



Abdirahman Ibrahim Mohammed

**DATA DRIVEN PROJECTION OF
FUTURE PURCHASING
BEHAVIOUR**

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to my father and mother for the endless encouragement, support and giving me the gift of life, without them, this journey would not have been possible. I am also grateful to my supervisor Ali Al-towati for providing me with all the necessary facilities for the research.

I also take this opportunity to express gratitude to all of my friends Jama Ismail and Lawal Olufowobi and to all of the Department faculty members for their help and support.

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author Abdirahman Ibrahim Mohammed
Title Data Driven Projection of Future Purchasing Behavior
Year 2019
Language English
Pages 40
Name of Supervisor Ali Altowati

Managing personal expenses is a problem for many people for numerous reasons, however financial status can become even more complicated. Considerable bills coupled with limited earnings can make money management intensely challenging. Even though financial pressure can be annoying, there are methods to manage finances and reclaim financial independence.

Categorization of determined data which organized hierarchy have been done before. In recent years, apps or software has been used to categorize personal expenses, in order to help the people to stick to spending limits by breaking up their cash into containers labeled with different spending categories.

The main objective of this project was to design and develop a frontend and backend web application that provides people reviewing day- to- day transactions including invoices, sales receipt, payments. The application is built with the lattes Vaadin framework and Spring Boot together to provide simple and intuitive web interface.

The amount of time needed to implement this project was inaccurately underestimated. Although some improvements have been presented, the application partially meets all the requirements and requires tremendous amount of work to be fully functional application.

Keywords Spring Boot, Vaadin, CSV Upload and bank statement

CONTENTS

ABSTRACT

1	INTRODUCTION.....	8
2	TECHNOLOGIES USED.....	9
	2.1 Spring Boot.....	9
	2.1.1 Auto-configuration	9
	2.2 Vaadin	10
	2.2.1 Vaadin Framework	10
	2.2.2 Server-side architecture	11
	2.3 Gradle	11
	2.4 Accessing data with Hibernate and MYSQL in Spring Boot.....	13
3	ANALYSIS.....	15
	3.1 Requirements.....	16
	3.1.1 Table of Requirements	16
	Requirement	16
	Description	16
	Priority 16	
	3.1.2 Bank Statement Management P1	16
	3.1.3 Import Statements.....	17
	3.1.4 Statement Format	17
	3.1.5 Retrieving Material	18
	3.1.6 Classification of Material.....	18
4	APPLICATION DESIGN AND IMPLEMENTATION	21
	4.1 Overview	21
	4.1.2 User Registration	21
	4.1.3 Login Form	22
	4.1.4 Uploading File	22
	4.1.5 Receiving Upload Data.....	23
	4.2 Uploading CSV File	24
	4.2.1 Manipulating CSV File.....	25
	4.2.2 Editing CSV file in Text Editor	25
	4.2.3 Cleaning Up the Data.....	26

4.2.4	Data File Preparation	26
4.3	Application Implementation.....	27
4.3.1	Project Structure	28
4.3.2	Class Diagram.....	29
4.4	Data Persistence	30
5	RESULTS.....	31
5.1	Admin Dashboard.....	31
5.1.1	User Interface.....	31
5.1.2	Login.....	31
5.1.3	Admin Page.....	32
5.1.4	User Registration	33
5.1.5	Upload Interface	34
6	TESTING.....	36
7	CHALLENGES AND CONCLUSION	38
	REFERENCES	40

LIST OF FIGURES AND TABLES

Figure 1. An example of Vaadin Flow framework	10
Figure 2. Gradle build file for building for Java projects	12
Figure 3. Application properties	13
Figure 4. Application Components overview	15
Figure 5. CSV file formatted with column headers	18
Figure 6. CSV file without column header	18
Figure 7.	Error! Bookmark not defined.
Figure 8. Upload Component	23
Figure 9. Use Case Diagram	27
Figure 10. Directory Structure	28
Figure 11. Class Diagram	29
Figure 12. Login Form	31
Figure 13. Admin Page	32
Figure 14. User Registration	33
Figure 15. Upload	34
Figure 16. Console output	36
Figure 17. Console output. The console displays errors, which means that the test is failed.	37
Table 1. Requirements and their descriptions	16
Table 2. Bank Statement	Error! Bookmark not defined.

LIST OF ABBREVIATIONS

UI	User Interface
CLI	Command Line Interface
JAR	Java Archive file, package file format that is used to aggerate java files into one file for distribution (Wikipedia 2017)
JDBC	Java Database Connection
JDK	Java Development Kit
DBMS	Database Management System
SQL	Structured Query Language
GWT	Google Web Toolkit
JVM	Java Virtual Machine
JPA	Java Persistence API
MIME	Multipurpose Internet Mail Extensions

1 INTRODUCTION

Categorizing bank transaction is one of the most important and critical jobs for tracking purchasing behavior. Categorizing bank transaction explains the difference between the bank balance shown in the statement and the corresponding amount shown on the person's accounting records on a particular date. It is necessary that a person's bank balance and book balance should match, but there might be a difference between these two. The reason for the difference may be a cheque issued but not presented for payment, a cheque deposited but not cleared, charges debited by bank, interest credited by a bank, or error made either by a bank or the person.

Until today, categorizing bank transaction is carried out manually by comparing the bank passbook or the bank statement issued by the bank to that of entries made in the system portable. The manual process is time-consuming and drawn to errors.

The advancement in technology especially the internet and information technology has led a new way of doing business in banking. As a result, now most of the banks have the provision to generate electronic bank statement's soft copy, instead of a traditional paper statement. E-statements look the same as paper statements and offer more value, convenience, and security with benefits, such as anytime, anywhere access.

This thesis project shows the use of e-statements for auto bank categorization in an application built with the latest technologies. This application is designed to take the work out of the whole categorization process, making it easy fast and efficient. With its feature it will only be a few short steps from the clarity that will help to measure and analyze according to gathered consumer behavior data from the bank statement.

2 TECHNOLOGIES USED

2.1 Spring Boot

Spring Boot is a Spring framework. It makes it easy to create a production-grade Spring-based Application. It is a brand new framework originally created by the Pivotal team, designed to make easy the bootstrapping and development of a modern Spring Application. The framework proceeds an opinionated approach to configuration, emancipating developers from the demand to give description boilerplate configuration. As therein, Boot intends to be a front-runner in the ever-growing rapid application development space. /1/

2.1.1 Auto-configuration

Spring Boot does all the work using Autoconfiguration and will take care of all internal dependencies that the application needs. Spring Boot auto-configures with Dispatcher Servlet, if Spring jar or Hibernate jar are in the class paths. It will auto-configure to the data source. Spring Boot provides pre-configured and selected Starter Projects to be included as a dependency in the project. /2/

During web application development, jars would be required to use by selecting the suitable version and connecting them together. For Java developers, is the enormous framework at which point an application can be easily developed following MVC architecture. Dependency injection or Inversion Of control characteristics makes them a part of the others. Spring Boot solves many problems, such as hardship to setup Hibernate Data source, Entity Manager, and Transaction Management. Hence, it easily needs to choose the proper version jars and rest of the work will be done by the spring boot. Thus, it is a fast and simple way of decreasing complexity for development /2/

2.2 Vaadin

Vaadin is an open source platform widely used for web application development. The Vaadin platform includes a set of features that provide a basic and common standard component model for the web, a Java web application framework which is designed to provide assistance for the development of web application including web services, and range of tools and application initiators. /3/

2.2.1 Vaadin Framework

As a broadly used Web UI framework, Vaadin allows Java developers to create and maintain a variety of web application rapidly. As for other Java web frameworks, Vaadin comes along features to simplify and speed up web application development. It is also compatible with primary operating systems, browsers, and web servers. Unlike other Java web frameworks, Vaadin features a server-side architecture, which means that user interfaces in Java programming can be written without using client-side-web technologies like HTML and JavaScript. /3/

```
// The following is an elementary example of Vaadin Flow Usage:
public class MyUI extends UI {
    protected void init(VaadinRequest request) {
        final TextField name = new TextField("Name");
        final Button greetButton = new Button("Greet");
        greetButton.addClickListener(
            e -> Notification.show("Hi " + name.getValue())
        );
        setContent(new VerticalLayout(name, greetButton));
    }
}
```

Figure 1. An example of Vaadin Flow framework

2.2.2 Server-side architecture

On the server-side architecture, Vaadin uses JavaScript in the browser to interact with components on the server side. It is built on Google Web Toolkit, but unlike a GWT application, Vaadin contains pre-built widgets that can be used to build UI. Rather than writing components in Java, that are compiled into Java-Script by the GWT compiler, developers simply use Vaadin classes to construct the UI in the Java code. At runtime, interaction between the server-side Java code and the JavaScript running in the browser is managed automatically. Therefore, the developers can access the data and business logic in quicker and simple way. /3/

2.3 Gradle

Gradle is an open source build automation tool that is intended to be pliant enough to build any type of software. Gradle runs only important tasks that need to run due to the modification of inputs and outputs and avoids operating unnecessary work. It can also be also used to build a cache to enable the reprocess or reuse of task outputs from earlier runs or even from various machines which share build cache. /4/

Gradle is the next developmental step in Java Virtual Machine based build tools. It requires JDK version 6 or higher to be installed in the system, because it uses JDK libraries, which are installed in the system, and then sets to the JAVA_HOME environmental variable. Gradle ties erudite lesson from established tools, for instance Ant and Maven and carries their excellence ideas to the next level. In the context of build-by convention approach, Gradle enables for declaratively forming developer's problem domain using an intense and expressive domain-specific language(DSL) applied in Groovy in lieu of XML. Because Gradle is inherently JVM, it allows to write custom logic in the language that developers are most comfortable with, which can be Java or Groovy. /4/

```

54 apply plugin: 'java'
55 apply plugin: 'eclipse'
56 apply plugin: 'application'
57
58 mainClassName = 'hello.HelloWorld'
59
60 // tag::repositories[]
61 repositories {
62     mavenCentral()
63 }
64 // end::repositories[]
65
66 // tag::jar[]
67 jar {
68     baseName = 'gs-gradle'
69     version = '0.1.0'
70 }
71 // end::jar[]
72
73 // tag::dependencies[]
74 sourceCompatibility = 1.8
75 targetCompatibility = 1.8
76
77 dependencies {
78     compile "joda-time:joda-time:2.2"
79     testCompile "junit:junit:4.12"
80 }
81 // end::dependencies[]
82
83 // tag::wrapper[]
84 // end::wrapper[]

```

Figure 2. Gradle build file for building for Java projects

In build file, the remote repositories can be specified to search for dependencies as **Figure 2** illustrates. Gradle understands different repository types and currently supports for example Maven and Ivy repositories, which access the repository via HTTP or other protocols to look for dependencies. By default, Gradle does not define repositories by itself, but instead it needs to be defined at least one. Maven is central as a dependency source for configuration, for example targets can be refer-

enced at once in the configuration to specify each target's artifacts into the file system. Dependency management is applied to automatically download above-mentioned artifacts from repository and make them available to the application. /4/

2.4 Accessing data with Hibernate and MYSQL in Spring Boot

Spring Boot has taken away the Spring framework complexity, and dramatically reduced the configuration and setup time required for spring projects. Projects can be set up with nearly zero configuration and begin building things that actually matter in the application. It includes an annotation called *@SpringBootApplication*, this is used particularly in the Application or Main class to enable host features for example Java-*@Configuration* (enable Spring Boot's auto configuration mechanism and inform Spring to automatically configure the current Application based on the dependencies that have been selected. This is important especially if the developer favors Java-based configuration over XML configurations, *@ComponentScan* (scans specifically other components that are specified in the package, and all the classes written from *@Controller* are discovered by this annotation), and *@EnableAutoConfiguration*. This annotation auto-configures the feature of Spring Boot, which can automatically do many things for the developers. For example, if a Spring MVC application is used for writing the web development with Thymeleaf JAR files on the application class path, then Spring Boot auto-configuration automatically configures the Thymeleaf template resolver, view resolver, and other more settings. /5/

Spring Boot attempts to auto-configure data if the spring-data-jpa module is implemented in the classpath by reading the database configuration from application.properties file. /6/

```
spring.jpa.hibernate.ddl-auto=create
spring.datasource.url=jdbc:mysql://localhost:3306/Account
spring.datasource.username=springuser
spring.datasource.password=ThePassword
```

Figure 3. Application properties

As shown in Figure 3 the application.properties allow grants to be connected to the database by specifying the datasource username and password properties. In the above file, the first property is for Hibernate as Spring Boot uses it for default JPA implementation. Here, spring.jpa.hibernate.ddl-auto is referred to *create*, it creates the database every time application is initialized. Because of this, there is no database structure yet. So, after first run, other options could switch it to *update* or *none*.

3 ANALYSIS

The application allows users to balance their expenses by understanding where the money goes. It provides the ability to handle both financial goals and budgets. The application will automatically create categories bank statements or transactions imported from the bank account into expense category. The appraisal involves calculating an effective analysis from the bank statement of the user. This process requires downloading the bank statement as a CSV file format from online banking service. The data will be stored in a plain text file and then the downloaded CSV file will be uploaded from the client to a remote database server. The client reads the uploaded the contents of the CSV file by executing a callback function to accomplish parse operation to populate CSV data into the Vaadin table. /7/



Figure 4. Application Components overview

3.1 Requirements

3.1.1 Table of Requirements

The table of requirements are shown below.

Table 1. Requirements and their descriptions

Requirement	Description	Prior-ity
Download	Downloading the bank statement from online banking service.	P1
Categorize	Categorizing according to the table header From the csv file provided by the bank	P2
Register	Enabling users to access the application features	P3
Login	Logging into application with registered Username and password	P4
Upload	Uploading the csv file contents into the application	P5

3.1.2 Bank Statement Management P1

It is usually complicated and hard to create a payment entry every time a payment is made or received and it is intensely time-consuming. Frequently, the date of certain transaction may not match the payment entry date. Another way to manage payments might be to create payment entries from bank statements on daily or weekly basis. The application does not include this characteristic function and is still under development. In this case, Nordea Online Accountant is used as a case study, which provided the statement management.

3.1.3 Import Statements

Ideally, a mechanism should be provided to connect to the bank account directly, which provides an access to retrieve the statements automatically, then each one of the statement transactions should be examined once they are imported and handled appropriately. At this moment it is not clear whether all the banks provide a simple method of importing the statements via an API that this application could use. As mentioned above more work needs to be accomplished in this area to find a right way of connecting to the bank and importing the statements automatically. So, this specification handles the second phase, where it is assumed that the statement is already imported in the csv format manually.

3.1.4 Statement Format

It is an improbable that all online banking services would generate the statements in the same format. Despite that, it is supposed that all of them will be able to provide necessary information in csv format. The required data components are:

1. Entry Date
2. Value Date
3. Payment Date
4. Amount
5. Beneficiary
6. Transaction
7. Reference Number
8. Originator's Reference
9. Message Card Number Receipt
10. Receipt /8/

3.1.5 Retrieving Material

Transactions from year 2017 are retrieved from Nordea online banking data, which will start storing the statements for up to 6 years from the date registered for online statements. Data in the files is not filtered or categorized in a reasonable way as can be seen from Figure 5 and 6 shown below, but rather a duplicated of collective data which need to be processed. /8/

Entry date	Value date	Payment date	Amount	Beneficiary/Remitter	Account number	BIC	Transaction	Reference number	Originator's reference	Message Card number	Receipt
02.01.2017	01.01.2017	01.01.2017	-467,75	VAASAN OPISKELIJA-ASUNTOSEÄITIÖ	205238-8648		NDEAFIHHXXX	Self service	1440 00103 14628		
02.01.2017	02.01.2017	02.01.2017	-13,44	K market Roihuvuori			Card purchase	161230015091	HELSINKI	4920170018324517	
02.01.2017	02.01.2017	02.01.2017	-3,01	S MARKET RUOHOILAHTI			Card purchase	161228836447	HELSINKI	4920170018324517	

Figure 5. CSV file formatted with column headers

26	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard
27	05.01.2017	05.01.2017	05.01.2017	-14,95	DRESSMANN 741			Card purchase	170103120001		HELSINKI
28											
29	05.01.2017	05.01.2017	05.01.2017	-24,95	CARLINGS F02			Card purchase	170103120001		HELSINKI
30											
31	05.01.2017	05.01.2017	05.01.2017	-9,99	H M HENNES MAURITZ OY			Card purchase	170103160006		HELSINKI
32											
33	05.01.2017	05.01.2017	05.01.2017	-14,49	S MARKET ITAKESKUS			Card purchase	170103800426		HELSINKI
34											
35	05.01.2017	05.01.2017	05.01.2017	-2,68	K market Ruoholahti			Card purchase	170104013744		HELSINKI
36											
37	05.01.2017	05.01.2017	05.01.2017	-45,00	NT Mai Oy Savor			Card purchase	170104019122		HELSINKI
38											
39	09.01.2017	05.01.2017	05.01.2017	-100,00				ATM withdr/Otto.			05601 URHO KEI
40											
41	09.01.2017	09.01.2017	09.01.2017	-9,44	LIDL ITAKESKUS MN228			Card purchase	170104000527		HELSINKI
42											
43	09.01.2017	09.01.2017	09.01.2017	-10,22	PAYPAL *DELLMONT			Card purchase	170104700710		EUR
44											
45	09.01.2017	09.01.2017	09.01.2017	-14,99	CLAS OHLSON IITA			Card purchase	170104800541		HELSINKI
46											
47	09.01.2017	09.01.2017	09.01.2017	-9,90	Yeti Nepal - Sairam Oy			Card purchase	170105132725		HelSinki
48											

Figure 6. CSV file without column header

3.1.6 Classification of Material

The actual task is to come up with a sensible classification for account transactions. Before the import emerges to the application via uploading feature, the system assumes that there is a classification method, which manages the contents of the csv file, and then returns the list of categorized transactions. This technique maps input data into two organized categories and potentially duplicates the transactions already entered manually as illustrated below.

Table 2. Data classification in Excel

	A	B
1	Recipient/ Payer	Categories
2	DRESSMANN 741	Clothing
3	CARLINGS F02	Fashion
4	H M HENNES MAURITZ OY	Clothing
5	S MARKET	Grocery
6	K MARKET	Grocery
7	NT MAI OY SAVOR	Restaurant
8	LIDL	Grocery
9	PYPAL	Financial
10	CLASS OHLSON	Electronics
11	YETI NEPAL - SAIRAM OY	Restaurant
12	FIINNKINO OY	Cinema
13	FAZER RESTAURANT	Cafeteria
14	EXPERT	Electronics
15	FINNPARK	Parking
16		

The illustration shows the relationship between two categories and how each element should match the proper category. For each entry from bank statement the classifier will guess the key word corresponding to the description in *Categories*. First all data will be loaded from the csv file and passed to the function to get the data from this file in a format acceptable to the database. This is basically a list of double-buffered data each of which contains *text* and *classification*. In this case, the text is the description of the transaction from the bank statement like (*DRESSMANN 741*), and the classification is the category (*Clothing*) that needs to be assigned. As soon as it receives the data, the classification just commands the classifier method, giving the text to classify.

Due to time constraints, the technical consideration of implementing this feature is relatively small. Even though it has to meet all component requirement, bank statement accuracy needs to be checked manually by matching data entry received from bank feeds. In addition, not all banks provide similar bank statements. So, each bank manages it is own summary of financial transaction, such as, the list that contains a list of items that have been processed through the bank account.

4 APPLICATION DESIGN AND IMPLEMENTATION

4.1 Overview

The application is divided into two primary components. One is the registration scheme, which shows interfaces to the user, the second is the upload scheme, which presents interfaces to the users and finally the third component is the application server, which processes the user's request before it can proceed a response through user authentication with suitable credentials.

Modern web applications are seen all throughout growing number of places for user input. These include web games, online shopping carts, and undoubtedly, website registration forms. Login, Logout, Sign Up and Register are functionalities seen on almost every website used in daily life. These functions keep track of which users are granted the access to admin pages. After logging in or signing in, users are then redirected to the index page. From an objective prospective some of the items are not considered here such as applying CSS style attribute.

4.1.2 User Registration

```

59     private User user;
60
61     Binder<User> binder = new Binder<>(User.class);
62
63     TextField firstName = new TextField("First name");
64     TextField lastName = new TextField("Last name");
65     TextField email = new TextField("Email");
66     TextField phoneNumber = new TextField("Phone number");
67     PasswordField password = new PasswordField("Password");
68     NativeSelect<String> phoneType = new NativeSelect<>("Phone Type");
69
70
71     Button save = new Button( label: "Save");
72     Button delete = new Button( label: "Delete");
73     Button cancel = new Button( label: "Cancel");

```

Code Snippet 1. User Registration function

At this step Vaadin UI components are used to capture the user details from the user's input. The user puts necessary information using *UserCreationForm* username, password, email and phone number before proceeding further to the next

step explained in the Login Form session. When the user is registered, the id will be verified and all the attributes can be received using `getUserById()` . Since this form does not have any field to recognize the usertype, the user will be saved as a user not admin. The username and password will be used later as an identifier to prove that the user is the right person who has been granted a permission to access the requested resources.

4.1.3 Login Form

```
48     Panel panel = new Panel( layout: "Login");
49     panel.setSizeUndefined();
50     addComponent(panel);
51     FormLayout content = new FormLayout();
52     TextField username = new TextField("Username");
53     content.addComponent(username);
54     PasswordField password = new PasswordField("Password");
55     content.addComponent(password);
56
```

Code Snippet 2. Login Function

After the `user_account` is created from now on the system will recognize the user by the record form provided under authentication credentials every time login activity happens. The login form includes a field for the username and password. When the user submits the login form it is essential that the code checks that the credentials are authentic, allowing the user to access the restricted resource. If the user cannot provide authentic credentials, it will not be possible to carry on passing the login form.

4.1.4 Uploading File

The Upload component enables the user to transfer data to the server. This feature has two different approaches controlled with `setImmediateMode` that influence user work processes. *Immediate* which is the default and *Non-immediate*, but this time the component is used with immediate upload mode, which displays a file name

entry box and a button for selecting or dropping the file. This method requires a receiver that performs `Upload.Receiver` in order to provide an output stream to which the transferring data is written by the server.

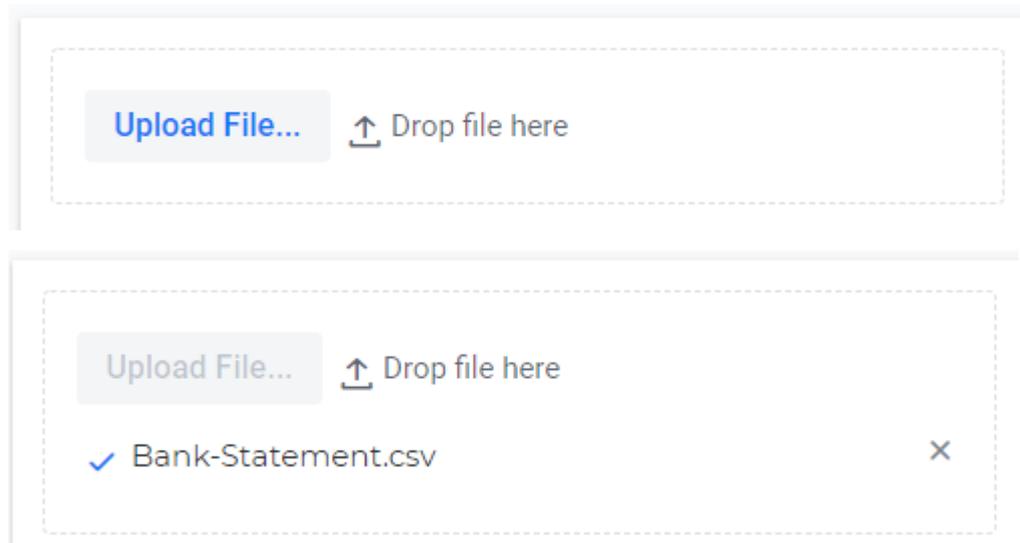


Figure 7. Upload Component

4.1.5 Receiving Upload Data

Typically, uploaded files are stored in a file system as files or other common database management systems are used to save files in different formats. Then the next step applies to the developer how the data is to be saved. The Vaadin framework provides an Upload component which writes the collected or received data to a method called *OutputStream* under *java.io*. The Upload component needs to implement the `Upload.Receiver` interface before it is used. Whenever the user clicks the upload button as is shown in **Code Snippet 3**, the `receiveUpload()` method is called. Once the method is deployed, it returns an Output stream which creates a file or memory buffer that the stream is written. The method receives the file name in a MIME type as stated by the browser. /9/

```

130     ByteArrayOutputStream baos = new ByteArrayOutputStream();
131
132     public OutputStream receiveUpload(String filename, String MIMEType) {
133
134         counter = 0;
135         fileName = filename;
136         mtype = MIMEType;
137         return baos;
138
139     }
140
141     public String getFileName() {
142         return fileName;
143     }
144
145     public String getMimeType() {
146         return mtype;
147     }
148
149     public int getLineBreakCount() {
150         return counter;
151     }

```

Code Snippet 3. File OutputStream

When getting an upload from the user, *ByteArrayOutputStream* is provided for Vaadin. Then the *OutputStream* interface receives the file name, file type in MIME, and *LineBreakCount*, which counts the number of lines in the uploaded file. /9/

4.2 Uploading CSV File

The file structure could be different depending on the file type. In this case, the Application restricts user only to upload CSV files on a specific page. If the user attempts to upload a nonsupport file type, the upload fails, and the system resends a formatting error on the document to the user. Once clicking on the choose file button, the importer will examine and determine the file and show an error if it cannot be imported. If everything goes well, the process will be completed successfully.

4.2.1 Manipulating CSV File

Before CSV file is uploaded to the system a certain formatting must be adhered to in order to avoid importing problems or receiving error messages. Sometimes the whole application may break down if there is a missing or irrelevant comma in a row. Every value after a missing or unneeded data field will be filed into the wrong column. In the worst case, the database may be debased so critically that it is necessary to return to an earlier developed version in order to backup, causing a loss of the recent data modifications.

The columns in a CSV file can appear in any order as far as the sequence is kept. Simply put, the arrangement in which column headings appear in the first row should be repeated in the following data rows, in order that the data in every field can be matched up with appropriate column. Unnecessary columns can be omitted if not wanted to add or edit data. In fact, it is a good practice to remove unnecessary columns to simplify data file structure accordingly and decrease the possibility of introducing errors in the non-required column. Fields required by the Upload tool cannot be removed, but fields required by the database providing the defaults can be removed and it is appropriate for all the records that those are being edited or added. If the default value is not suitable for any records in the data file, then it should include that column and indicate suitable values for those records.

Objective fields in the database store various versions of the values that are introduced through webpages. In this case, the file is stored in the databases as 'Account'. The values are allocated or mapped to each other and converted as data is uploaded.

4.2.2 Editing CSV file in Text Editor

CSV files are compatible with every sort of text editor, but some spreadsheets may corrupt the data, so this should be avoided. Text editors are commonly much better conducted, so those developers who are comfortable keeping the CSV file structure in text editors can use any text editor they prefer. Many people prefer to use a spreadsheet program, but the data is vulnerable on import, especially when leading

zeros can be deleted or long numbers changed to standard notation and rounded off, or on export though irrelevant characters can be added. It might not even be visible that data has changed until the file exported to CSV and then opened in a text editor. Characters that were invisible or hidden when the file was viewed via the spreadsheet application are abruptly visible when the file is viewed through a text editor. If a spreadsheet needs to be used, then it is important to prepare the spreadsheet in advance in order to handle numerical values and remove all bad characters in the output. Before uploading an edited data, it is essential to check it in order to see if the file exports cleanly.

4.2.3 Cleaning Up the Data

If it is discovered, that bad characters have been inserted into .csv data file by the spreadsheet application, still this spreadsheet application is useable by providing a clean-up solution after it by opening .csv data file in a text editor and removing any undesired characters before proceeding to the data upload. In theory, the data in each field is enclosed in double straight quotes. If double curly quotes exists in the file after export, the curly quotes are replaced with double straight quotes by a find and replace operation. Single curly quotes should also be replaced with single straight quotes.

4.2.4 Data File Preparation

- It is important to carry out some basic data checking before or during importing into CSV file. A proper structure in the CSV file is preserved while eliminating carefully the most common cause of errors.
- Correct values and columns have been assigned in upload action as required by the UPLOAD data tool.
- The upload action column should have a value specifying the action needed to perform on that row. Record where displayed without data can be named by 'none', but it is better to remove these rows entirely.
- Whenever adding record, it is necessary to check for consistency and any duplicates to be removed. The receipts database will not allow the same exact name

to be entered twice unless the columns and rows data in the field are different then that time the application will accept , but with different Id.

➤ The common and correct date format is 'YYYY/MM/DD' for date-related fields. The 'entry_date', 'value_date', and 'payment_date' are the exception to this rule. These fields may include dates for multiple values. Those dates are the same as a comma delimited string. Dates in other formats will not be denied absolutely during the validation, but the Upload Data tool will try to force the data into the 'YYYY/MM/DD' format, which later would issue in data corruption in the incorrectly formatted date fields.

4.3 Application Implementation

The requirements obtained in the requirement table can facilitate to create a use case diagram for the application.

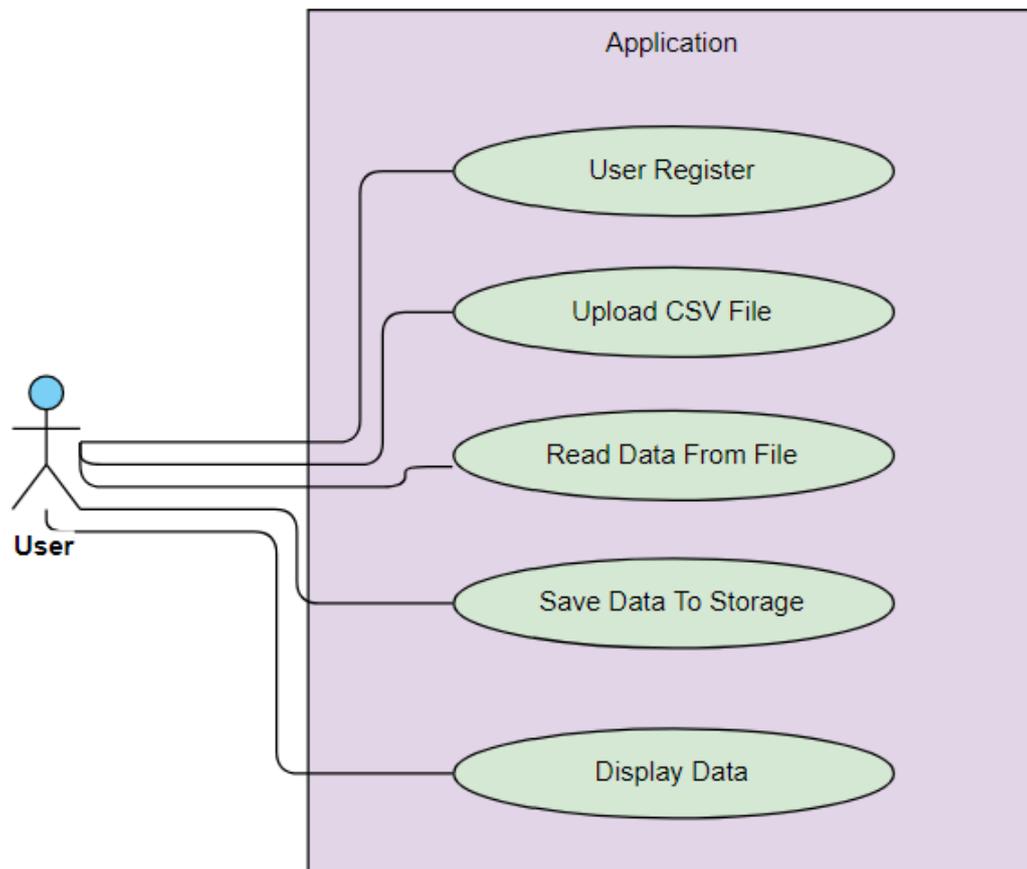


Figure 8. Use Case Diagram

4.3.1 Project Structure

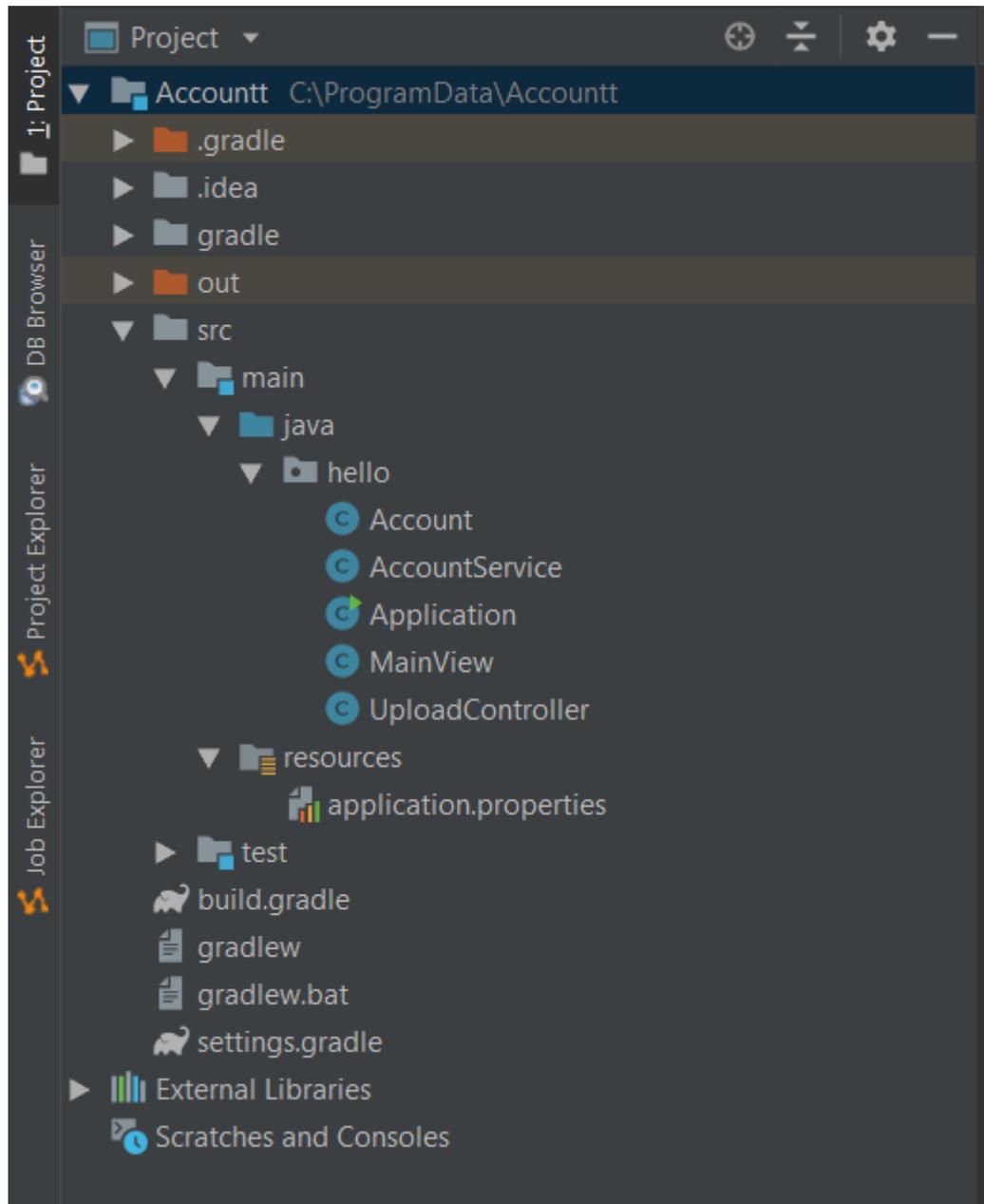


Figure 9. Directory Structure

The project is defined in Figure 12 by Gradle standards. The folders at the top of the project are the default project group for the application. For instance, the UI class has been located to the *src* directory. The project hierarchy displayed in the Project Explorer is shown in Account project.

4.3.2 Class Diagram

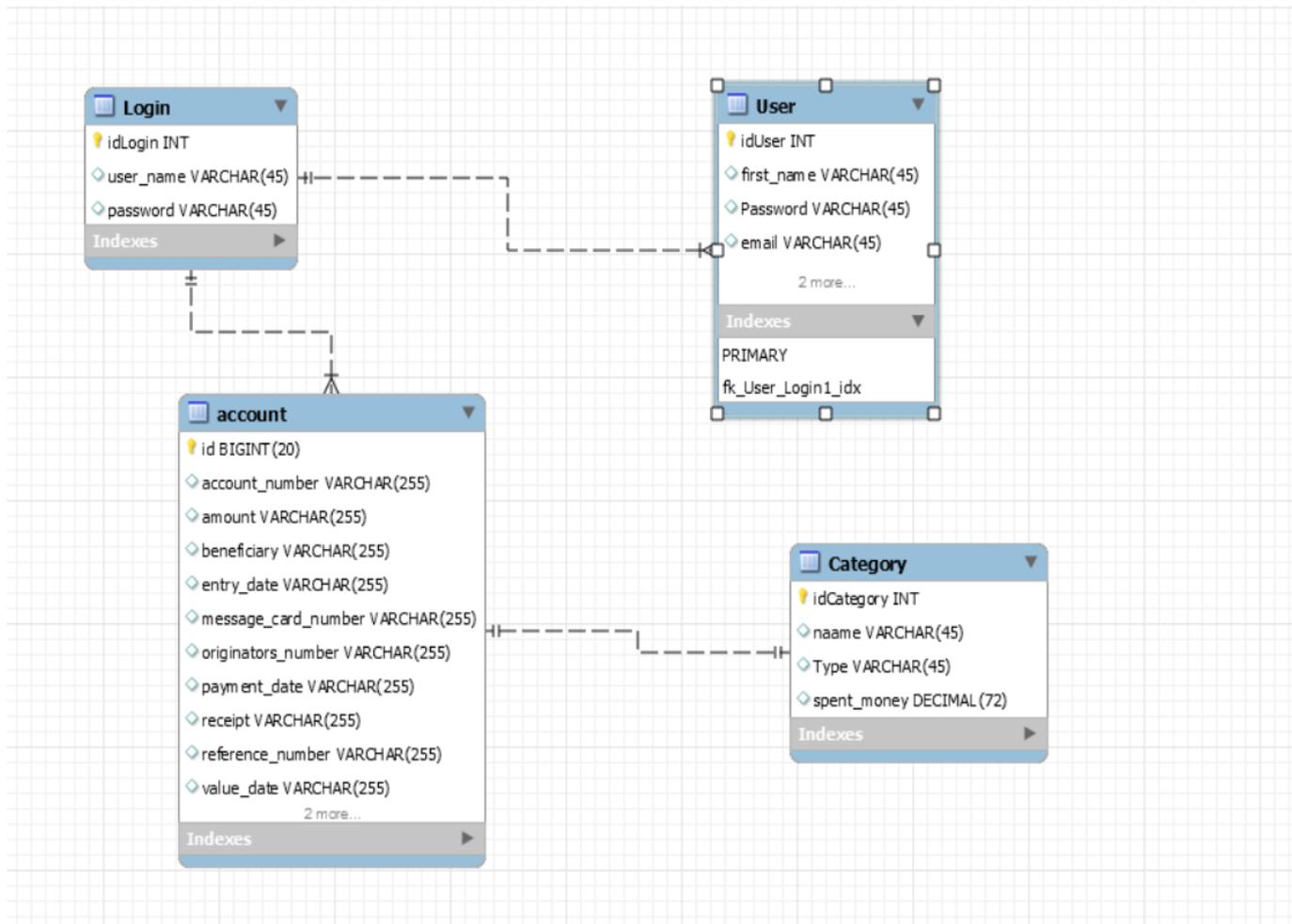


Figure 10. Class Diagram

Each one of the classes shown in Figure 13 contains various attributes and methods or functions. They are interconnected and share data by calling other class attributes.

The category uses an account to obtain access for the user respectively. It uses a username as a qualifier to select particular user. The attributes of account class represent the statement of the bank account which includes account number, amount, etc.

4.4 Data Persistence

Persisting data in an application can be executed by either storing it on the local host or transferring it to a webserver through a network connection. This application is implemented and designed to be server-less. To store the data MYSQL database was provided, and this is more suited to the project, and it uses indexes to speed up performance and sort data. /10/

5 RESULTS

5.1 Admin Dashboard

5.1.1 User Interface

The application consists of three pages of data entry and upload, the third page gives users the access and management rights and the opportunity to review the application. In this application, there are currently one type of user role, Admin. Admin at the moment is the main function. At the beginning the system will show the login page which requires the user to give the username and password. If the information is correctly given, then it will proceed to the dashboard. A few users are granted the access that have been stored in the database.

5.1.2 Login

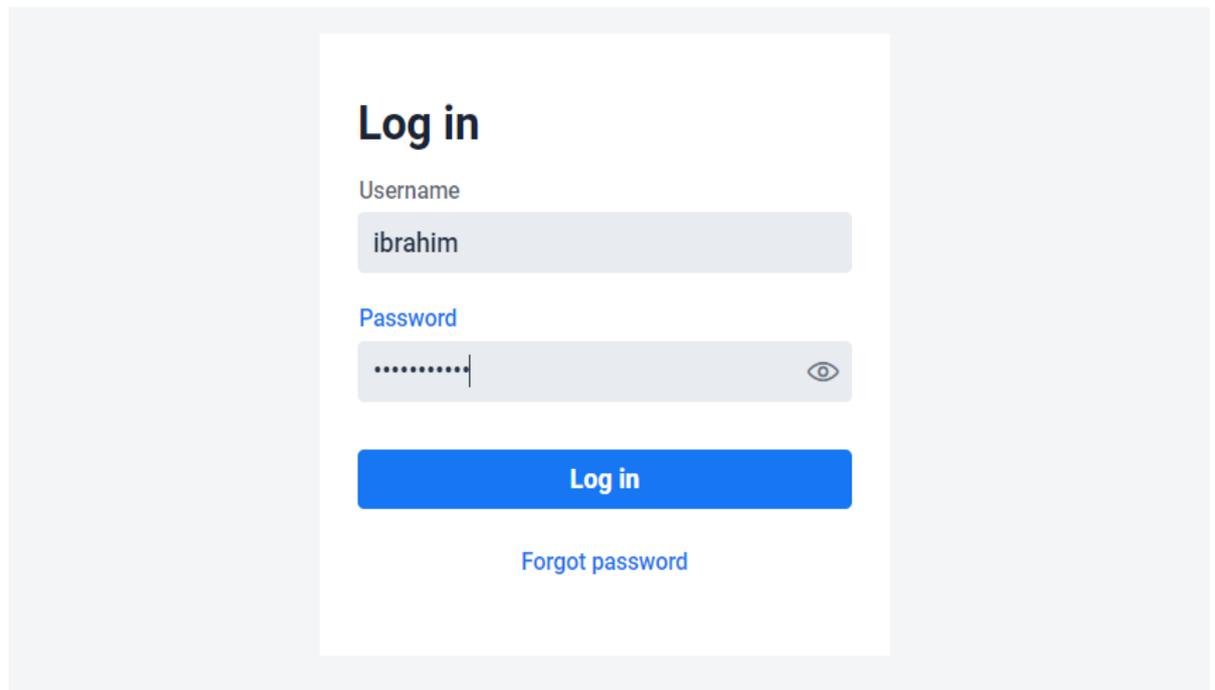
The image shows a login form with a white background centered on a light gray background. At the top, the text "Log in" is displayed in a bold, black font. Below this, there are two input fields. The first is labeled "Username" and contains the text "ibrahim". The second is labeled "Password" and contains a series of dots, with a small eye icon to its right. Below the input fields is a blue button with the text "Log in" in white. At the bottom of the form, there is a link that says "Forgot password" in blue text.

Figure 11. Login Form

First, the login panel will appear as shown in Figure 10. A unique name and password are required. The username and password can be later changed. If the login credentials are correct, the admin page as shown in Figure 11 will appear.

5.1.3 Admin Page

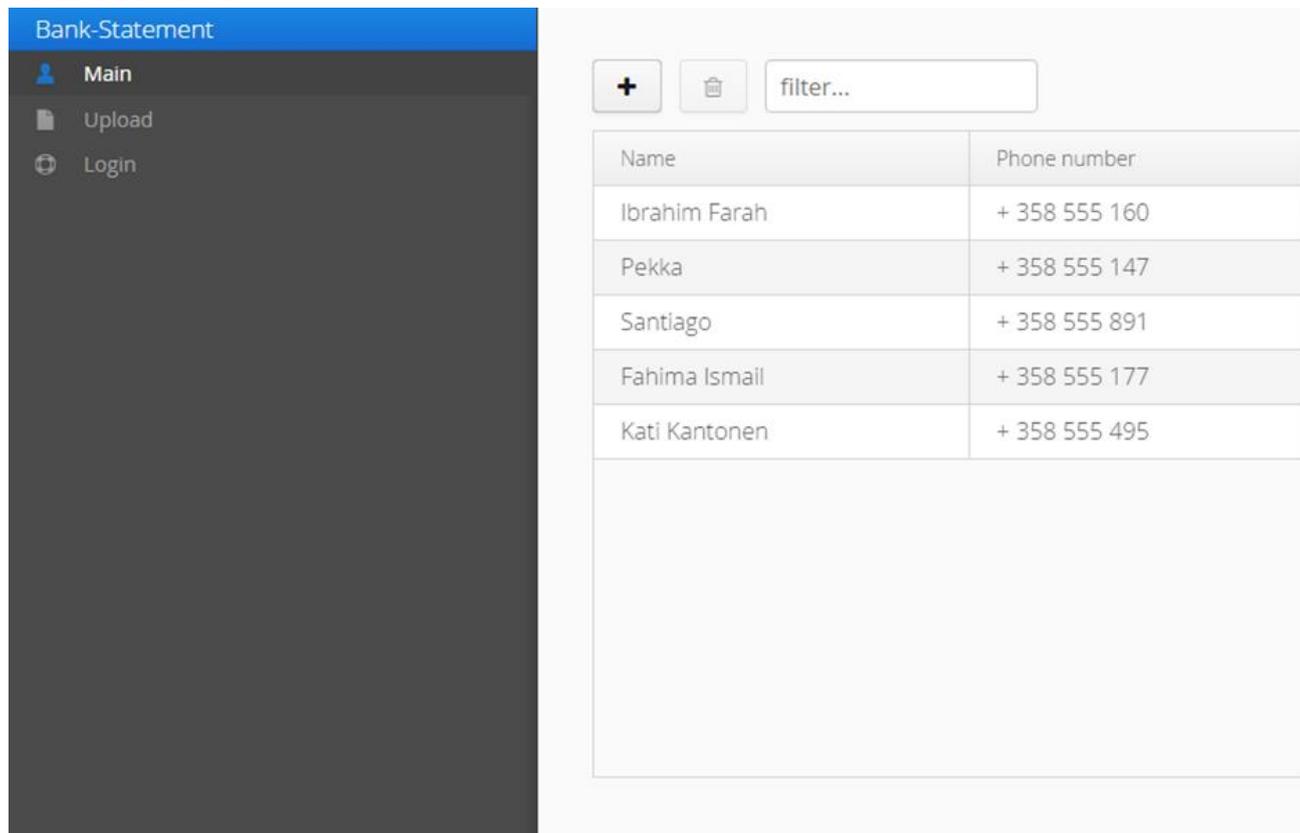
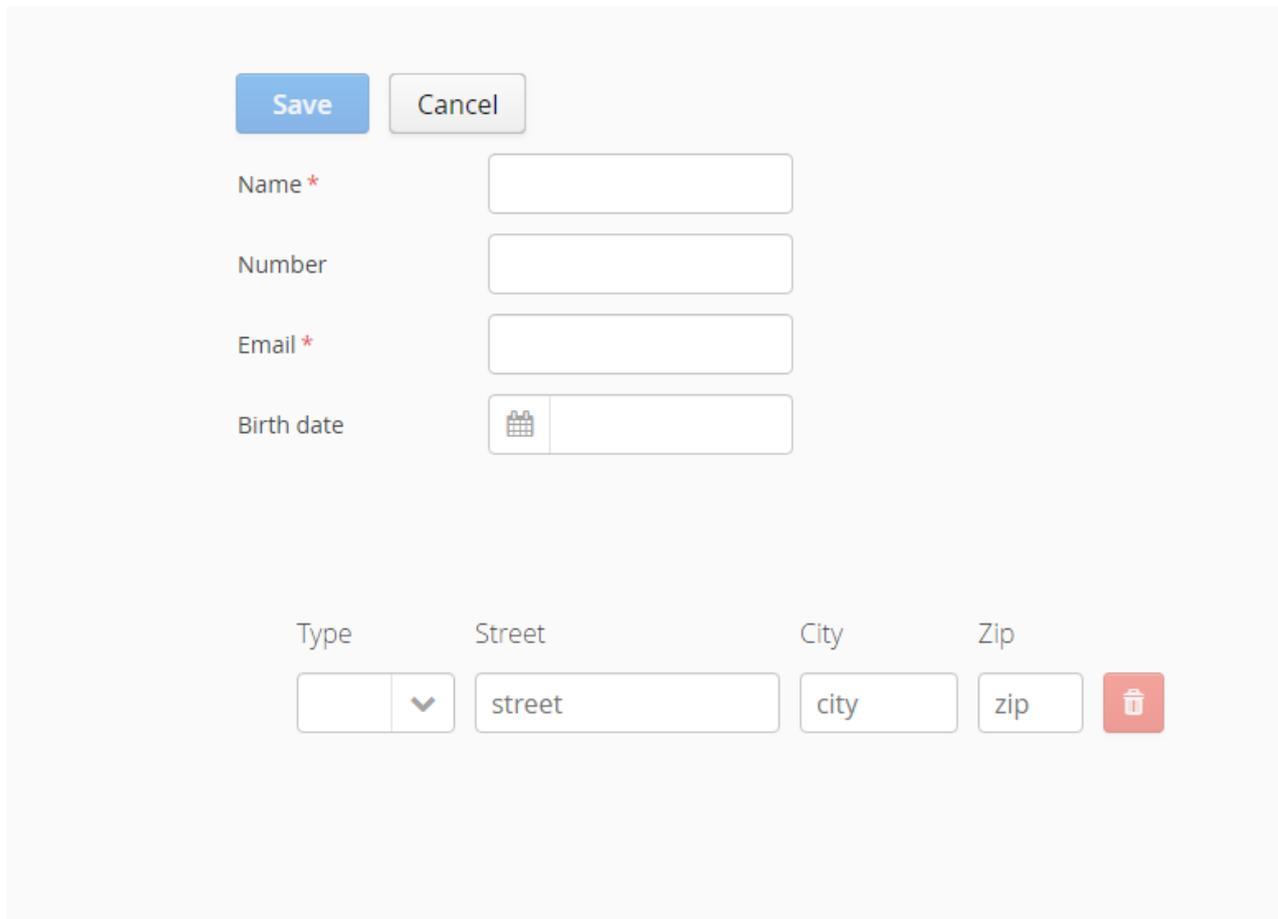


Figure 12. Admin Page

If the login information is correct, then main page as shown in Figure 11 will appear. On the left panel, the upload and login options will be shown. Here the user admin can see the list of user information that have been saved in advance and later could be edited or more user information added by clicking the add button. These users have a set of permissions, which depend on their role and function in the application.

5.1.4 User Registration



The image shows a user registration form with the following elements:

- Buttons: "Save" (blue) and "Cancel" (grey).
- Fields:
 - Name * (text input)
 - Number (text input)
 - Email * (text input)
 - Birth date (calendar icon and text input)
 - Type (dropdown menu)
 - Street (text input, containing "street")
 - City (text input, containing "city")
 - Zip (text input, containing "zip")
- Actions: A red trash icon button is located to the right of the Zip field.

Figure 13. User Registration

It is first necessary to enter personal data into the registration form to make the user information available on the main page list. This assigns users the access to upload pages, where each process done by that particular user in the upload will be kept for the user only, therefore data merging between two users should be intervened. On the other hand, it allows the user to keep track of its own data activity every time the user tries to login the application with the same credentials that are registered in the application as a member. The registration issues a certificate of recording for each username registered. Name and Email fields are mandatory to fill as shown in the figure, otherwise it will give an error message. Other fields, such as Phone Number, City, Birthday and Street-code are optional fields.

5.1.5 Upload Interface

Id	Entry Date	Value Date	Payment D...	Amount	Beneficiary	Account N...	Reference ...	Originators...	Message C...	Receipt
1	14/12/2015	12/5/2015	13/07/2018	540	VAASAN-OP...		45584-859855	1.48786E+11	ESPOO	14595659899
2	14/12/2015	12/5/2015	13/07/2018	540	VAASAN-OP...		45584-859855	1.48786E+11	ESPOO	14595659899
3	14/12/2015	12/5/2015	13/07/2018	540	VAASAN-OP...		45584-859855	1.48786E+11	ESPOO	14595659899
4	14/12/2015	12/5/2015	13/07/2018	540	VAASAN-OP...		45584-859855	1.48786E+11	ESPOO	14595659899
5	14/12/2015	12/5/2015	13/07/2018	540	VAASAN-OP...		45584-859855	1.48786E+11	ESPOO	14595659899
6	14/12/2015	12/5/2015	13/07/2018	540	VAASAN-OP...		45584-859855	1.48786E+11	ESPOO	14595659899
7	14/12/2015	12/5/2015	13/07/2018	540	VAASAN-OP...		45584-859855	1.48786E+11	ESPOO	14595659899

Figure 14. Upload

This is an example of seven different CSV file data entries uploaded and imported into the database with columns and rows, then the stored data is populated and viewed on the Vaadin container, as shown in Figure 13. The terms on the top of the table are the ones found in the uploaded file, while the terms found in the field row are account names. The terms are 11 including the Id field.

Table 3. Bank Statement

Id	The unique number that applications to each importing activity
Entry Date	Date that the transaction was handled by the bank
Value Date	Date that the transaction was handled by the bank
Payment Date	Date that the transaction was handled by the bank
Amount	The amount paid in the transaction
Beneficiary	Is the bank data which stated In the BSB code
Account Number	The IBAN specified in the payment

Reference Number	Is the number which is indicated on the invoice
Originator's Number	Is the number which is indicated on the invoice
Card Number	Is a credit card identification data with electronic payment orders.
Receipt	Payments sent to a receiver with receipt purpose code

6 TESTING

The methodology used in the development of each module demands a testing proceeded at each cycle of development where the developer tests all features and ensures that everything is working accordingly. Bugs are errors are fixed before proceeding to the next feature. During the development stage of each module a proper unit testing and integration testing techniques were applied. To initiate a testing session the Run and Debug commands are executed on the terminal which created a temporary configuration and is saved as a configuration dialogue. When the tests run in the background, it lists the directories and libraries on the console. If the test runs without errors, it means that the application can proceed to further development, but if there is an error, an output which needs to be fixed, then the event log will display those output directories which do not correspond to existing modules. These directories need to be cleaned.

```
Located MBean 'dataSource': registering with JMX server as MBean [com.zaxxer.  
Tomcat started on port(s): 8080 (http) with context path ''  
Started Application in 29.74 seconds (JVM running for 32.481)  
Customers found with findAll():  
-----
```

Figure 15. Console output

This is a test output displayed on the terminal after the test method is executed on the application by using a run with a coverage test. This event log shows that everything is working and all files are up-to-date.

```
! Error:(33, 42) java: cannot find symbol
  symbol:   variable customer
  location: class hello.Application
! Error:(48, 41) java: cannot find symbol
  symbol:   method findByLastNameStartsWithIgnoreCase(java.lang.String)
  location: variable repository of type hello.CustomerRepository
C:\ProgramData\gs-vaadin-with-crud\src\main\java\hello\CustomerEditor.java
! Error:(56, 21) java: cannot find symbol
  symbol:   variable gfj
  location: class hello.CustomerEditor
```

Figure 16. Console output. The console displays errors, which means that the test is failed.

7 CHALLENGES AND CONCLUSION

Each project comes with its own special variety of challenges, and this project is not an exception. Whether it is data processing or designing management, each process needs to be well structured and organized. Although all the necessary ideas and theories for this was project were available, nevertheless more time was needed in order to achieve project objectives and deliverables. In terms of productivity sophisticated tools were used to make the application more productive, so that the workflow would be organized in the most efficient way.

Integration testing revealed errors with interfaces between different program components before the deployment. Moreover, testing can demonstrate different issues which consumes time and effort. Commonly slow applications are not appreciated. The speed of the application is defined by the requirement of the user. Performances usually get are weakened by underestimating application requirements and overextending its features.

The knowledge of frameworks or platforms are the beginning for development methodologies. They facilitate the boost performance. Some of the frameworks are more complex than others, some are flexible. Skills and have a clear understanding of their functionalities is required.. As this project was done individually without collaboration from an external assistant, the Spring Boot and Vaadin framework were needed to get familiar before using them.

The main objective of this project was to provide an opportunity on account data solution. An application is built and the reason is to make personal finance easier to manage using the mechanization of regular steps that people go through when doing a budgeting. Particular design and implementation challenge we addressed, emphasizing the key context and technical knowledge that was obtained in order to overcome the challenges.

The purpose of the project was to create an application that implemented an instinctive way to display and manage personal finance and precisely predict a user future transactions foundation on their history.

The limitation of the project was receiving a clean CSV file without corrupted data inside it. The file needs to be opened in an editor before importing into database, and to manipulate the data to remove any unnecessary fields in order to get the best result. On the other hand, throughout the development new technologies and techniques were studied continuously. The time needed to implement this project was underestimated. Despite the fact some improvements have been introduced, but the application partially meets all the requirements and requires enormous number of works to be a completely functional application.

REFERENCES

- /1/ Spring Boot framework. Accessed 18.5.2019. <https://spring.io/services>
- /2/ Auto-configuration. Accessed 18.5.2019. <https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-auto-configuration.html>
- /3/ Vaadin Vaadin-architecture. Accessed 18.5.2019. <https://vaadin.com/components>
- /4/ Gradle. Accessed 18.5.2019. <https://gradle.org/features/#build-scans>
- /5/ Accessing data with Hibernate and MYSQL in Spring Boot. Accessed 18.5.2019. <https://spring.io/guides/gs/accessing-data-mysql/>
- /6/ Spring Boot application.properties file (JPA) . Accessed 18.5.2019. <https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>
- /7/ Vaadin Framweork Data binding. Accessed 18.5.2019. <https://vaadin.com/forum/category/1007558/04-framework>
- /8/ Nordea Bank statement as CSV file format. Accessed 18.5.2019. <https://solo1.nordea.fi/nsp/login>
- /9/ Receiving Vaadin Upload. Accessed 18.5.2019. <https://vaadin.com/components/vaadin-upload/java-examples>
- /10/ Data persistence. Accessed 18.5.2019. <https://dzone.com/articles/what-is-persistent-data>