout.addComponent(right);

        return editEventLayout;
    }


    /***
     * song table source...drop from
     * @return
     */
    private Table editSongsDrag(){
        songDB = new SongService();
        ArrayList<Song> list = new ArrayList<Song>();
        list=songDB.getAllSongs();

        // Create a container for beans
        BeanItemContainer<Song> songBean = new
BeanItemContainer<Song>(Song.class);

        // Add some beans to it
        for(Song song:list){
            songBean.addBean(new Song(song.getSongId(),
song.getTitle(), song.getAuthor(), song.getKey()));
        }

        // Bind a table to it
        editSongsDrag = new Table("Drag and drop a song to the second
table to add it to the event:", songBean);
        editSongsDrag.setWidth("700px");
        editSongsDrag.setPageLength(8);
        editSongsDrag.setColumnExpandRatio(songBean.firstItemId(), 1);
        editSongsDrag.setImmediate(true);
        editSongsDrag.setSelectable(true);
        editSongsDrag.setNullSelectionAllowed(true);
        editSongsDrag.setMultiSelect(true);
        editSongsDrag.setSortAscending(true);
        editSongsDrag.setSortContainerPropertyId("title");
        editSongsDrag.setVisibleColumns(new Object[]{"title",
"author", "key"});
```

```
        editSongsDrag.setColumnWidth("key", 35);
        editSongsDrag.setColumnAlignment("key", Table.ALIGN_CENTER);
        editSongsDrag.setColumnExpandRatio("title", 1);

        // enable drag and drop options
        editSongsDrag.setDragMode(TableDragMode.ROW);
        editSongsDrag.setDragMode(TableDragMode.MULTIROW);

        return editSongsDrag;
    }

    /***
     * song table destination...drop to
     * @return
     */
    @SuppressWarnings({ "static-access" })
    private Table editSongsDrop(){

        // table with selected songs
        editSongsDrop = new Table("Drop songs here:");

        // Create a container for beans
        songBeanSelected = new BeanItemContainer<Song>(Song.class);
        songBeanSelected.addBean(new Song(0, "title", "author",
"key"));

        // list of songs in event
        String selected_songs = "";
        Object itemId = songBeanSelected.firstItemId();
        for (int i = 0; i<songBeanSelected.size(); i++ ){
            selected_songs += ((Song)itemId).getSongId();

            if (itemId != songBeanSelected.lastItemId()){
                itemId = songBeanSelected.nextItemId(itemId);
                selected_songs += ",";
            }
        }

        // table properties
        editSongsDrop.setContainerDataSource(songBeanSelected);
        editSongsDrop.setWidth("700px");
        editSongsDrop.setPageLength(8);
        editSongsDrop.setStyleName(Reindeer.TABLE_BORDERLESS);
        //eventSongSelected.setImmediate(true);
        //eventSongSelected.setSelectable(true);
        //eventSongSelected.setNullSelectionAllowed(true);
        //eventSongSelected.setMultiSelect(true);
        editSongsDrop.setVisibleColumns(new Object[]{"title",
"author", "key"});
        editSongsDrop.setColumnWidth("key", 35);
        editSongsDrop.setColumnAlignment("key", Table.ALIGN_CENTER);

editSongsDrop.setRowHeaderMode(editSongsDrop.ROW_HEADER_MODE_INDEX);
        editSongsDrop.setColumnExpandRatio("title", 1);

        // enable drag and drop features
        editSongsDrop.setDragMode(TableDragMode.ROW);
        editSongsDrop.setDragMode(TableDragMode.MULTIROW);
```

```
        // drop handler
        editSongsDrop.setDropHandler(new DropHandler() {
            private static final long serialVersionUID = -
7005945929912511061L;
            public AcceptCriterion getAcceptCriterion() {
                //return new And(acceptCriterion, AcceptItem.ALL);
                return new Not(VerticalLocationIs.MIDDLE);
                //return AcceptAll.get();
            }

            public void drop(DragAndDropEvent event) {
                // Wrapper for the object that is dragged
                DataBoundTransferable t =
(DataBoundTransferable)event.getTransferable();

                // Make sure the drag source is one of the two tables
                if (t.getSourceComponent() != editSongsDrag &&
                    t.getSourceComponent() != editSongsDrop )
                    return;

                AbstractSelectTargetDetails target =
(AbstractSelectTargetDetails) event.getTargetDetails();

                // Get ids of the dragged item and the target item
                Object sourceItemId = t.getData("itemId");
                Object targetItemId = target.getItemIdOver();

                // Do not allow drop on the item itself
                if (sourceItemId.equals(targetItemId))
                    return;

                Song bean = null;
                if (sourceItemId instanceof BeanItem<?>) {
                    bean = (Song)
                        ((BeanItem<?>) sourceItemId).getBean();
                }
                else if (sourceItemId instanceof Song) {
                    bean = (Song) sourceItemId;
                }

                // On which side of the target the item was dropped
                VerticalDropLocation location =
target.getDropLocation();

                // The table was empty or otherwise not on an item
                if (targetItemId == null) {
                    songBeanSelected.addItem(bean); // Add to the end
                }

                // Drop at the top of a subtree -> make it previous
                else if (location == VerticalDropLocation.TOP) {

songBeanSelected.addItemAt(songBeanSelected.indexOfId(targetItemId),
bean);
                }

                // Drop below another item -> make it next
                else if (location == VerticalDropLocation.BOTTOM) {
                    songBeanSelected.addItemAfter(targetItemId, bean);
```

```
                }

            }

        });
        return editSongsDrop;
    }



    /***
     * delete event window
     */
    private HorizontalLayout DeleteEventWindow(final Object
deletable){

        //layout holder
        deleteLayout = new HorizontalLayout();
        deleteLayout.setSpacing(true);
        setWidth("100.0%");

        // delete button
        Button confirmDelete = new Button("Delete");
        confirmDelete.addListener(new Button.ClickListener() {
            private static final long serialVersionUID =
8855123333095887 27L;
            public void buttonClick(ClickEvent event) {
                try {
                    eventDB = new HappeningService();
                    int iidee = ((Happening)deletable).getEventId();

                    eventDB.deleteEvent(iidee);
                    eventBean.removeItem(deletable);
                    eventTable.sort();
                    eventTable.requestRepaint();

deleteEventSubwindow.getParent().removeWindow(deleteEventSubwindow);
                } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        });

        // cancel button
        Button cancelDelete = new Button("Cancel");
        cancelDelete.addListener(new Button.ClickListener() {
            private static final long serialVersionUID = -
5876105296264453353L;
            public void buttonClick(ClickEvent event) {
                // close the window by removing it from the parent
window

(deleteEventSubwindow.getParent()).removeWindow(deleteEventSubwindow);
            }
        });

        deleteLayout.addComponent(confirmDelete);
        deleteLayout.addComponent(cancelDelete);
```

```
        return deleteLayout;
    }


    /***
     * listeners
     */
    public class SaveButtonListener implements Button.ClickListener {
        private static final long serialVersionUID =
2749307457283317485L;
        @SuppressWarnings("rawtypes")
        public void buttonClick(ClickEvent event) {
            // Check that the password and confirm password equal
            try {
                HappeningService db = new HappeningService();

                // list of people in event
                String selected_people = "";
                Object[] person_objects =
((Set)eventPeopleSelector.getValue()).toArray();
                for (int i=0; i<person_objects.length; i++) {
                    selected_people +=
((Person)person_objects[i]).getPersonId() + ",";
                }
                selected_people = selected_people.substring(0,
(selected_people.length()-1));

                // list of songs in event
                String selected_songs = "";
                Object itemId = songBeanSelected.firstItemId();
                for (int i = 0; i<songBeanSelected.size(); i++ ){
                    selected_songs += ((Song)itemId).getSongId();

                    if (itemId != songBeanSelected.lastItemId()){
                        itemId = songBeanSelected.nextItemId(itemId);
                        selected_songs += ",";
                    }
                }
                //getWindow().showNotification("Selected people: " +
selected_people);

                int newEventId = db.getEventCount() + 1;
                db.createEvent(eventTitle.getValue().toString(),
eventDate.getValue().toString(), selected_people, selected_songs);
                eventBean.addBean(new Happening(newEventId,
eventTitle.getValue().toString(), eventDate.getValue().toString(),
selected_people, selected_songs));
                eventTable.sort();
                eventTable.requestRepaint();

(addEventSubwindow.getParent()).removeWindow(addEventSubwindow);
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
```

```
    //clear the add song window
    public void clearModalWindow(){
        eventTitle.setValue("");
        eventDate.setValue(null);
        eventPeopleSelector.setValue(null);
        //eventSongSelected.removeAllItems();
    }


    // launch add event window
    public class AddEventListener implements Button.ClickListener {
        private static final long serialVersionUID =
5488766819787033905L;
        public void buttonClick(ClickEvent event) {
            addEventSubwindow = new Window("Add a new Event");
            addEventSubwindow.center();

            // Configure the windows layout; by default a
VerticalLayout
            VerticalLayout layout = (VerticalLayout)
addEventSubwindow.getContent();
            layout.setMargin(true);
            layout.setSpacing(true);
            layout.setSizeUndefined();

            // Add some content
            addEventSubwindow.addComponent(buildAddEventLayout());
            if (addEventSubwindow.getParent() != null) {
                // window is already showing
                getWindow().showNotification(
                        "Window is already open");
            } else {
                clearModalWindow();                          // clear
contents of fields from previous entries
                getWindow().addWindow(addEventSubwindow);    // Open
the sub-window by adding it to the parent
            }
        }
    }


    // launch the delete window
    public class DeleteButtonListener implements Button.ClickListener
{
        private static final long serialVersionUID = 1L;
        public void buttonClick(ClickEvent event) {
            Happening itemId = (Happening)event.getButton().getData();

            // Create the window
            deleteEventSubwindow = new Window("Do you want to delete
the Event: " + itemId.getTitle() + "?");
            deleteEventSubwindow.setModal(true);

            // Configure the windows layout; by default a
VerticalLayout
            VerticalLayout deleteHolder = (VerticalLayout)
deleteEventSubwindow.getContent();
            deleteHolder.setMargin(true);
            deleteHolder.setSpacing(true);
```

```
            deleteHolder.setSizeUndefined();

            //songTable.getValue();
            // Add content to subwindow

deleteEventSubwindow.addComponent(DeleteEventWindow(itemId));
            //open the window
            if (deleteEventSubwindow.getParent() != null) {
                getWindow().showNotification("Window is already
open");
            } else {
                //getWindow().showNotification("Selected: " +
clicked);
                getWindow().addWindow(deleteEventSubwindow);
// Open the subwindow by adding it to the parent
            }
        }
    }

    // launch the edit window
    public class EditButtonListener implements Button.ClickListener {
        private static final long serialVersionUID = 1L;
        public void buttonClick(ClickEvent event) {
            editEventSubwindow = new Window("Edit the song");
            editEventSubwindow.center();

            // Configure the windows layout; by default a
VerticalLayout
            VerticalLayout holder = (VerticalLayout)
editEventSubwindow.getContent();
            holder.setMargin(true);
            holder.setSpacing(true);
            holder.setSizeUndefined();

            // Add content to subwindow
            Happening clicked =
(Happening)event.getButton().getData();
            editEventSubwindow.addComponent(EditEventWindow(clicked));
            //open the window
            if (editEventSubwindow.getParent() != null) {
                getWindow().showNotification("Window is already
open");          // window is already showing
            } else {
                getWindow().addWindow(editEventSubwindow);
// Open the subwindow by adding it to the parent
            }
        }
    }

    public void valueChange(ValueChangeEvent event) {
        // Get the new value and format it to the current locale
        DateFormat dateFormatter = DateFormat.getDateTimeInstance();
        Object value = event.getProperty().getValue();
        if (value == null || !(value instanceof Date)) {
            getWindow().showNotification("Invalid date entered");
        } else {
            String dateOut = dateFormatter.format(value);
            // Show notification
            getWindow().showNotification("Starting date: " + dateOut);
```

```
        }
    }
}
```

```
//PersonView.java

package com.aaronpratt.teamplanner.views;

import java.util.ArrayList;

import com.aaronpratt.teamplanner.data.Person;
import com.aaronpratt.teamplanner.service.PersonService;
import com.vaadin.annotations.AutoGenerated;
import com.vaadin.data.util.BeanItemContainer;
import com.vaadin.ui.AbsoluteLayout;
import com.vaadin.ui.Alignment;
import com.vaadin.ui.Button;
import com.vaadin.ui.Component;
import com.vaadin.ui.CustomComponent;
import com.vaadin.ui.HorizontalLayout;
import com.vaadin.ui.Label;
import com.vaadin.ui.Table;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.TextField;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.Window;
import com.vaadin.ui.themes.Reindeer;

public class PeopleView extends CustomComponent {

    @AutoGenerated
    private AbsoluteLayout mainLayout;
    @AutoGenerated
    private HorizontalLayout buttonLayout_1;
    @AutoGenerated
    private Button addPeopleButton;
    @AutoGenerated
    private Button emailButton;
    @AutoGenerated
    private Label headerLable;
    @AutoGenerated
    private Table peopleTable;
    private static final long serialVersionUID = 4961689989814043201L;

    // windows
    Window subwindow;
    public Window editPersonSubwindow;
    public Window deletePersonSubwindow;

    // layouts
    private VerticalLayout verticalLayout;
    private HorizontalLayout nameLayout;
```

```
    private HorizontalLayout buttonLayout;
    private HorizontalLayout horizontal1;
    private HorizontalLayout horizontal2;
    private VerticalLayout personEditLayout;
    private HorizontalLayout deleteLayout;

    // UI components
    private TextField firstName;
    private TextField lastName;
    private TextField phone;
    private TextField email;
    private TextField fNameEdit;
    private TextField lNameEdit;
    private TextField emailEdit;
    private TextField phoneEdit;
    private TextField hiddenId;
    private Button saveButton;
    private Button cancelButton;
    private Button saveEdit;
    private Button cancelEdit;
    private Button tableEditButton;
    private Button tableDeleteButton;

    // objects
    PersonService db;
    ArrayList<Person> list;
    BeanItemContainer<Person> personBean;


    /**
     * The constructor should first build the main layout, set the
     * composition root and then do any custom initialization.
     *
     * The constructor will not be automatically regenerated by the
     * visual editor.
     */
    public PeopleView() {
        buildMainLayout();
        setCompositionRoot(mainLayout);
    }


    @AutoGenerated
    private AbsoluteLayout buildMainLayout() {
        // common part: create layout
        mainLayout = new AbsoluteLayout();

        // top-level component properties
        setWidth("100.0%");
        setHeight("100.0%");

        // headerLable
        headerLable = new Label();
        headerLable.setWidth("-1px");
        headerLable.setHeight("-1px");
        headerLable.setValue("Calvary Chapel Turku People");
        headerLable.setStyleName("h2");
        headerLable.setContentMode(3);
```

```
        mainLayout.addComponent(headerLable,
"top:5.0px;left:20.0px;");

        // buttonLayout_1
        buttonLayout_1 = buildButtonLayout_1();
        mainLayout.addComponent(buttonLayout_1,
"top:10.0px;right:30.0px;");

        // people table
        peopleTable = buildPeopleTable();
        mainLayout.addComponent(peopleTable,
"top:40.0px;left:20.0px;");

        // edit layout
        //personEditLayout = EditPersonLayout();
        //mainLayout.addComponent(personEditLayout,
"top:255.0px;left:20.0px;");

        return mainLayout;
    }

    @AutoGenerated
    private HorizontalLayout buildButtonLayout_1() {
        // common part: create layout
        buttonLayout_1 = new HorizontalLayout();
        buttonLayout_1.setWidth("110px");
        buttonLayout_1.setHeight("34px");
        buttonLayout_1.setImmediate(true);
        buttonLayout_1.setMargin(false);
        buttonLayout_1.setSpacing(true);

        // emailButton
        emailButton = new Button("email");
        emailButton.setWidth("-1px");
        emailButton.setHeight("-1px");
        emailButton.setStyleName("link");
        emailButton.setImmediate(true);
        buttonLayout_1.addComponent(emailButton);
        buttonLayout_1.setComponentAlignment(emailButton, new
Alignment(34));

        // addPeopleButton
        addPeopleButton = new Button("add people");
        addPeopleButton.setWidth("-1px");
        addPeopleButton.setHeight("-1px");
        addPeopleButton.setStyleName("link");
        addPeopleButton.setImmediate(true);
        addPeopleButton.addListener(new AddPeopleListener());
        buttonLayout_1.addComponent(addPeopleButton);
        buttonLayout_1.setComponentAlignment(addPeopleButton, new
Alignment(34));

        return buttonLayout_1;
    }

    private Table buildPeopleTable(){
        db = new PersonService();
        list = new ArrayList<Person>();
        list=db.getAllPeople();
```

```
        // Create a container for beans
        personBean = new BeanItemContainer<Person>(Person.class);

        // Add some beans to it
        for(Person person:list){
            personBean.addBean(new Person(person.getPersonId(),
person.getfName(), person.getlName(), person.getEmail(),
person.getPhoneNumber()));
        }

        // peopleTable
        peopleTable = new Table("", personBean);
        peopleTable.setWidth("97.0%");
        peopleTable.setPageLength(10);
        peopleTable.setStyleName(Reindeer.TABLE_BORDERLESS);
        peopleTable.setSelectable(true);
        peopleTable.setMultiSelect(true);
        peopleTable.setImmediate(true);                 // react at
once when something is selected
        peopleTable.setColumnReorderingAllowed(true);
        peopleTable.setSortAscending(true);
        peopleTable.setSortContainerPropertyId("fName");

        //edit button column
        peopleTable.addGeneratedColumn("edit", new
Table.ColumnGenerator() {
            private static final long serialVersionUID =
7190293891473776387L;
            public Component generateCell(Table source, Object itemId,
Object columnId) {
                //Item item = songTable.getItem(itemId);
                tableEditButton = new Button("edit");
                tableEditButton.setData(itemId);
                tableEditButton.setStyleName("link");
                tableEditButton.setImmediate(true);
                tableEditButton.addListener(new EditButtonListener());

                return tableEditButton;
            }
        });

        //delete button column
        peopleTable.addGeneratedColumn("delete", new
Table.ColumnGenerator() {
            private static final long serialVersionUID =
6709348760708076615L;
            public Component generateCell(Table source, Object itemId,
Object columnId) {
                tableDeleteButton = new Button("delete");
                tableDeleteButton.setData(itemId);
                tableDeleteButton.setStyleName("link");
                tableDeleteButton.setImmediate(true);
                tableDeleteButton.addListener(new
DeleteButtonListener());

                return tableDeleteButton;
            }
        });
```

```
        //column headers
        peopleTable.setVisibleColumns(new Object[]{"fName", "lName",
"email", "phoneNumber", "edit", "delete"});
        peopleTable.setColumnHeader("fName", "First Name");
        peopleTable.setColumnHeader("lName", "Last Name");
        peopleTable.setColumnHeader("phoneNumber", "Phone Number");
        peopleTable.setColumnHeader("edit", "");
        peopleTable.setColumnHeader("delete", "");
        peopleTable.setColumnWidth("phoneNumber", 100);
        peopleTable.setColumnWidth("edit", 50);
        peopleTable.setColumnWidth("delete", 50);
        peopleTable.setColumnAlignment("edit", Table.ALIGN_CENTER);
        peopleTable.setColumnAlignment("delete", Table.ALIGN_CENTER);

        return peopleTable;
    }


    /***
     * Person edit window
     * @return
     */
    @AutoGenerated
    private VerticalLayout EditPersonWindow(Person editable) {

        int personId = editable.getPersonId();
        //getWindow().showNotification("" + songId);
        PersonService db = new PersonService();
        editable = db.getPersonById(personId);
        //getWindow().showNotification("" + editable.getSongId());

        // create layout
        personEditLayout = new VerticalLayout();
        personEditLayout.setImmediate(true);
        personEditLayout.setSpacing(true);
        // top-level component properties
        setWidth("100.0%");
        setHeight("100.0%");

        // first and last name
        horizontal1 = new HorizontalLayout();
        horizontal1.setSpacing(true);
        fNameEdit = new TextField("First Name", editable.getfName());
        lNameEdit = new TextField("Last Name", editable.getlName());
        horizontal1.addComponent(fNameEdit);
        horizontal1.addComponent(lNameEdit);
        personEditLayout.addComponent(horizontal1);

        // email
        emailEdit = new TextField("email", editable.getEmail());
        emailEdit.setWidth("100.0%");
        personEditLayout.addComponent(emailEdit);

        // phone
        phoneEdit = new TextField("Phone Number",
editable.getPhoneNumber());
        personEditLayout.addComponent(phoneEdit);
```

```
        // buttons
        horizontal2 = new HorizontalLayout();
        horizontal2.setSpacing(true);
        saveEdit = new Button("Save");
        saveEdit.addListener(new SaveEditListener());
        horizontal2.addComponent(saveEdit);

        cancelEdit = new Button("Cancel");
        cancelEdit.addListener(new Button.ClickListener() {
            private static final long serialVersionUID =
8595209353865391442L;
            public void buttonClick(ClickEvent event) {
                // close the window by removing it from the parent
window
                personEditLayout.setVisible(false);
            }
        });
        horizontal2.addComponent(cancelEdit);

        //hidden field for person id
        hiddenId = new TextField("", "" + editable.getPersonId());
        hiddenId.setVisible(false);
        hiddenId.setEnabled(false);
        horizontal2.addComponent(hiddenId);

        personEditLayout.addComponent(horizontal2);

        return personEditLayout;
    }



    /**
     * Modal Subwindow for adding a person
     *
     */
        private VerticalLayout buildModalWindow() {
            // common part: create layout
            verticalLayout = new VerticalLayout();
            verticalLayout.setWidth("387px");
            verticalLayout.setHeight("250px");
            verticalLayout.setImmediate(false);
            verticalLayout.setMargin(true);
            verticalLayout.setSpacing(true);

            // nameLayout
            nameLayout = buildNameLayout();
            verticalLayout.addComponent(nameLayout);

            // email
            email = new TextField();
            email.setWidth("90.0%");
            email.setHeight("-1px");
            email.setCaption("E-Mail Address");
            verticalLayout.addComponent(email);

            // phone
            phone = new TextField();
            phone.setWidth("-1px");
```

```
        phone.setHeight("-1px");
        phone.setCaption("Phone Number");
        verticalLayout.addComponent(phone);

        // buttonLayout
        buttonLayout = buildButtonLayout();
        verticalLayout.addComponent(buttonLayout);

        return verticalLayout;
    }

    private HorizontalLayout buildNameLayout() {
        // common part: create layout
        nameLayout = new HorizontalLayout();
        nameLayout.setWidth("100.0%");
        nameLayout.setHeight("-1px");
        nameLayout.setImmediate(true);
        nameLayout.setMargin(false);

        // firstName
        firstName = new TextField();
        firstName.setWidth("-1px");
        firstName.setHeight("-1px");
        firstName.setCaption("First Name");
        nameLayout.addComponent(firstName);

        // lastName
        lastName = new TextField();
        lastName.setWidth("-1px");
        lastName.setHeight("-1px");
        lastName.setCaption("Last Name");
        nameLayout.addComponent(lastName);

        return nameLayout;
    }

    private HorizontalLayout buildButtonLayout() {
        // common part: create layout
        buttonLayout = new HorizontalLayout();
        buttonLayout.setWidth("-1px");
        buttonLayout.setHeight("-1px");
        buttonLayout.setImmediate(true);
        buttonLayout.setMargin(false);
        buttonLayout.setSpacing(true);

        // saveButton
        saveButton = new Button();
        saveButton.setWidth("66px");
        saveButton.setHeight("26px");
        saveButton.setCaption("Save");
        saveButton.setImmediate(true);
        saveButton.addListener(new SaveButtonListener());
        buttonLayout.addComponent(saveButton);

        // cancelButton
        cancelButton = new Button();
        cancelButton.setWidth("-1px");
        cancelButton.setHeight("26px");
        cancelButton.setCaption("Cancel");
```

```
            cancelButton.setImmediate(true);
            cancelButton.addListener(new Button.ClickListener() {
                private static final long serialVersionUID =
8595209353865391442L;
                public void buttonClick(ClickEvent event) {
                    // close the window by removing it from the parent
window
                    subwindow.getParent().removeWindow(subwindow);
                }
            });
            buttonLayout.addComponent(cancelButton);

            return buttonLayout;
        }


        /***
         * Delete Person window
         */
        private HorizontalLayout DeletePersonWindow(final Object
deletable){

            //layout holder
            deleteLayout = new HorizontalLayout();
            deleteLayout.setSpacing(true);
            setWidth("100.0%");

            // delete button
            Button confirmDelete = new Button("Delete");
            confirmDelete.addListener(new Button.ClickListener() {
                private static final long serialVersionUID =
885512333309588727L;
                public void buttonClick(ClickEvent event) {
                    try {
                        db = new PersonService();
                        int iidee = ((Person)deletable).getPersonId();

                        db.deletePerson(iidee);
                        personBean.removeItem(deletable);
                        peopleTable.sort();
                        peopleTable.requestRepaint();

deletePersonSubwindow.getParent().removeWindow(deletePersonSubwindow);
                    } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
            });

            // cancel button
            Button cancelDelete = new Button("Cancel");
            cancelDelete.addListener(new Button.ClickListener() {
                private static final long serialVersionUID = -
5876105296264453353L;
                public void buttonClick(ClickEvent event) {
                    // close the window by removing it from the parent
window
```

```
(deletePersonSubwindow.getParent()).removeWindow(deletePersonSubwindow
);
                }
            });

            deleteLayout.addComponent(confirmDelete);
            deleteLayout.addComponent(cancelDelete);

            return deleteLayout;
        }



        /***
         *  Listeners
         */
        public class SaveButtonListener implements
Button.ClickListener {
            private static final long serialVersionUID =
2749307457283317485L;
            public void buttonClick(ClickEvent event) {
                // Check that the password and confirm password equal
                try {
                    db = new PersonService();
                    int newPersonId = db.getPersonCount() + 1;
                    db.createPerson(firstName.getValue().toString(),
lastName.getValue().toString(), email.getValue().toString(),
phone.getValue().toString());
                    personBean.addBean(new Person(newPersonId,
firstName.getValue().toString(), lastName.getValue().toString(),
email.getValue().toString(), phone.getValue().toString()));
                    peopleTable.sort();
                    peopleTable.requestRepaint();
                    subwindow.getParent().removeWindow(subwindow);
                } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }

        // save edit listener
        public class SaveEditListener implements Button.ClickListener
{
            private static final long serialVersionUID =
2749307457283317485L;
            public void buttonClick(ClickEvent event) {
                try {
                    db = new PersonService();
                    int personId =
Integer.parseInt(hiddenId.getValue().toString());

db.editPerson(hiddenId.getValue().toString(),fNameEdit.getValue().toSt
ring(), lNameEdit.getValue().toString(),
emailEdit.getValue().toString(), phoneEdit.getValue().toString());
                    Object itemId = personBean.firstItemId();
                    Person s = (Person)itemId;
                    for (int i = 0; i<personBean.size(); i++ ){
```

```
                        s = (Person)itemId;
                        if (s.getPersonId() == personId){
                            personBean.removeItem(itemId);
                            personBean.addBean(new Person(personId,
fNameEdit.getValue().toString(), lNameEdit.getValue().toString(),
emailEdit.getValue().toString(), phoneEdit.getValue().toString()));
                            peopleTable.sort();
                            peopleTable.requestRepaint();
                            break;
                        }
                        if (itemId != personBean.lastItemId())
                            itemId = personBean.nextItemId(itemId);
                    } //while (itemId != personBean.lastItemId());

editPersonSubwindow.getParent().removeWindow(editPersonSubwindow);
                } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }

        //clear the add song window
        public void clearModalWindow(){
            firstName.setValue("");
            lastName.setValue("");
            email.setValue("");
            phone.setValue("");
        }


        // launch add person window
        public class AddPeopleListener implements Button.ClickListener
{
            private static final long serialVersionUID =
3524941338036158477L;
            public void buttonClick(ClickEvent event) {
                subwindow = new Window("Add a new Person");
                subwindow.setModal(true);

                // Configure the windows layout; by default a
VerticalLayout
                VerticalLayout layout = (VerticalLayout)
subwindow.getContent();
                layout.setMargin(true);
                layout.setSpacing(true);
                layout.setSizeUndefined();

                // Add some content; a label and a close-button
                subwindow.addComponent(buildModalWindow());
                if (subwindow.getParent() != null) {
                    // window is already showing
                    getWindow().showNotification("Window is already
open");
                } else {
                    clearModalWindow();
                    getWindow().addWindow(subwindow);
                }
            }
```

```
        }

        // launch the delete window
        public class DeleteButtonListener implements
Button.ClickListener {
            private static final long serialVersionUID = 1L;
            public void buttonClick(ClickEvent event) {
                Person itemId = (Person)event.getButton().getData();

                // Create the window
                deletePersonSubwindow = new Window("Do you want to
delete the person: " + itemId.getfName() + " " + itemId.getlName() +
"?");
                deletePersonSubwindow.setModal(true);
                deletePersonSubwindow.center();

                // Configure the windows layout; by default a
VerticalLayout
                VerticalLayout deleteHolder = (VerticalLayout)
deletePersonSubwindow.getContent();
                deleteHolder.setMargin(true);
                deleteHolder.setSpacing(true);
                deleteHolder.setSizeUndefined();

                // Add content to subwindow

deletePersonSubwindow.addComponent(DeletePersonWindow(itemId));
                //open the window
                if (deletePersonSubwindow.getParent() != null) {
                    getWindow().showNotification("Window is already
open");          // window is already showing
                } else {
                    getWindow().addWindow(deletePersonSubwindow);
// Open the subwindow by adding it to the parent
                }
            }
        }

        // launch the edit window
        public class EditButtonListener implements
Button.ClickListener {
            private static final long serialVersionUID = 1L;
            public void buttonClick(ClickEvent event) {
                editPersonSubwindow = new Window("Edit the person");
                editPersonSubwindow.center();

                // Configure the windows layout; by default a
VerticalLayout
                VerticalLayout holder = (VerticalLayout)
editPersonSubwindow.getContent();
                holder.setMargin(true);
                holder.setSpacing(true);
                holder.setSizeUndefined();

                // Add content to subwindow
                Person clicked = (Person)event.getButton().getData();

editPersonSubwindow.addComponent(EditPersonWindow(clicked));
                //open the window
```

```
                if (editPersonSubwindow.getParent() != null) {
                    getWindow().showNotification("Window is already
open");     // window is already showing
                } else {
                    getWindow().addWindow(editPersonSubwindow);
// Open the subwindow by adding it to the parent
                }
            }
        }
}
```

```
//SongView.java

package com.aaronpratt.teamplanner.views;

import java.util.ArrayList;

import com.aaronpratt.teamplanner.data.Song;
import com.aaronpratt.teamplanner.service.SongService;

import com.vaadin.annotations.AutoGenerated;
import com.vaadin.data.Property;
import com.vaadin.data.Property.ValueChangeEvent;
import com.vaadin.data.util.BeanItemContainer;
import com.vaadin.ui.AbsoluteLayout;
import com.vaadin.ui.Alignment;
import com.vaadin.ui.Button;
import com.vaadin.ui.Button.ClickEvent;
import com.vaadin.ui.Component;
import com.vaadin.ui.CustomComponent;
import com.vaadin.ui.HorizontalLayout;
import com.vaadin.ui.Label;
import com.vaadin.ui.NativeSelect;
import com.vaadin.ui.RichTextArea;
import com.vaadin.ui.Table;
import com.vaadin.ui.TextField;
import com.vaadin.ui.VerticalLayout;
import com.vaadin.ui.Window;
import com.vaadin.ui.themes.Reindeer;

public class SongView extends CustomComponent implements
Property.ValueChangeListener {


    @AutoGenerated
    private AbsoluteLayout mainLayout;
    @AutoGenerated
    private Button saveEdit;
    @AutoGenerated
    private RichTextArea songEditArea;
    @AutoGenerated
    private Button cancelEdit;
```

```
    @AutoGenerated
    private NativeSelect changeKey;
    @AutoGenerated
    private TextField authorEdit;
    @AutoGenerated
    private TextField titleEdit;
    @AutoGenerated
    private Table songTable;
    @AutoGenerated
    private HorizontalLayout buttonLayout;
    @AutoGenerated
    private Button addSongButton;
    @AutoGenerated
    private Label headerLabel;
    private static final long serialVersionUID = 4821432957877802753L;

    //windows
    private Window addSongSubwindow;
    private Window editSongSubwindow;
    private Window deleteSongSubwindow;

    //layouts
    private VerticalLayout modalLayout;
    private VerticalLayout editLayout;
    private HorizontalLayout horizontalLayout;
    private HorizontalLayout deleteLayout;
    private HorizontalLayout hL1;
    private HorizontalLayout hL2;

    //components
    private RichTextArea songTextArea;
    private TextField title;
    private TextField hiddenId;
    private TextField author;
    private NativeSelect keyBox;
    private Button saveButton;
    private Button cancelButton;
    private Button tableEditButton;
    private Button tableDeleteButton;

    //objects
    SongService db;
    ArrayList<Song> list;
    BeanItemContainer<Song> songBean;

    /**
     * The constructor should first build the main layout, set the
     * composition root and then do any custom initialization.
     *
     * The constructor will not be automatically regenerated by the
     * visual editor.
     */

    public SongView() {
        buildMainLayout();
        setCompositionRoot(mainLayout);
    }
```

```
    @AutoGenerated
    private AbsoluteLayout buildMainLayout() {
        setWidth("100.0%");
        setHeight("100.0%");

        // common part: create layout
        mainLayout = new AbsoluteLayout();

        // headerLabel
        headerLabel = new Label();
        headerLabel.setWidth("-1px");
        headerLabel.setHeight("-1px");
        headerLabel.setStyleName("h2");
        headerLabel.setValue("Calvary Chapel Turku Songs");
        headerLabel.setContentMode(3);
        headerLabel.setImmediate(true);
        mainLayout.addComponent(headerLabel,
"top:5.0px;left:20.0px;");

        // buttonLayout
        buttonLayout = buildButtonLayout();
        mainLayout.addComponent(buttonLayout,
"top:10.0px;right:30.0px;");

        songTable = buildSongTable();
        mainLayout.addComponent(songTable, "top:40.0px;left:20.0px;");

        return mainLayout;
    }

    @AutoGenerated
    private HorizontalLayout buildButtonLayout() {
        // common part: create layout
        buttonLayout = new HorizontalLayout();
        buttonLayout.setWidth("-1px");
        buttonLayout.setHeight("-1px");
        buttonLayout.setImmediate(true);
        buttonLayout.setMargin(false);
        buttonLayout.setSpacing(true);

        // addSongButton
        addSongButton = new Button("add song");
        addSongButton.setWidth("-1px");
        addSongButton.setHeight("-1px");
        addSongButton.setStyleName("link");
        addSongButton.setImmediate(true);
        addSongButton.addListener(new AddSongListener());
        buttonLayout.addComponent(addSongButton);
        buttonLayout.setComponentAlignment(addSongButton, new
Alignment(34));

        return buttonLayout;
    }

    public Table buildSongTable(){
        db = new SongService();
        list = new ArrayList<Song>();
        list=db.getAllSongs();
```

```
        // Create a container for beans
        songBean = new BeanItemContainer<Song>(Song.class);
        for(Song song:list){
            songBean.addBean(new Song(song.getSongId(),
song.getTitle(), song.getAuthor(), song.getKey()));
        }
        // Use the title property as the item ID of the bean
        //songBean.setBeanIdProperty("title");

        // Bind a table to it
        songTable = new Table("", songBean);
        songTable.setWidth("97.0%");
        songTable.setPageLength(15);
        songTable.setStyleName(Reindeer.TABLE_BORDERLESS);
        songTable.setImmediate(true);
        songTable.setSelectable(true);
        songTable.setSortAscending(true);
        songTable.setSortContainerPropertyId("title");

        //edit button column
        songTable.addGeneratedColumn("edit", new
Table.ColumnGenerator() {
            private static final long serialVersionUID =
6709348760708076615L;
            public Component generateCell(Table source, Object itemId,
Object columnId) {
                //Item item = songTable.getItem(itemId);
                tableEditButton = new Button("edit");
                tableEditButton.setData(itemId);
                tableEditButton.setStyleName("link");
                tableEditButton.setImmediate(true);
                tableEditButton.addListener(new EditButtonListener());

                return tableEditButton;
            }
        });

        //delete button column
        songTable.addGeneratedColumn("delete", new
Table.ColumnGenerator() {
            private static final long serialVersionUID =
6709348760708076615L;
            public Component generateCell(Table source, Object itemId,
Object columnId) {
                //Item item = songTable.getItem(itemId);
                tableDeleteButton = new Button("delete");
                tableDeleteButton.setData(itemId);
                tableDeleteButton.setStyleName("link");
                tableDeleteButton.setImmediate(true);
                tableDeleteButton.addListener(new
DeleteButtonListener());

                return tableDeleteButton;
            }
        });

        //column headers
        songTable.setVisibleColumns(new Object[]{"title", "author",
"key", "edit", "delete"});
```

```
        songTable.setColumnHeader("edit", "");
        songTable.setColumnHeader("delete", "");
        songTable.setColumnWidth("key", 35);
        songTable.setColumnWidth("edit", 50);
        songTable.setColumnWidth("delete", 50);
        songTable.setColumnAlignment("key", Table.ALIGN_CENTER);
        songTable.setColumnAlignment("edit", Table.ALIGN_CENTER);
        songTable.setColumnAlignment("delete", Table.ALIGN_CENTER);
        //songTable.setColumnExpandRatio("title", 1);

        return songTable;
    }



    /***
     * song edit window layout
     */
    @AutoGenerated
    private VerticalLayout EditSongWindow(Song editable) {

        int songId = editable.getSongId();
        //getWindow().showNotification("" + songId);
        SongService db = new SongService();
        editable = db.getSongById(songId);
        //getWindow().showNotification("" + editable.getSongId());

        // common part: create layout
        editLayout = new VerticalLayout();
        editLayout.setImmediate(true);
        editLayout.setSpacing(true);
        // top-level component properties
        setWidth("100.0%");
        setHeight("100.0%");

        // horizontalLayout
        hL1 = new HorizontalLayout();
        hL1.setWidth("100.0%");
        hL1.setHeight("-1px");
        hL1.setImmediate(true);
        hL1.setMargin(false);
        hL1.setSpacing(true);

        // titleField
        titleEdit = new TextField("Title:", editable.getTitle());
        titleEdit.setWidth("225px");
        titleEdit.setHeight("-1px");
        hL1.addComponent(titleEdit);

        // author
        authorEdit = new TextField("Author:", editable.getAuthor());
        authorEdit.setWidth("225px");
        authorEdit.setHeight("-1px");
        hL1.addComponent(authorEdit);

        // change key box
        String[] available_keys = new String[] { "C", "Db", "D", "Eb",
"E", "F", "F#", "G", "Ab", "A", "Bb", "B"};
        changeKey = new NativeSelect();
```

```
        for (int i = 0; i < available_keys.length; i++) {
            changeKey.addItem(available_keys[i]);
        }

        changeKey.setNullSelectionAllowed(false);
        changeKey.setValue(editable.getKey());
        changeKey.setImmediate(true);
        changeKey.addListener(this);
        changeKey.setWidth("60px");
        changeKey.setHeight("-1px");
        changeKey.setCaption("Key:");
        hL1.addComponent(changeKey);

        //add horizontal layout
        editLayout.addComponent(hL1);

        // SongTextArea
        songEditArea = new RichTextArea("", editable.getSongText());
        songEditArea.setWidth("600px");
        songEditArea.setHeight("400px");
        songEditArea.setImmediate(true);
        editLayout.addComponent(songEditArea);

        // buttonLayout
        hL2 = new HorizontalLayout();
        hL2.setWidth("-1px");
        hL2.setHeight("-1px");
        hL2.setImmediate(true);
        hL2.setMargin(false);
        hL2.setSpacing(true);

        // saveButton
        saveEdit = new Button();
        saveEdit.setWidth("-1px");
        saveEdit.setHeight("-1px");
        saveEdit.setCaption("Save");
        saveEdit.setImmediate(true);
        saveEdit.addListener(new SaveEditListener());
        hL2.addComponent(saveEdit);

        // cancelButton
        cancelEdit = new Button();
        cancelEdit.setWidth("-1px");
        cancelEdit.setHeight("-1px");
        cancelEdit.setCaption("Cancel");
        cancelEdit.setImmediate(true);
        cancelEdit.addListener(new Button.ClickListener() {
            private static final long serialVersionUID =
7215324540929658891L;
            public void buttonClick(ClickEvent event) {
                // close the window by removing it from the parent
window

(editSongSubwindow.getParent()).removeWindow(editSongSubwindow);
            }
        });

        hL2.addComponent(cancelEdit);
```

```
        //hidden field for song id
        hiddenId = new TextField("", "" + editable.getSongId());
        hiddenId.setVisible(false);
        hiddenId.setEnabled(false);
        hL2.addComponent(hiddenId);

        editLayout.addComponent(hL2);

        return editLayout;
    }

    /***
     * Delete song window
     */
    private HorizontalLayout DeleteSongWindow(final Object deletable){

        //layout holder
        deleteLayout = new HorizontalLayout();
        deleteLayout.setSpacing(true);
        setWidth("100.0%");

        // delete button
        Button confirmDelete = new Button("Delete");
        confirmDelete.addListener(new Button.ClickListener() {
            private static final long serialVersionUID =
885512333309588727L;
            public void buttonClick(ClickEvent event) {
                try {
                    db = new SongService();
                    int iidee = ((Song)deletable).getSongId();

                    db.deleteSong(iidee);
                    songBean.removeItem(deletable);
                    songTable.sort();
                    songTable.requestRepaint();

deleteSongSubwindow.getParent().removeWindow(deleteSongSubwindow);
                } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        });

        // cancel button
        Button cancelDelete = new Button("Cancel");
        cancelDelete.addListener(new Button.ClickListener() {
            private static final long serialVersionUID = -
5876105296264453353L;
            public void buttonClick(ClickEvent event) {
                // close the window by removing it from the parent
window

(deleteSongSubwindow.getParent()).removeWindow(deleteSongSubwindow);
            }
        });

        deleteLayout.addComponent(confirmDelete);
```

```
        deleteLayout.addComponent(cancelDelete);

        return deleteLayout;
    }



    /***
     * build modal window for adding a song
     */
    @AutoGenerated
    private VerticalLayout buildModalWindow() {
        // common part: create layout
        modalLayout = new VerticalLayout();
        modalLayout.setImmediate(true);
        modalLayout.setMargin(true);
        modalLayout.setSpacing(true);
        // top-level component properties
        setWidth("100.0%");
        setHeight("100.0%");

        // horizontalLayout
        horizontalLayout = buildHorizontalLayout();
        modalLayout.addComponent(horizontalLayout);

        // SongTextArea
        songTextArea = new RichTextArea();
        songTextArea.setWidth("600px");
        songTextArea.setHeight("400px");
        songTextArea.setImmediate(true);
        modalLayout.addComponent(songTextArea);

        // buttonLayout
        buttonLayout = buildButtonHolder();
        modalLayout.addComponent(buttonLayout);

        return modalLayout;
    }

    @AutoGenerated
    private HorizontalLayout buildHorizontalLayout() {
        // common part: create layout
        horizontalLayout = new HorizontalLayout();
        horizontalLayout.setWidth("100.0%");
        horizontalLayout.setHeight("-1px");
        horizontalLayout.setImmediate(true);
        horizontalLayout.setMargin(false);
        horizontalLayout.setSpacing(true);

        // titleField
        title = new TextField("Title:");
        title.setWidth("225px");
        title.setHeight("-1px");
        horizontalLayout.addComponent(title);

        // author
        author = new TextField("Author:");
        author.setWidth("225px");
        author.setHeight("-1px");
```

```
        //author.setNewItemsAllowed(true);
        //author.setNewItemHandler(this);
        //author.setFilteringMode(Filtering.FILTERINGMODE_STARTSWITH);
        //author.addListener(this);
        horizontalLayout.addComponent(author);

        // key
        String[] available_keys = new String[] { "C", "Db", "D", "Eb",
"E", "F", "F#", "G", "Ab", "A", "Bb", "B"};
        keyBox = new NativeSelect();

        for (int i = 0; i < available_keys.length; i++) {
            keyBox.addItem(available_keys[i]);
        }
        //keyBox.setNullSelectionAllowed(false);
        keyBox.setImmediate(true);
        keyBox.addListener(this);
        keyBox.setWidth("60px");
        keyBox.setHeight("-1px");
        keyBox.setCaption("Key:");
        horizontalLayout.addComponent(keyBox);

        return horizontalLayout;
    }

    @SuppressWarnings("serial")
    @AutoGenerated
    private HorizontalLayout buildButtonHolder() {
        // common part: create layout
        buttonLayout = new HorizontalLayout();
        buttonLayout.setWidth("-1px");
        buttonLayout.setHeight("-1px");
        buttonLayout.setImmediate(true);
        buttonLayout.setMargin(false);
        buttonLayout.setSpacing(true);

        // saveButton
        saveButton = new Button("Save");
        saveButton.setWidth("-1px");
        saveButton.setHeight("-1px");
        saveButton.setImmediate(true);
        saveButton.addListener(new SaveButtonListener());
        buttonLayout.addComponent(saveButton);

        // cancelButton
        cancelButton = new Button("Cancel");
        cancelButton.setWidth("-1px");
        cancelButton.setHeight("-1px");
        cancelButton.setImmediate(true);
        cancelButton.addListener(new Button.ClickListener() {
            public void buttonClick(ClickEvent event) {
                // close the window by removing it from the parent
window

(addSongSubwindow.getParent()).removeWindow(addSongSubwindow);
            }
        });
        buttonLayout.addComponent(cancelButton);
```

```
        return buttonLayout;
    }



    /***
     *  Save Button Listeners
     */
    public class SaveButtonListener implements Button.ClickListener {
        private static final long serialVersionUID =
2749307457283317485L;
        public void buttonClick(ClickEvent event) {
            try {
                db = new SongService();
                int newSongId = db.getSongCount() + 1;
                db.createSong(title.getValue().toString(),
author.getValue().toString(), keyBox.getValue().toString(),
songTextArea.getValue().toString());
                songBean.addBean(new Song(newSongId,
title.getValue().toString(), author.getValue().toString(),
keyBox.getValue().toString()));
                songTable.sort();
                songTable.requestRepaint();

addSongSubwindow.getParent().removeWindow(addSongSubwindow);
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

    public class SaveEditListener implements Button.ClickListener {
        private static final long serialVersionUID =
2749307457283317485L;
        public void buttonClick(ClickEvent event) {
            try {
                db = new SongService();
                int songId =
Integer.parseInt(hiddenId.getValue().toString());

db.editSong(hiddenId.getValue().toString(),titleEdit.getValue().toStri
ng(), authorEdit.getValue().toString(),
changeKey.getValue().toString(), songEditArea.getValue().toString());
                Object itemId = songBean.firstItemId();
                Song s = (Song)itemId;
                for (int i = 0; i<songBean.size(); i++ ){
                    s = (Song)itemId;
                    if (s.getSongId() == songId){
                        songBean.removeItem(itemId);
                        songBean.addBean(new
Song(songId,titleEdit.getValue().toString(),
authorEdit.getValue().toString(), changeKey.getValue().toString(),
songEditArea.getValue().toString()));
                        songTable.sort();
                        songTable.requestRepaint();
                        break;
                    }
                    if (itemId != songBean.lastItemId())
```

```
                        itemId = songBean.nextItemId(itemId);
                } //while (itemId != songBean.lastItemId());


editSongSubwindow.getParent().removeWindow(editSongSubwindow);
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

    // change of selection in a combo-box
    public void valueChange(ValueChangeEvent event) {
    /*  if (event.getProperty() == keyBox){
            getWindow().showNotification("Selected key: " +
event.getProperty());
        }else if (event.getProperty() == author){
            getWindow().showNotification("Selected author: " +
event.getProperty());
        }
    */
    }

    // clear the add song window
    public void clearModalWindow(){
        title.setValue("");
        author.setValue("");
        keyBox.setValue(null);
        songTextArea.setValue("");
    }

    // launch the add song Window
    public class AddSongListener implements Button.ClickListener {
        private static final long serialVersionUID =
3645087944675292900L;
        public void buttonClick(ClickEvent event) {
            addSongSubwindow = new Window("Add a new song");
            addSongSubwindow.setModal(true);

            // Configure the windows layout; by default a
VerticalLayout
            VerticalLayout layout = (VerticalLayout)
addSongSubwindow.getContent();
            layout.setMargin(true);
            layout.setSpacing(true);
            layout.setSizeUndefined();

            // Add content to sub-window
            addSongSubwindow.addComponent(buildModalWindow());
            // button for opening the sub-window
            if (addSongSubwindow.getParent() != null) {
                // window is already showing
                getWindow().showNotification("Window is already
open");
            } else {
                clearModalWindow();                         // clear
contents of fields from previous entries
```

```
                getWindow().addWindow(addSongSubwindow);    // Open
the subwindow by adding it to the parent
            }
        }
    }

    // launch the delete window
    public class DeleteButtonListener implements Button.ClickListener
{
        private static final long serialVersionUID = 1L;
        public void buttonClick(ClickEvent event) {
            Song itemId = (Song)event.getButton().getData();

            // Create the window
            deleteSongSubwindow = new Window("Do you want to delete
the song: " + itemId.getTitle() + "?");
            deleteSongSubwindow.setModal(true);

            // Configure the windows layout; by default a
VerticalLayout
            VerticalLayout deleteHolder = (VerticalLayout)
deleteSongSubwindow.getContent();
            deleteHolder.setMargin(true);
            deleteHolder.setSpacing(true);
            deleteHolder.setSizeUndefined();

            //songTable.getValue();
            // Add content to subwindow

deleteSongSubwindow.addComponent(DeleteSongWindow(itemId));
            //open the window
            if (deleteSongSubwindow.getParent() != null) {
                getWindow().showNotification("Window is already
open");
            } else {
                //getWindow().showNotification("Selected: " +
clicked);
                getWindow().addWindow(deleteSongSubwindow);
// Open the subwindow by adding it to the parent
            }
        }
    }

    // launch the edit window
    public class EditButtonListener implements Button.ClickListener {
        private static final long serialVersionUID = 1L;
        public void buttonClick(ClickEvent event) {
            editSongSubwindow = new Window("Edit the song");
            editSongSubwindow.center();

            // Configure the windows layout; by default a
VerticalLayout
            VerticalLayout holder = (VerticalLayout)
editSongSubwindow.getContent();
            holder.setMargin(true);
            holder.setSpacing(true);
            holder.setSizeUndefined();

            // Add content to subwindow
```

```
            Song clicked = (Song)event.getButton().getData();
            editSongSubwindow.addComponent(EditSongWindow(clicked));
            //open the window
            if (editSongSubwindow.getParent() != null) {
                getWindow().showNotification("Window is already
open");          // window is already showing
            } else {
                getWindow().addWindow(editSongSubwindow);
// Open the subwindow by adding it to the parent
            }
        }
    }
}
```

2 Database Configurations

Creating the database:

```
CREATE DATABASE `team_planner` ;
```

Creating the ’events’ table:

```
CREATE TABLE `team_planner`.`events` (
    `E_id` INT( 11 ) NOT NULL AUTO_INCREMENT PRIMARY KEY ,
    `title` VARCHAR( 200 ) NOT NULL ,
    `date` DATETIME NULL DEFAULT NULL ,
    `person_id_list` VARCHAR( 100 ) NULL ,
    `song_id_list` VARCHAR( 100 ) NULL
) ENGINE = InnoDB;
```

Creating the ’people’ table:

```
CREATE TABLE `team_planner`.`people` (
    `P_id` INT( 11 ) NOT NULL AUTO_INCREMENT PRIMARY KEY ,
    `fName` VARCHAR( 30 ) NOT NULL ,
    `lName` VARCHAR( 30 ) NOT NULL ,
    `email` VARCHAR( 50 ) NULL DEFAULT NULL ,
    `phone` VARCHAR( 20 ) NULL DEFAULT NULL
) ENGINE = InnoDB;
```

Creating the ’songs’ table:

```
CREATE TABLE `team_planner`.`songs` (
    `S_id` INT( 11 ) NOT NULL AUTO_INCREMENT PRIMARY KEY ,
    `title` VARCHAR( 150 ) NOT NULL ,
    `author` VARCHAR( 100 ) NULL DEFAULT NULL ,
    `songKey` VARCHAR( 5 ) NULL DEFAULT NULL ,
    `songText` LONGTEXT NULL DEFAULT NULL
) ENGINE = InnoDB;
```