Nguyen Nhat Minh

Building a component-based modern web application: full-stack solution

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

21 September 2018

Metropolia

Author	Nguyen Nhat Minh
Title	Building a component-based modern web application: full-stack solution
Number of Pages	55 pages
Date	21 September 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen, Head of Department (ICT)

This thesis paper aimed to review relevant literature and build up a theoretical background for practical code implementation and analysis about software engineering in general and specifically full-stack web development. A relatively modern choice of full-stack web development with MySQL, PHP (CodeIgniter 3), JavaScript (in KnockoutJS), HTML5 and Bootstrap3 will be reviewed in this study.

Software engineering is a comprehensive discipline; building robust and reliable software requires not just learning and doing but also questioning existing theories. Why build software by using specific tools and methods instead of choosing others? This study intends to review some industry standards of software-building process and explore component-based thinking and advantages of using software frameworks in web development.

The case study for this project is Aurora Exchange Oy, a peer-to-peer lending platform and an ideal start-up to be analysed, as its software is in a rapid development process and each implementing software feature must be delivered quickly without sacrificing quality with minimum testing effort.

Keywords	AuroraX, Web, MySQL, PHP, JavaScript, CSS.

Contents

List of Abbreviations

HTML	Hyper Text Markup Language
CSS	Cascading Style Sheet
JS	JavaScript
AJAX	Asynchronous JavaScript and XML
SQL	Structured Query Language
JSON	JavaScript Object Notation
XML	Extensible Markup Language
HTTP	Hyper Text Transfer Protocol
REST	Representation State Transfer
DOM	Document Object Model
API	Application Programming Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
MIME	Multipurpose Internet Mail Extensions
MVC	Model View Controller
CRUD	Create Read Update Delete
POJO	Plain Old Java Object
JDBC	Java Database Connectivity
JDO	Java Data Object
CI	CodeIgniter
CGI	Common Gateway Interface
SMTP	Secure Mail Transfer Protocol
SDLC	Software Development Life Cycle
GUI	Graphical User Interface
UI	User Interface
HCI	Human Computer Interface
UCD	User Centered Design
ROI	Return on Investment
GPD	Gross Domestic Product
IoC	Inversion of Control
DI	Dependency Injection
XSS	Cross-site scripting
SPA	Single-page application

1 Introduction

Technology is advancing rapidly, and software is the core element of the trend; after every few years new software frameworks become available, a new programming paradigm appears and the choices for doing software development become more varied. Choosing a right software solution for a project will essentially determine deliverability and quality for the software and eventually contribute to the success of the project.

From enterprise to small-and-medium company to start-up, each entity contains its requirements and software development cycle which is comparable to the others but not the same, however. A software development process is often more formal and structured in an enterprise than another one in a start-up. This thesis paper concentrates on reviewing architecture design in a start-up and explore a few implemented features to see how software engineering practices were applied in the environment.

The case study company is AuroraX, a start-up with a few years of existence. AuroraX was established in 2012, four years after the financial crisis in 2008. The company aims to take part in the consumer lending market which were dominated by credit institution and small-loan companies. According to “suomenpankki.fi.” [1], the stock of household consumer credit Finland was almost €19 billion which is roughly 8.7% of Finland GDP ($251.88 billion from “tradingeconomics.com” [2]) in 2017; although only a small portion of the total household consumer credit contributes directly to Finland GDP, the proportion of the consumer credit’s size to the GDP size is substantial. In traditional consumer lending, taking a loan takes unpredictable time, interest rate is determined by one side (often the lenders), and no opportunity exists for normal people to invest in loans of other people. To solve those issues, AuroraX targets to facilitate lending money activities from investors to borrowers, serving well interests of both parties. On Aurorax’s platform, borrowers can consider offers with different interest rates from investors, and investors can diversify and reduce risk by invest in multiple borrowers with different interest rates in different risk rate categories that they feel comfortable with. The company’s first-step operations together with other peer-to-peer lending businesses’ activities hopefully will prove a new sustainable business model which contributes positively to financial systems of Finland and other future operating countries.

Building AuroraX’s software comprises the challenge of resource constraints of a start-up, each implementing feature required independent thinking from a developer and test

activity is often minimal. From the challenge, AuroraX's development team sees that only truly agile development and fast-to-react development method will work. Uncertainty is a part of daily work and the AuroraX' development team prioritizes only features which contribute to the immediate better user experience.

2 Theoretical background

This chapter reviews theory related to server-client operations, relevant technologies as well as some popular web development frameworks and software design patterns which are applicable in a start-up context.

2.1 Client and server model

Client/Server computing is the technology that resolves many modern organizations' data management problems. Client and server are two independent processes which are defined in the Client/Server computing model. A Client requests services from the server process and a Server provides requested services for the Client. [3, p. 1.]

In various kinds of server such as file server, print server, application server etc., the web server is pertinent in the context of this thesis paper. In web server, web application provides access to data and documents for clients, which often request through a web browser to the server by HTTP protocol. The medium for transfer data between those two processes is a network, which can be an organization's intranet or the internet, depend on purpose of the web application. [3, p. 1.]

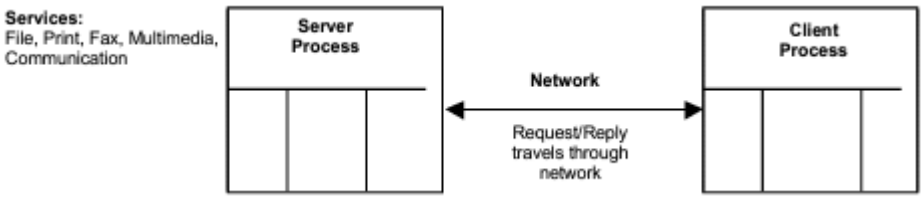


Figure 1. Basic Client/Server computing model – reprinted from [3, p. 1].

Figure 1 illustrates that a variety of computers in the network can provide services. The medium where request process happens is the most important element here. For example, in the Client/Server Database system, the functionalities come from both a server

system and multiple clients where some task can be performed on the client system through the network of computers. [3, p. 2.]

For a web application, developing the client side requires using front-end technologies and for the server side, back-end technologies are needed. In the next two subsections, front-end and back-end technologies will be introduced to continue building up theoretical background for this thesis paper.

2.2 Front-end technologies

HTML is the industry standard mark-up language for structuring a website on the World Wide Web. HTML5 is the latest version of HTML and it contains larger number of supported technologies to provide richer user experience. HTML5 consists of many functionalities and those can be divided into groups base on their function such as “connectivity, semantics, offline and storage, multimedia, 2D/3D graphics and effects, performance and integration, device access, styling”. With HTML5, a developer can define more clearly the content of a site, ensure seamless communication between server and client, store data on the client-side locally and enable video and audio to user. Furthermore, the developer can present 2D/3D effect on a site, utilize more effective computer resource, build application for different devices and write more sophisticated styling. [4.]

In a commercial website, HTML would not be sufficient to represent complex data in a professional and coherent format. CSS is the technology that used to describe how elements of a document such as HTML will be rendered on a user’s screen. CSS also enables the reusability of a stylesheet as the same one can be used with different HTML documents.

To “make webpages alive”, JavaScript was invented in 1995. It is a multipurpose programming language which is best known as the scripting language for web pages or for the front-end part of a website. JavaScript can handle user interaction on a site such as adding new HTML elements to the page, changing existing content and styles, reacting to user actions, sending network request through AJAX, getting and setting cookie, access data on the client-side (“local storage”). JavaScript conforms to ECMAScript (ES) specification with most features (depending on the ES version) supported by all major browsers. At present, the programming language is executable on server side or any device with a JavaScript engine (V8, SpiderMonkey etc.). [5.]

HTML, CSS and JavaScript together can solidly construct the front-end part website for presenting data purpose. However, without back-end technologies, a website limits to displaying data, responding to user interactions and processing simple data on a device's browser. For most of commercial web applications, back-end technologies are essential.

2.3 Back-end technologies

Back-end technologies were developed to facilitate the saving, processing and retrieving of data for web applications from the server side. A database server, an application server and a server-side language are core elements of a website's backend technical stack.

The application server contains the role of performing all operations between backend business applications or databases and users. Some popular choices of development are Apache HTTP, Apache Tomcat, IBM WebSphere, JBoss, Glassfish and Internet Information Services (IIS).

The database server can be Oracle, Microsoft SQL Server, Informix, Oracle, DB2, PostgreSQL, Ingres, MySQL etc. SQL Server is a "relational database management system" (RDBMS) which is a bundle of services working together to help retrieving, storing data easier. SQL Server was developed by Microsoft to compete against Oracle which is the dominant server until the 1980s. The first major version of SQL Server was 7.0, which was released in 1998. [6, p. 2.]

343 systems in ranking, August 2018

Rank Aug 2018	Rank Jul 2018	Rank Aug 2017	DBMS	Database Model	Score Aug 2018	Score Jul 2018	Score Aug 2017
1.	1.	1.	Oracle	Relational DBMS	1312.02	+34.24	-55.85
2.	2.	2.	MySQL	Relational DBMS	1206.81	+10.74	-133.49
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1072.65	+19.24	-152.82
4.	4.	4.	PostgreSQL	Relational DBMS	417.50	+11.69	+47.74
5.	5.	5.	MongoDB	Document store	350.98	+0.65	+20.48
6.	6.	6.	DB2	Relational DBMS	181.84	-4.36	-15.62
7.	7.	↑ 9.	Redis	Key-value store	138.58	-1.34	+16.68
8.	8.	↑ 10.	Elasticsearch	Search engine	138.12	+1.90	+20.47
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	129.10	-3.48	+2.07
10.	10.	↓ 8.	Cassandra	Wide column store	119.58	-1.48	-7.14
11.	11.	11.	SQLite	Relational DBMS	113.73	-1.55	+2.88
12.	12.	12.	Teradata	Relational DBMS	77.41	-0.82	-1.83
13.	13.	↑ 16.	Splunk	Search engine	70.49	+1.26	+9.03
14.	14.	↑ 18.	MariaDB	Relational DBMS	68.29	+0.78	+13.60

Figure 2. Ranking of popular DBMS – reprinted from [7].

Figure 2 demonstrates ranking of popular Database Management System (DBMS) based on several relevant metrics on “db-engines.com”. According to the site [7], six variables were taken into consideration for ranking:

- “Number of mentions of the system on websites” (search engine queries on Google, Bing, Yandex).
- “General interest of the system” (Google Trends).
- “Frequency of technical discussions about the system” (on Stack Overflow and DBA Stack Exchange).
- “Number of jobs offers, in which the system is mentioned” (on Indeed and Simply Hired).
- “Relevance in Social networks” (number of Twitter tweets).

The server-side programming languages can be PHP, Ruby, Python, Java, Scala, C# (.NET), JavaScript etc., each of them consists of different characteristics and the implementation choice depends on a developer team’s preference and competence.

2.4 Popular web development frameworks

Web development framework owns a narrower definition than software framework, so it is ideal to first clarify the second definition. Techopedia defines software framework as “a conceptual or concrete platform where generic functionality can be selectively specialized or overridden by developers or users”. Frameworks comprises of libraries where

application programming interface (API) is reusable inside the developing application. [8.]

Some important features of a software framework following Techopedia [8] are described below:

- Default Behaviour: a framework consists of default behaviour to user interaction.
- Inversion of Controller: the framework defines the global flow of control rather than the caller.
- Extensibility: a framework can be extended but cannot be modified to maintain the purposes of the framework as simplifying development environment, enabling developers to focus on project requirements rather than rebuilding repetitive functions and libraries.

Web development framework consists of those features which serve a smaller environment, applying to web applications. For both front-end and back-end technologies, numerous (web development) frameworks enable rapid development, promote the reusability and enhance quality of software. Those frameworks are often open-source meaning that the source codes are visible for anyone who is interested in reading, testing and contributing. Therefore, large communities of developer are able to review and contribute to fixing bugs and releasing new versions for those frameworks.

Bootstrap, Foundation and Material are popular front-end framework choices at present. Bootstrap is a famous example front-end framework, as built-in stylesheet classes can be reused as many times as needed, helping in rapid prototype and development of a website.

In back-end development, various frameworks are available for back-end language such as Laravel, Zend, Symphony, CakePHP, CodeIgniter etc. for PHP; Django, Flask etc. for Python and JavaServer Faces, PrimeFaces, Spring, Struts etc. for Java.

2.5 Software development cycles and development methodologies

Most software projects use the same standard development practices, which is referred to as the “software development life cycle (SDLC)”. Waterfall and agile are two main methodologies and each one contains many variations.

2.5.1 Waterfall development model

The waterfall model is a method that consists of origin from other processes of engineering such as manufacturing and construction. Each step of the process is “well-defined” and must be completed before continuing to the next one. [9, p. 37.]

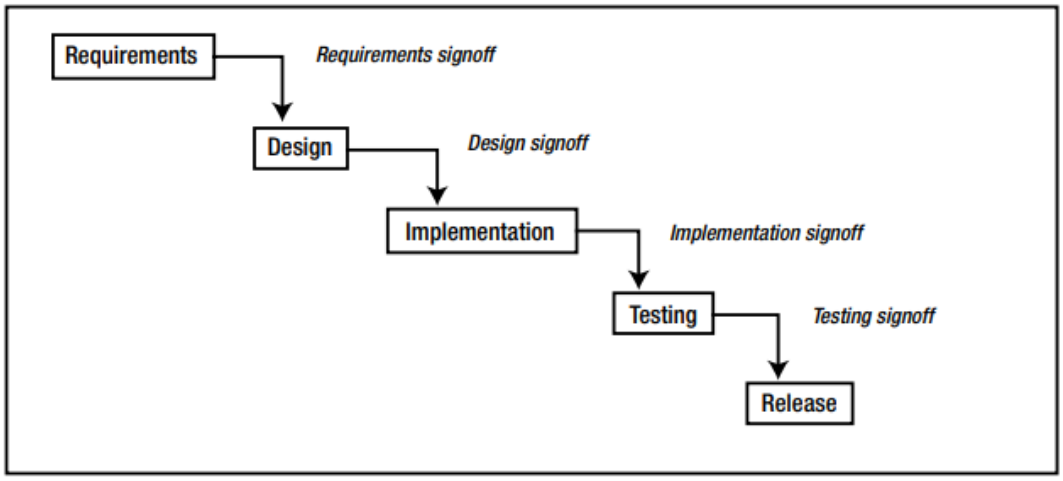


Figure 3. Waterfall process – reprinted from [9, p. 38].

The waterfall methodology requires heavy documentation as it often involves independent teams in development of each phase. Requirements must be documented fully for all needed functionalities by business users before design phase can start. In design phase, often technical manager, business manager and project manager review the design from the technical team before moving to implementation phase. The review will produce a “Gap Analysis” document which produces the requirements that are not resolved by the design. [9, p. 38.]

In project implementation, Gantt charts help project manager keep track of details of the work to be carried out. Several Gantt charts can exist at the same time and one of them can be dedicated for tracking coding effort. Therefore, status of implementing items and progress on each of them together with dependencies between can be easily visible for every stakeholder of a project. Furthermore, a regular project meeting involving development team and managers carry out the mission of updating the charts. [9, p. 38.]

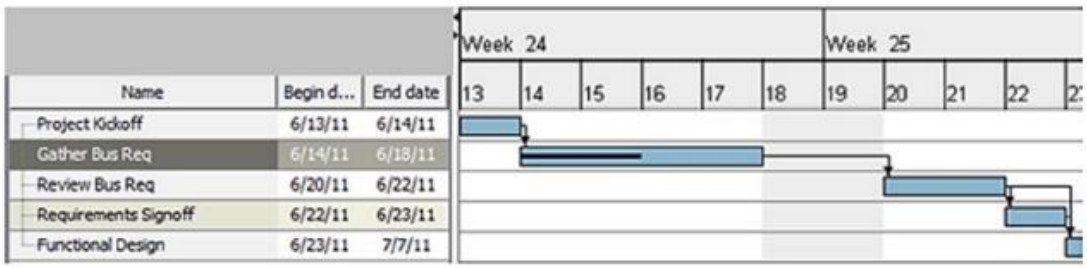


Figure 4. An example of a Gantt chart – reprinted from [9, p. 39].

With waterfall model, testing and verification take part late in a project. Base on the requirement documents, a dedicated testing team will execute a test plan to make sure all requirements are implemented correctly, and system works as expected. Testing consists of several types and depending on the project. Not all of them need to be executed every time.

Table 1. Testing types and descriptions– reprinted from [9, p. 39].

Testing type	**Description**
Regression Testing	If changes appear in implemented code, all old test cases must be re-executed to make sure new changes do not break the old code-base.
User Acceptance Testing	Business users will take part in this test to see all defined requirements are implemented.
Integration Testing	Testing the correct interfacing between the testing system and external systems or between sub-systems in the same system.
Black Box Testing	The tester is not permitted to access to the codebase, testing is conducted by the tester from the view of users.
White Box Testing	The tester knows about code implementation of the project and can provide insights about reasons of a bug.
Performance Testing	Usually testers test load to see how many users that system can handle in a timeframe (10 seconds, a minute etc.) and see potential scalability issues.

When all test activities are carried out, the final product can be released to the end users. Maintenance often happens in this phase, and minor changes/enhancements are introduced. On theory, testing in waterfall model can take advantage of a well-defined requirement documentation from the previous stage to build a well-structured test plan and estimate exact needed testing effort, hence conclude a successful software project. In a

real-life software project, requirement changes are often introduced at a later stage of a project, causing difficulty in testing, cascading effect of delaying and late delivery as the whole rigid model must start over. As a result, the testing team must wait until new requirements are conducted. The disadvantage can be mitigated by performing the model on a smaller scale with many short iterations, this method is named as “iterfall”. [9 pp. 39-40.]

2.5.2 Agile development model

In contrast to Waterfall development model, Agile development model embraces changes in a software project. The model implements practical process control, where “frequent feedback and adaptation” decides direction of development rather than strictly-defined documentation. Scrum is considered one of the most popular agile framework; in fact, many text books dedicate to write about Scrum. The framework contains the objective of managing releases of software in short interactions, where timeline of development can be precisely scheduled. Scrum’s goal is to produce software in short sprints, each sprint delivers something that can be tested. To achieve the goal, Scrum aims to reduce hierarchical structure of a team, the three simplified roles are: the product owner, the development team and the Scrum Master. [9, p. 40.]

The Product Owner is an individual who possesses vision and need of building a software for solving customer problems. The person does not own the competency to build the software, and s/he needs to fulfil the role of creating requirements and keeping track as well as prioritizing and organizing each requirement. Input from the Product Owner is only needed when the development team asks. [9, p. 41.]

The Scrum Master owns only a role of facilitating communication and coordination between the Product Owner and the product development team. The person runs necessary meetings for the Scrum process and manage to ensure conformation to Scum rules such as time limits, speaking restriction etc. The development team in Scrum contains no structure and requires doing self-organizing. Scrum rules may be enforced by the Scrum Master; however, no other people intervenes in the development except the development team itself. A cross-functional team is encouraged with members who possess various expertise such as technical writer, back-end developer, front-end developer, designer, tester etc. but limit to those who are essential for completing a sprint. The team might need input from outside in the process of a sprint such as clarification

from production owner about a requirement or the help of the Scrum Master if slow response from an external party exists. [9, p. 41.]

It is important to ask what the typical characteristics of a sprint are and what the detailed process behind it is. Scrum's time length varies between two to four weeks. Pros and cons exist in exceedingly long or exceedingly short the length, as the short timespan makes delivery difficult, and the long timespan lengthens the reacting time to requirement changes. A sprint starts with a "Spring Planning Meeting" with limit dedicated time between development team and the product owner to produce a sprint backlog and a product backlog.

A product backlog is created by the product owner, and it includes statements about the finished product. Those statements are called "stories", which will need to be selected for prioritizing and then clarifying during the first half of the sprint planning meeting. The development team needs to estimate how much effort for each story and how many story points can be finished in the next sprint. The selected subset of stories will be considered further in the second part of the meeting for planning the sprint backlog. The prioritized story items are analysed to present more details for creating a sprint backlog, and later each item will be assigned to a development team member with estimated time for completion. [9 pp. 41-42.]

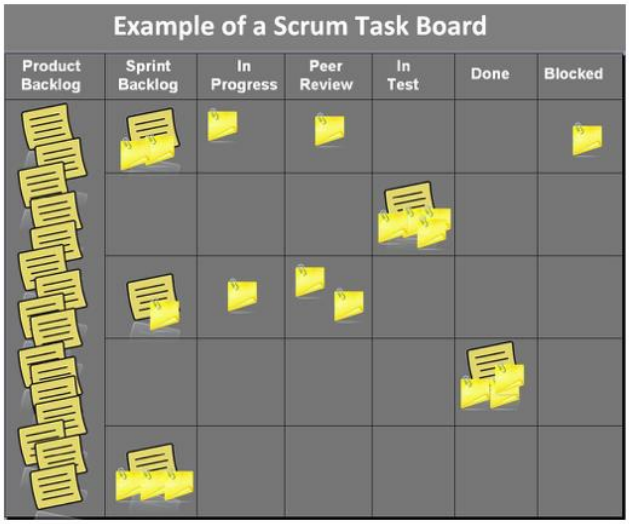


Figure 5. Scrum task board example – reprinted from [10].

To effectively manage the sprint backlog items, spreadsheet or other special software can be used. Another popular choice of use is sticky notes or index cards on a whiteboard

with one item on a card (figure 5). When a sprint starts, the development team and the Scrum Master do daily scrum. This meeting usually lasts for about 15 minutes, in which each development team member answers few questions such as what a person has been working on since the last meeting, what is s/he planning to do before the next meeting and had s/he experienced any difficult in completion of the personal task. [9, p. 42.]

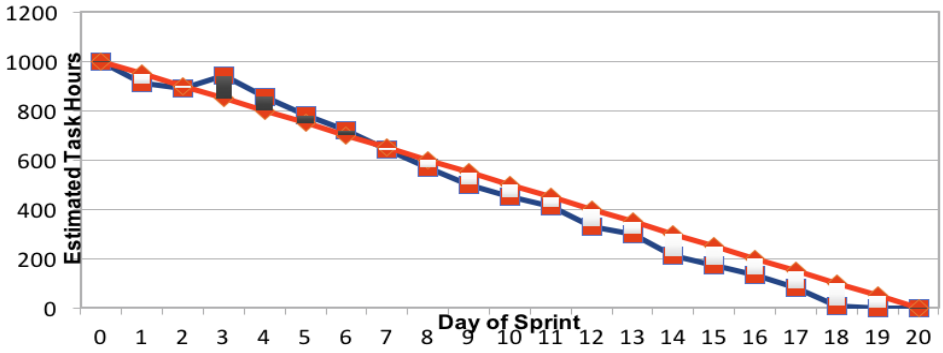


Figure 6. Burndown chart example – reprinted from [10].

To track development progress, a burndown chart can be constructed with completed items and time on two axes of the chart (figure 6). That brings visibility of number of items/task hours to be conducted versus time and how fast the development speed as well as the overall progress of the project. The Scrum Master will gain confident in understand is that the project on track and s/he will be able to communicate to the product owner through the burndown chart. [9 pp. 42-43.]

When each sprint ends, product owner will be demonstrated the completed working functionality in a time-boxed meeting. The meeting aims to gather feedback about the completed stories and serve few meaningful purposes. Those purposes include: demonstrating progress is made; demonstrating in an incremental manner each completed functionality; misunderstanding can be verified easily if the implementation is different from the story when everyone retains a fresh memory about that and wasted time is limited to only one sprint; priorities can be remade frequently. [9 pp. 43-44.]

An additional meeting can be organized which is the “sprint retrospective” which aims to review the sprint to see what the team did well, or other improvements can be made in the next sprint. This kind of meeting is required after every sprint in the beginning of the project, later the meeting can be organized less frequent and only when requests exist. Scrum does not provide details for actual development, it often combines with other

practices such as extreme programming. In addition, Scrum can be used by the development team in small scale in large organization where the waterfall method is deep-rooted in the project management structure. [9, p. 41.]

2.6 Popular software development patterns

Software development patterns aim to solve common software development problems, based on popular-known terminology and implementation when the problems are recognized. One of the earliest definitions of software patterns comes from “Design Patterns: Elements of Reusable Object-oriented Software (Gamma et al., 1994)”, referring to as the “Gang of Four” (GoF). In the later part of this section, few popular architectural patterns will be reviewed. The reviewing patterns include observer, façade, singleton which are divided into categories of behavioral, structural and creational, respectively. In addition, two enterprise patterns: Model-View-Controller and Inversion of Control will be discussed. [9, p. 19.]

2.6.1 The Observer Pattern (Behavioral)

The observer pattern contains of two components, one of those exposes an event that the other can register to listen for and will be notified when event changes from the first component happen. When the notification is not needed, the observers can unregister to save computing resources. [9, p. 19.]

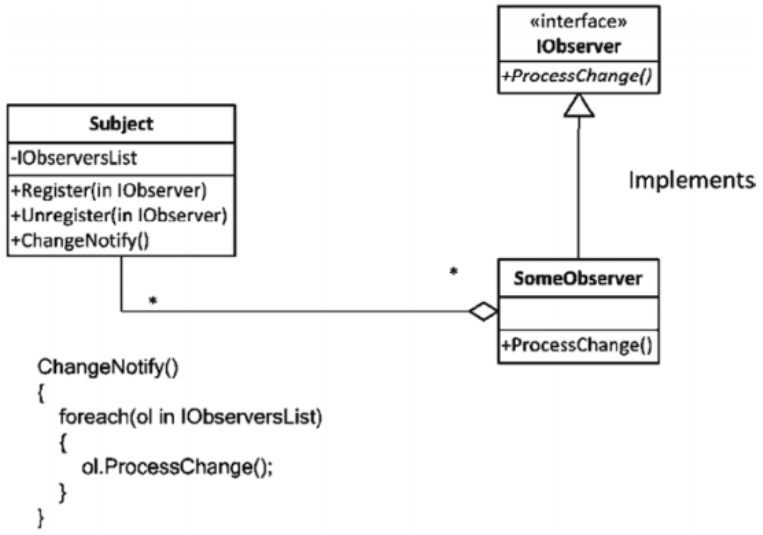


Figure 7. Observer diagram – reprinted from [9, p. 20].

Figure 7 reveals structure of the observer pattern in which an observer implements an interface that was defined by the Subject class. The “ChangeNotify” method loops over a list of observers to call the method on each of them. If needed, the subject can implement an interface for receiving notification from the observers and trigger desired functions. Some examples include live football updates, live weather updates, instant messaging, chat rooms etc. [9, p. 20.]

```
<div ng-app="myApp" ng-controller="myCtrl">
    Name: <input ng-model="firstname">
    <h1>{{firstname}}</h1>
</div>
<script>
    var app = angular.module('myApp', []);
    app.controller('myCtrl', function($scope) {
        $scope.firstname = "John";
    });
</script>
```

Figure 8. An example of AngularJS data-binding [11].

Figure 8 demonstrates a popular example of using AngularJS (observer pattern), any text input from user in the input field will be reflected in the <h1> element, the observer (input) will notify the subject (h1) and trigger change event on the subject whenever a change in the input field happens. Initially the h1 element will contain text value of “John” resulting from setting the “myCtrl” object property “firstname” to “John” in the script part.

```
<p>Login name: <input data-bind="value: userName" /></p>
<p>Password: <input type="password" data-bind="value: userPassword" /></p>
<script type="text/javascript">
    var viewModel = {
        userName: ko.observable(""),        // Initially blank
        userPassword: ko.observable("abc"), // Prepopulate
    };
</script>
```

Figure 9. An example of KnockoutJS data-binding [12].

Figure 9 illustrates a similar example of implementing observer pattern on two input fields, in which username input field is initially empty and password input field contains value of string “abc”. Any changes in those input fields will update the “userName” and “userPassword” observables to hold to the same values.

2.6.2 The Facade Pattern (Structural)

The facade pattern aims to provide simplified external interface for a complex internal system. The internal system can be built by small independent components for better performance, easier testing and other purposes. [9, p. 20.]

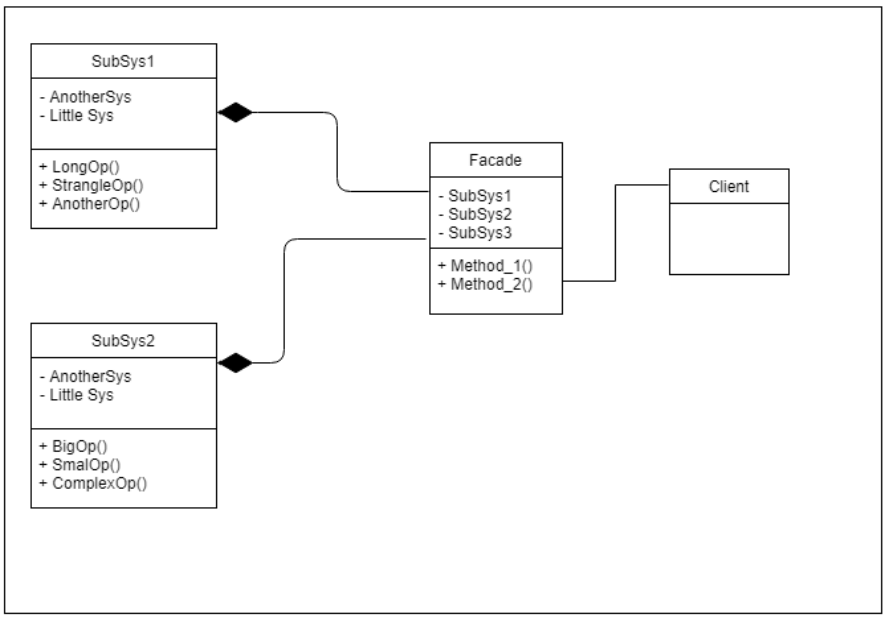


Figure 10. Façade pattern structure – modified from [9, p. 21].

In figure 10, Method_1() and Method_2() will be available for client which will do not necessarily concern with the complexity of the SubSys1 and SubSys2. The client can query SubSys1 and SubSys2 if the intention of a method does not change, even when changes in the subsystems appear. This is structural pattern as it does not necessarily solve a functional problem and components can be re-arranged to achieve the result. [9, p. 21.]

2.6.3 The Singleton Pattern (Creational)

The Singleton Pattern is widely-used in Java and C#, which guarantees that only an instance of a class exits at runtime. This is achieved by using private constructor and static “Create” or “getInstance” methods. This is useful when a resource-heavy component is guaranteed not to be constantly created and destroyed. A new object created from a class will always refer to an existing instance. [9, p. 21.]

The Singleton pattern is an example in Creational category; another example is Factory pattern. The Factory pattern helps to create efficiently instances of a class, depend on different parameters pass in. Beside Singleton and Factory patterns, the Builder pattern (in Singleton category) facilitates the creation a complex object from different sub-objects and the pattern's user does not necessarily concern with how the object is created. [9, p. 21.]

2.6.4 Model-View-Controller (Enterprise Patterns)

Most software design patterns are useful and applicable, however Model-View-Controller (MVC) pattern and Inversion of Control (Dependency Injection) are especially abundant in any distributed and/or enterprise environment. Those patterns contribute greatly to building a component-based software system.

MVC is one of the oldest software design patterns and is still being used in many web and mobile applications. Each component in the pattern possesses its own concern and single responsibility. The Model defines data in an application and aware of changes of data's state. The View display system's data and accepts inputs through some Graphical User Interfaces (GUI) or another non-human interface. The Controller obtains input from the view and queries data from the model; base on defined business logic, actions will be handled on the Model or the View. [9, p. 22.]

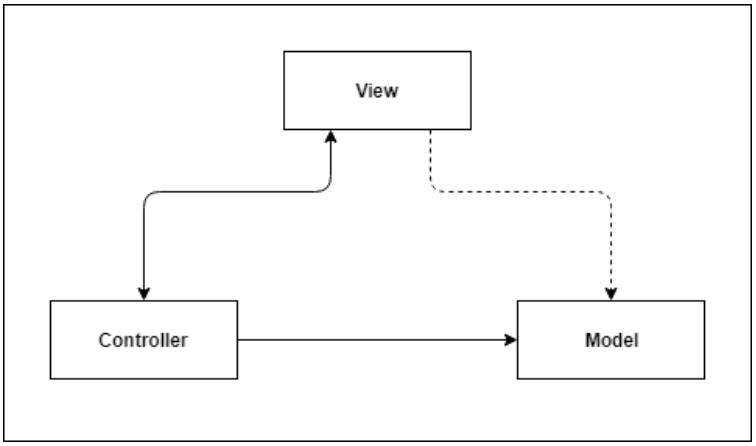


Figure 11. Model-View-Controller (MVC) pattern – modified from [9, p. 23].

Figure 11 reveals that in the pattern, a controller can support many views, the model does not depend on other component and the view contains dependencies on the model. In addition, the controller communicates directly with the model and the view. [9, p. 23.]

2.6.5 Inversion of Control (Enterprise Patterns)

For system that consists of many components which must be tested separately, Inversion of Control (IoC) or Dependency Injection (DI) pattern is essential. The pattern implements the “separation of duties” concept, in which each component contains well-defined tasks to fulfill. [9, p. 23.]

In the pattern, if a class (consumer) uses service of another class (provider), the consumer should not instantiate an instance of the provider, instead the class should use an abstraction (usually interface) from the provider. The interface’s reference will be kept in the consumer so that it can use any class implementation of the interface. [9 pp. 23-24.]

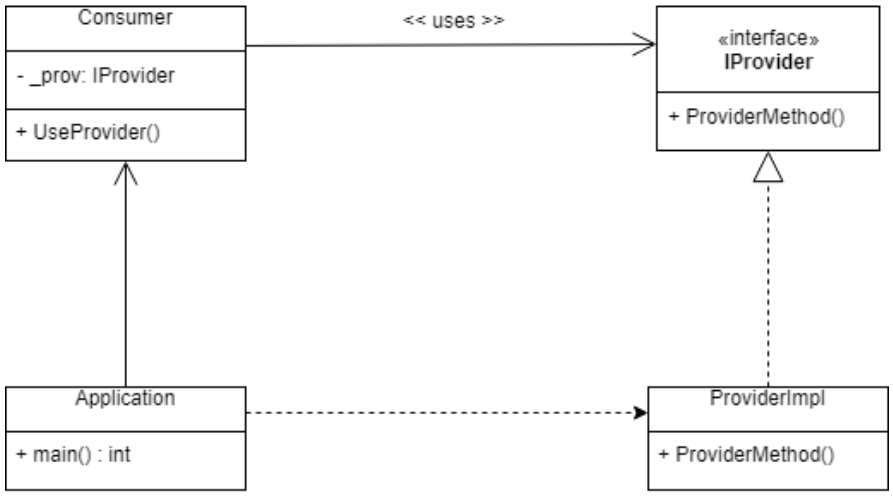


Figure 12. Façade pattern structure – modified from [9, p. 24].

Dependent objects are instantiated externally, to be fed to the consumer who does not control the creation process of those objects. The external provider can be passed into consumer; that implementation is often referred as “dependency injection” which consists of three methods to achieve: constructor injection, method injection and setting injection. [9, p. 24.]

2.7 Human-computer-interaction (HCI) and user-centric design (UCD)

Human computer interaction (HCI) started “in the early 1980s” as a research in computer science’s special field including human factors engineering and cognitive science. Until late 1970s, interaction with computers is limited to information technology professionals and dedicated hobbyist. The period is also a milestone of humanity when personal computing with personal computer platform (operating systems, programming language, hardware) and personal software (applications such as spreadsheets, text editors for productivity) made computer a desirable product for every person on the world and emphasize the needs of development of “usability” for people who want to utilize computers’ resources more efficiently. [13.]

In the end of 1970s, personal computer poses challenge as well as opportunities for the first appearances of various projects in cognitive science including “cognitive psychology, cognitive anthropology, the philosophy of mind, artificial intelligence, and linguistics”. A part of cognitive science’s programme is “cognitive engineering” which aims to build systematic and scientific applications though synthetisation of science and engineer. One of the first example of cognitive engineering is HCI. In parallel to HCI development, human factors engineering, and documentation development are two notable fields in engineer and design area next to HCI. Human factors engineering aims to evaluate human-system interaction in aviation and manufacturing, in which human operators often possess higher problem-solving capability. On the other side, Documentation had mission of producing technical description with a cognitive approach by combining theories of reading, writing, media and practical user testing. In document development, the “usability” of documentation is strongly emphasized. [13.]

Due to the unmanageable complexity of software in the 1970s (the “software crisis”), software engineering must concentrate on non-functional requirements comprising of maintainability and usability, beside functional requirements. Therefore, iterative prototyping and practical testing are relied heavily in software development processes. Computer graphics and information retrieval (started in 1970s) are two fields that see the needs of interactive systems to go further than the past’s achievements. All attempts in early development of computer science reveals the necessity of computer to better understand and empower users. [13.]

HCI consists of the technical focus on “usability”, which can be naively understood as “easy to learn, easy to use”. Yet, the usability concept inside HCI is continuously

becoming richer, now often containing of fun, well-being, aesthetic tension, flow, human development support, etc. Nowadays, HCI has expanded larger than computer science, with less focus on “individual or generic user behaviour” but more on social, original computing, accessibility for the elders and impairers. HCI grown to include “games, learning and educations, commerce, health and medical applications, emergency planning and response, system to support collaboration and community”. Furthermore, HCI includes both the “early graphical user interfaces to interaction techniques, handheld and context-aware interactions”, etc. [13.]

Figure 13. Disciplinary knowledge involved in design of HCI – reprinted from [13].

Figure 13 illustrates “user experience/usability” is the contemporary cross intersection of all disciplinaries and HCI. HCI academic programs train many professional types: “user experience designers, interaction designers, user interface designers, application designers, usability engineers, user interface developers, application developers, technical communicators/online information designers”, etc. It is fascinating to see HCI moved “beyond the desktop”, in which “interaction-design.org” specifies three distinct senses or boundaries. Firstly, by realizing the desktop metaphor is limited than it was, meaning that the “messy desktop” for direct interaction between users and data objects/folders should be substituted or enhanced by searching feature. Secondly, the Internet changed our society and interaction methods between people. People can use tools and application to collaborate through instant message, wikis, blogs, online forums, social networking, media spaces and other workspaces for collaboration. New “paradigms and mechanism for collective activity” appeared such as “online auctions, reputation systems, crowd sourcing” etc. Facebook, Twitter, Linkedin, Linux and Github communities are few examples of daily computing experiences for many people. Thirdly, with the explosion of

appearance of new computing devices, started by laptops in the early 1980s, and handhelds in the mid-1980s; computing now appear in almost all human daily activities through interaction with cars, home appliances, furniture, clothing, remote conference call devices, smartphones, etc. Understanding of HCI from science and engineer viewpoints provide a solid foundation for further study of building useful applications (with great usability) to harness the “pervasive incorporation of computing into human habitats” and enhance human activity and experience. [13.]

The “software crisis” happened in 1970s was not the only time that humanity faces difficulty of building software; some notable recent software failures and costs can be seen from article names “Why Software Fails” on IEEE-Spectrum by Robert N. Charette.

YEAR	COMPANY	OUTCOME (COSTS IN US $)
2005	Hudson Bay Co. [Canada]	Problems with inventory system contribute to $33.3 million* loss.
2004–05	UK Inland Revenue	Software errors contribute to $3.45 billion* tax-credit overpayment.
2004	Avis Europe PLC [UK]	Enterprise resource planning (ERP) system canceled after $54.5 million† is spent.
2004	Ford Motor Co.	Purchasing system abandoned after deployment costing approximately $400 million.
2004	J Sainsbury PLC [UK]	Supply-chain management system abandoned after deployment costing $527 million.†
2004	Hewlett-Packard Co.	Problems with ERP system contribute to $160 million loss.
2003–04	AT&T Wireless	Customer relations management (CRM) upgrade problems lead to revenue loss of $100 million.
2002	McDonald's Corp.	The Innovate information-purchasing system canceled after $170 million is spent.
2002	Sydney Water Corp. [Australia]	Billing system canceled after $33.2 million† is spent.
2002	CIGNA Corp.	Problems with CRM system contribute to $445 million loss.
2001	Nike Inc.	Problems with supply-chain management system contribute to $100 million loss.
2001	Kmart Corp.	Supply-chain management system canceled after $130 million is spent.
2000	Washington, D.C.	City payroll system abandoned after deployment costing $25 million.

Figure 14. Recent software project failures and estimated costs – modified from [14].

It was estimated that one trillion USD will be spent in software projects in 2005 worldwide; however, 5 to 15 percent of those projects will be abandoned right after delivery and many others will be competed late with over budget. Those failures are due to twelve reasons following the Robert N. Charette’s article. In a talk in 2011 for “humanfactors.com”, Dr. Susan Weinschenk mentioned three out of the twelve reasons of failures are related directly to user experience or User-centered design (UCD) work including: “badly defined system requirements; poor communication among customers, developers and users; stakeholder politics”. Furthermore, she concluded the three issues can be resolved by a user-experience professional’s typical work such as stakeholder interviews, user research, testing, user-center design. [15.]

UCD is a design process which is carried out in different iterations, in each iteration, designers focus on user needs and involve users in the design process by “research and design techniques” to build accessible and usable products for the user. Designers need to employ investigative and generative methods and tools to deepen understanding about users. In UCD, designers possess goals of building the whole user experience for a product, hence the design team must comprise of multiple disciplinary professionals such as psychologist, software and hardware engineer, domain experts, stakeholders and users. [16.]

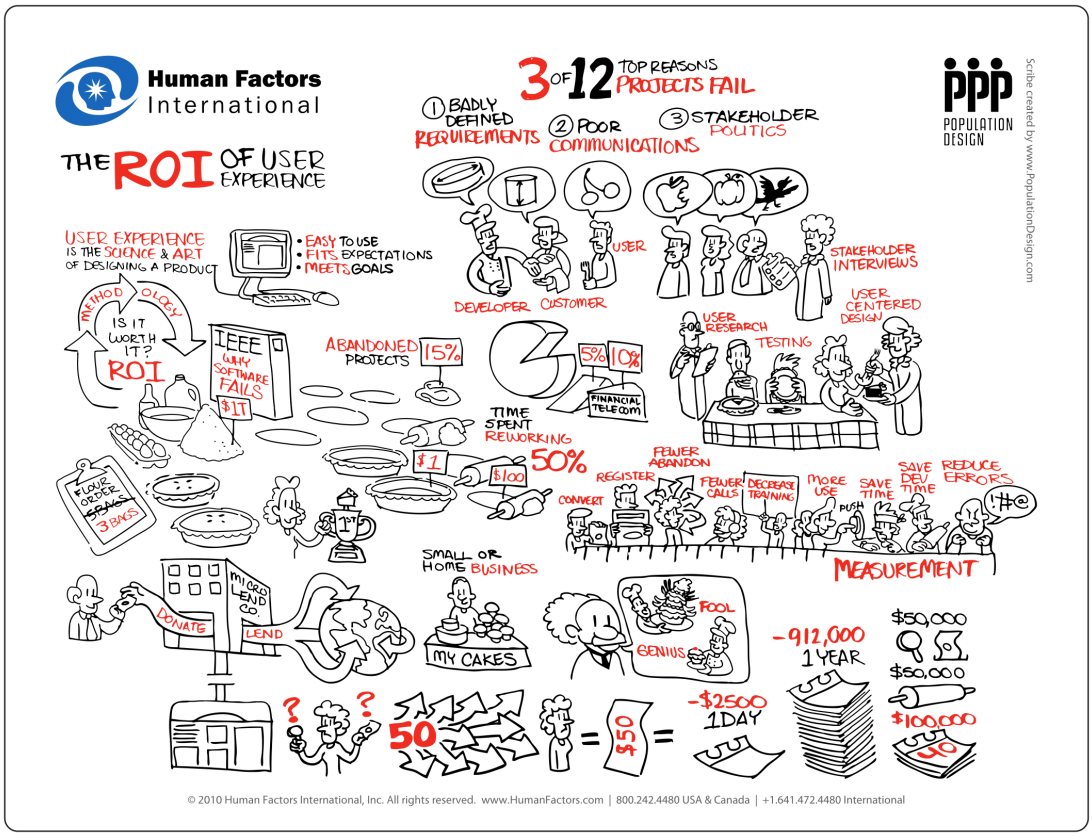


Figure 15. Summary of “The ROI of User Experience” talk – reprinted from [17].

Figure 15 demonstrates summary of the “ROI of User Experience” talk by Dr. Susan Weinschenk (2011), which many points of views resonate with the author theoretical understanding and practical work experience. It is essential for a programmer/developer to possess sufficient understanding about UCD (which is often conducted by User Experience Designer), to make sure that every line of written code must contain meaningful purpose(s) and ideally results in better “usability” of an application.

2.8 REST architecture

Representation State Transfer (REST) is an architectural style which is becoming more popular in many recent web development projects, therefore understanding of REST and its constraints is essential for any web developer. REST is used for distributed hypermedia system with constrains to retain a set of software engineering principles. In Roy Fielding's dissertation in 2000, he mentions REST as an architectural style applying "to the design of Hypertext Transfer Protocol (HTTP) and Uniform Resource Identifier (URI)". The REST architecture aims to reduce interaction latency between general interfaces and intermediary components, therefore system which was built with REST will be able to scale up to Internet size. [18.]

Some important characteristics of REST for HTTP and URIs by Sundvall et al. [18] are:

- URIs as forms of resource identifiers identify resources which are targets of references.
- Various representations formats for resources such as "HTML, XML, JSON (JavaScript Object Notation), serialized Java objects and plain text" were built to be included in the body of the messages. The HTTP header includes representation metadata such as media (MIME) type in the HTTP protocol.
- To act on resources, specified methods (HTTP verbs) are used in request from client. Methods such as OPTIONS and GET should not create the server side's changes, meaning that results of the methods' results can be cached to serve for future requests. Representation of resources is fetched by GET method on the URI identifies of the resource (which is accomplished by requesting an URL on a browser). To update content or create if the content is missing at a target URI, PUT method should be used. DELETE method is used to delete resources. In addition, POST method is used for adding resources or modify existing ones.
- Headers, a HTTP status code, and optionally a body are included in a response. Furthermore, status codes consist of different meanings such as success: "200 OK", "2001 Created" etc.; redirection: "301 Moved Permanently", "304 Not Modified"; request errors: "404 Not Found", etc.; server errors: "500 Internal Server Error". "Location" header field with target URL should be included in new resources responses' creation or redirection.

Uniform Resource Identifier defines a physical or abstract resource. URI contains multiple components, those can be illustrated in figure bellow.

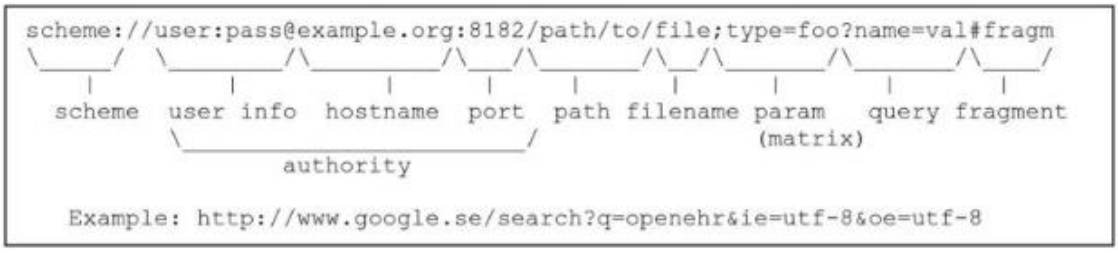


Figure 16. URI Components– reprinted from [18].

Figure 16 reveals how an URI is constructed with the first three components: scheme or protocol (http, ftp, mailto, urn), optional user info part and connecting server and port. The other components of the URI help to point to the correct file in a specific part or help to build up an identifier that can be sent to the server, being processed differently. [18.]

If the fragment part (using only for the client - after "#") is changed, it is not necessary to send new request to the server. Client-side code can use the fragment to redirect user to correct segment of a webpage; the fragment also helps in bookmarking and enhancing back button a browser as the button can help to redirect user even when the history is generated by the client-side, not by fetching from the server. [18.]

The REST architecture is not ideal in all use cases, for example when a server must frequently initiate interaction or sending notifications. REST can achieve better scalability level if it is enhanced correctly with other design patterns and solutions. Sockets can help for reducing overhead of unnecessary features in POST and GET requests if those requests are small and frequent between components or between client and server. WebSockets can take advantage of the upgrade mechanism of HTTP by opening over a single TCP socket a bidirectional communication channel. In addition, reducing requests' number and caching and can help further to solve scalability issues. [18.]

As Fielding stated in section 2.3.1.3 of his dissertation about Network Efficiency: "An interesting observation about network-based applications is that the best application performance is obtained by not using the network", application designers can try to reduce total necessary calls by making less (larger) requests instead of many small ones. Caching or temporary storing of fetched files can help to reduce request number to a server if the server is able to understand state of the fetched data. The HTTP/1.1 protocol header provides helpful information that can be used to understand information related to cache between client and server. Some examples are: "expires" header, "cache-control" header ("max-age"); "ETag", "last-modified"; "If-None-Match", "If-Match". The first and the

second set of headers indicate when new response need to be re-sent to the client-side. “If-None-Match” helps server to determine whether to serve a page from GET request or not; if the previous cached response matches, HTTP status code “304 not modified” will be sent by the server. Similarly, if a client sends a PUT request, “If-Match” header can be examined to see the necessity of updating a resource which might be updated by someone else. [18.]

Many more advanced techniques such as vertical scale up, horizontal scale out (Sharing, MapReduce) that will not be covered in this section, as the reviewed architecture style and techniques are more suitable for a start-up’s immediate concerns.

2.9 Introducing several selected software development frameworks with code implementation

Software frameworks often are built following industry standard with numerous necessary design patterns. Model-View-Controller (MVC) is a prevalent pattern which applies in many web development frameworks. In this thesis section, few web development frameworks will be mentioned and introduced briefly. For brevity, this section focuses on introducing few back-end frameworks together with their controllers’ core features.

Server-side web frameworks are software that helps to increase productivity in writing, maintaining and scaling the back-end parts of web applications. Common web development tasks will be simplified by libraries and tools, those tasks include routing URLs to correct handlers, performing Create, Read, Update, Delete (CRUD) operations on databases, session management, authorization, security enhancement and output handling (JSON, XML, HTML). [19.]

A developer can build web applications without frameworks; however, that incurs the unnecessary tasks of manually building and testing of some commonly-available software components. In contrast, using web frameworks, the developer can use the existing well-tested, low-level software components to write simpler syntax and deal with easier, high-level code. [19.]

```
1  const express = require('express');
2  const router = express.Router();
3  const { data } = require('../data/flashcardData.json');
4  const { cards } = data;
5
6  router.get( '/', ( req, res ) => {
7    const numberOfCards = cards.length;
8    const flashcardId = Math.floor( Math.random() * numberOfCards );
9    res.redirect( `/cards/${flashcardId}` )
10 });
11
12 router.get('/:id', (req, res) => {
13     const { side } = req.query;
14     const { id } = req.params;
15
16     if ( !side ) {
17         return res.redirect(`/cards/${id}?side=question`);
18     }
19     const name = req.cookies.username;
20     const text = cards[id][side];
21     const { hint } = cards[id];
22
23     const templateData = { id, text, name, side };
24
25     if ( side === 'question' ) {
26       templateData.hint = hint;
27       templateData.sideToShow = 'answer';
28       templateData.sideToShowDisplay = 'Answer';
29     } else if ( side === 'answer' ) {
30       templateData.sideToShow = 'question';
31       templateData.sideToShowDisplay = 'Question';
32     }
33
34     res.render('card', templateData);
35 });
36
37 module.exports = router;
```

Figure 17. Express framework code sample.

Figure 17 illustrates routing setup of a small flashcard game application which was written on Express framework. Express.js (or Express) is a web application framework for Node.js which was released in 2010 with designed architecture for simplifying the process of building APIs and web applications [20]. From the figure, the first line calls Express library, following by the second line which assigns all Router library of Express to the “router” constant. Line 6 illustrates the handling of user request to the root route of the application with GET method (which is typically carried out by a user accessing to an URL on a browser). In the call-back function, request and response objects are available and response (res) handles redirection to another route of the application. Similarly, in line 12, another router was built to handle GET method to the root URL with additional parameter “:id”. The Express framework helps to catch easily request parameter such as the “id” constant in line 14 for identifying which card a user is requesting. If no side can be extracted from the request parameter, a default option will be provided, meaning

the user will be redirect to a webpage containing a question view for a card (line 17). Line 23 reveals construction of a “templateData” object which will be provided when a “card” view is rendered in line 34. Finally, the “router” object will be exported to be available for using in the whole application with the statement in line 37. In this example, the route file, naming “cards.js” acts as a controller to handle request (routing) to application, serve relevant data to different views.

Spring is another web framework that is essential to mention in this section. Spring is the most popular framework for building enterprise application in Java. That is an open source Java platform which was released first time in 2003, initially by Rod Johnson. Spring framework can be used to build any Java application; however, it is famous as a framework for building web applications on JavaEE platform. Spring framework contains numerous benefits in using such as enabling development of applications using POJO (Plain Old Java Object) in enterprise-class level. In addition, Spring is modular; even with substantial number of built-in classes, developers only need to concern with classes that they need. Moreover, Spring embraces testing as code that depends on environment is available in the framework and POJO empowers dependency injection for injecting test data. Spring handles well exceptions from other connected technologies such as Java Database Connectivity (JDBC), Hibernate or Java Data Objects (JDO). Many other benefits of Spring framework for building Java web application make the framework becomes the top selection for various Java projects. [21.]

```
1  import org.springframework.beans.factory.annotation.Autowired;
2  import org.springframework.stereotype.Controller;
3  import org.springframework.ui.ModelMap;
4  import org.springframework.web.bind.annotation.PathVariable;
5  import org.springframework.web.bind.annotation.RequestMapping;
6
7  import java.util.List;
8
9  @Controller
10 public class CategoryController {
11     @Autowired
12     private CategoryRepository categoryRepository;
13
14     @Autowired
15     private GifRepository gifRepository;
16
17     @RequestMapping("/categories")
18     public String listCategories(ModelMap modelMap) {
19         List<Category> categories = categoryRepository.getAllCategories();
20         modelMap.put("categories", categories);
21         return "categories";
22     }
23
24     @RequestMapping("/category/{id}")
25     public String category(@PathVariable int id, ModelMap modelMap) {
26         Category category = categoryRepository.findById(id);
27         modelMap.put("category",category);
28
29         List<Gif> gifs = gifRepository.findByCategoryId(id);
30         modelMap.put("gifs",gifs);
31
32         return "category";
33     }
34 }
```

Figure 18. Spring framework code sample.

Figure 18 illustrates a shortened version of a controller in Spring framework. The first line imports “Autorwire” annotation feature to the controller, enabling injection of two objects “categoryRepository” and “gifRepository”. The classes “CategoryRepository” and “GifRepository” are annotated with “@Component” annotation so that the two object instances can expose necessary methods inside the “CategoryController” class. Line 17 illustrates an example of handling GET request to “/categories” route, which return a view name “categories” with data available from the “categories” object fetching from line 20. “ModelMap” is a built-in interface which helps to make data available to any view in the String framework. Line 24 demonstrates an example of how to handle a route when a user clicks on an item in the “categories” list and tries to see detail of an individual category. Spring helps to extract URL parameter “id” with “PathVariable” annotation, hence in line 26, a single category can be obtained. In line 29, all “gif” items in a category can be queried and assigned to the “gifs” variable to be used in the view “category”.

CodeIgniter is a relatively modern MVC development framework that will be reviewed in this section. Currently, CodeIgniter (CI) 3 is the application production version, CI2 is the legacy version and CI4 is coming. CodeIgniter is an Application Development framework building on top of PHP programming language. PHP is a back-end programming language which owns long history of existence. The programming language was created in 1994 by Rasmus Ledof who used it to track visitors on his online resume page. Originally, the program is named “Personal Homepage Tools” and was written in C programming language to create a set of Common Gateway Interface (CGI) binaries. Through time, PHP language has evolved to the current version of PHP 7. [22.]

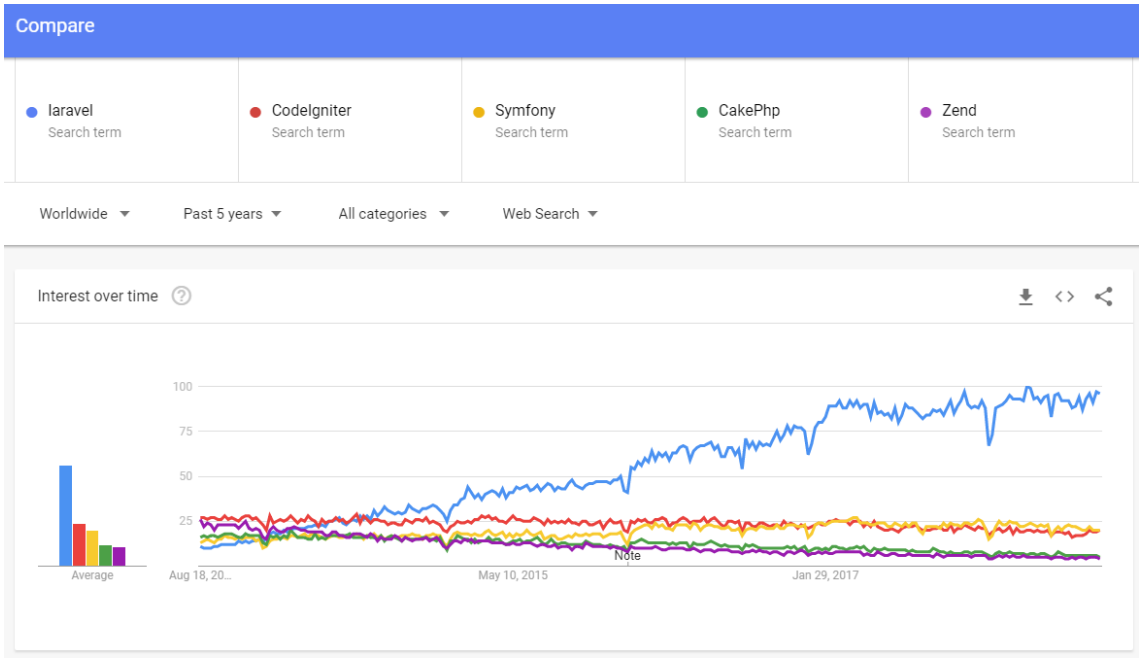


Figure 19. Google Trend comparison of popular PHP frameworks [23].

Many of features are available out-of-the-box from CodeInigniter3 including: Model-View-Controller Based System, Light Weight, Full Feature database classes, Query Builder Database Support, Form and Data Validation, Security and XSS filtering, Session Management, Email Sending Class (Support Attachments, HTML/Text email, multiple protocol with SMTP, Mail etc.), Image Manipulation Library (cropping, resizing, rotating etc.), File Uploading Class and many more. Those listed libraries build up a solid foundation of functionalities for any commercial software web application (in PHP). More than dozens of frameworks for PHP currently exist; however, CodeIgniter is arguably one of the top choice, although not as popular as the rising Laravel framework. [24.]

In the following section, two excepts of sample PHP code written in PHP will be analyzed to demonstrate the simplicity of the CodeIgniter framework.

```
1  <?php
2     class Stud_controller extends CI_Controller {
3
4        function __construct() {
5           parent::__construct();
6           $this->load->helper('url');
7           $this->load->database();
8        }
9
10       public function index() {
11          $query = $this->db->get("stud");
12          $data['records'] = $query->result();
13
14          $this->load->helper('url');
15          $this->load->view('Stud_view',$data);
16       }
17
18       public function add_student_view() {
19          $this->load->helper('form');
20          $this->load->view('Stud_add');
21       }
22
23       public function add_student() {
24          $this->load->model('Stud_Model');
25
26          $data = array(
27             'roll_no' => $this->input->post('roll_no'),
28             'name' => $this->input->post('name')
29          );
30
31          $this->Stud_Model->insert($data);
32
33          $query = $this->db->get("stud");
34          $data['records'] = $query->result();
35          $this->load->view('Stud_view',$data);
36       }
```

Figure 20. An example controller performing CRUD written in CI framework – part 1 [25].

Figure 20 reveals an example of a controller written in CodeIgniter, the controller here consists of several responsibilities: displaying views and perform CRUD operations. Line 2 illustrates inheritance of “Stud_controller” from “CI_Controller” which is a base class in CodeIgniter framework which contains many helper classes. The constructor of the class guarantees that whenever the class is use, the “url” helper and “database” libraries are loaded in line 6,7. The “public” accessor ensures that the function can be call directly from a browser URL, for example “index” function can be call by request the URL: baseURL + “/” + “Stud” (base URL is the root URL of a web application such as http://localhost and “+” is an operator for string concatenation). The “index” function obtains data to an associative array “data[‘records’]”. Line 15 illustrates the simplicity of serving a view “Stud_view” with “data” fetched from the database; on the view data will be available in

the PHP object “records” and freely to be used to construct the dynamic part of the “Stud_view”. This “index” function owns a purpose of serving default data about students when a user is redirected to the route “/Stud_controller”.

In addition, the controller consists of other responsibilities such as serving views and performing CRUD operations. Function “add_student_view” demonstrates how to serve the view with name “Stud_add” to user. Following CI convention, the view file is in a separated folder (often named “views”). Function “add_student” starts on line 23 acts as a callable function for adding student to the database; the action logically will come from the “Stud_add” view but can come from any other view as well. The function loads another Model with name “Stud_model” on line 24; this model is another PHP class (located in “models” folder) which contains a single responsibility in the CI framework - perform interaction with the database (CRUD operations). The function expects form input coming from POST method from two input fields with name “roll_no” and “name” coming from a view. On line 31, the helper database method “insert” performs inserting the request data into the database. Before inserting, no validation or sanitization of input data “$data” was considered as the code segment illustrates a simplified example. After inserting data, student data needs to be re-fetch to be available to the front-end part, performing by line 33 and 34. Finally, line 35 serves the view “Stud_view” with all students’ information including the just added student information to a user.

```
37
38          public function update_student_view() {
39              $this->load->helper('form');
40              $roll_no = $this->uri->segment('3');
41              $query = $this->db->get_where("stud",array("roll_no"=>$roll_no));
42              $data['records'] = $query->result();
43              $data['old_roll_no'] = $roll_no;
44              $this->load->view('Stud_edit',$data);
45          }
46
47          public function update_student(){
48              $this->load->model('Stud_Model');
49
50              $data = array(
51                 'roll_no' => $this->input->post('roll_no'),
52                 'name' => $this->input->post('name')
53              );
54
55              $old_roll_no = $this->input->post('old_roll_no');
56              $this->Stud_Model->update($data,$old_roll_no);
57
58              $query = $this->db->get("stud");
59              $data['records'] = $query->result();
60              $this->load->view('Stud_view',$data);
61          }
62
63          public function delete_student() {
64              $this->load->model('Stud_Model');
65              $roll_no = $this->uri->segment('3');
66              $this->Stud_Model->delete($roll_no);
67
68              $query = $this->db->get("stud");
69              $data['records'] = $query->result();
70              $this->load->view('Stud_view',$data);
71          }
72      }
73  ?>
```

Figure 21. A sample controller performing CRUD written in CI framework – part 2 [25].

With CI framework, it is also simple to perform update and delete data. Function “update_student_view” performs serving a view for updating a student information by expecting a student number obtaining from the URL in segment 3rd in line 3 (eg. with request to the URL: baseURL + “/” + “Stud” + “/” + update_student + “/” + 3, the “roll no” variable will hold value of 3). Line 41 and 42 query a student data and line 44 serves the view “Stud_edit” with the queried data. Starting on line 47, function “update_student” performs updating data of a student. To interact with database, model “Stud_Model” is loaded in line 48. Similarly, to the “add_student” function, this function expects two input fields sending through POST method, containing roll number and name to be updated. The “Stud_Model” model will perform update those new data on a row which is found by querying in the database with “old_roll_no” number. When the action is successfully conducted, “Stud_view” with all refreshed students’ data will be served on line 60. The function “delete_student” will help to delete a student by “roll_no” number, and the operation sequence is comparable to other functions as demonstrated above. It is noticeable here

that the complexity of “Stud_Model” model is abstracted away from the “Stud_controller” controller, a call to “delete” function on line 66 expects the model does it work which can be carried out by a CI database helper method or a regular SQL query.

The pattern here is clear, a function to serve a view for changing data and a function does the actual work in a controller of CI framework. Certainly, developers own the flexibility to write code differently, but fundamentally interaction with database is simplified when coding is conducted in an expected pattern which is available out-of-the-box from CI framework (documentation).

Many other notable software frameworks that are not introduced here such as Django/Flask for Python and ASP.Net for C#; however, the fundamental MVC design patterns are similar in all mentioned frameworks in this section from the author point of view, hence code samples are not introduced.

2.10 Functional programming (JavaScript)

Functional programming is a software development style that has become more popular recently. This section aims to concisely review functional programming concepts, why and how to use those concepts in JavaScript – an essential programming language in most modern web development projects.

The users’ demand compounding with rapid development of web platforms and browsers changes the way of web application development today. Frameworks can help to build more robust, scalable applications; however, the complexity of the code base can easily go out of control. Object oriented programming can partially solve the problem, although programmers need more. The popular questions that application designers must answer [26, p. 4] include:

- Resource Extensibility: does developer need to constantly refactor code to add new functionalities?
- Easy to modularize: does changes in one file affect other files?
- Reusability: are there unnecessary duplications?
- Testability: is that easy to write unit test for a function?
- Easy to reason about: is that the application well-structured and written code easy to follow?

Functional programming today is supported by built-in API functional supports in many major programming languages: Scala, Java8, Python, JavaScript etc. In JavaScript, functional programming helps programmers to be more productive by enabling clean, modular, testable and succinct code. In functional programming, four fundamental concepts need to be maintained, including: declarative, pure functions, referential transparency, immutability. [26, p. 5.]

Declarative programming aims to construct a set of operations without revealing how those were implemented or how data flow through them, in contrast to imperative or procedural programming that exist in most structured (C etc.) and object-oriented languages (C++, C#, Java etc.). For example, to square all numbers in an array, declarative programming would be carried out differently.

```
1  var my_arr = [0,1,2,3,4,5,6,7,8,9,10];
2  for(let i = 0; i < my_arr.length; i++){
3      my_arr[i] = Math.pow(array[i], 2);
4  }
5
6  my_arr; //-> [0,1,4,9,16,25,36,49,64,81]
```

Figure 22. Procedure/imperative looping in JavaScript – modified from [26, p. 8].

Figure 22 illustrates imperative style of programming by telling how to perform a task for computer. In contrast, declarative program concerns on what logic to achieve without specifying a program's state changes or control flow. The Array.map() function helps programmer to focus on implementing the correct behaviour on each element of an array.

```
[0,1,2,3,4,5,6,7,8,9,10].map({
    function(){
        return Math.pow(num, 2);
    }
});

//-> [0,1,4,9,16,25,36,49,64,81]
```

Figure 23. Declarative looping in JavaScript – modified from [26, p. 8].

Figure 23 demonstrates that programming can be freed from managing a loop, which is difficult to reuse and bring overhead in maintain more written code (meaning more bugs). In addition, high-order functions such as map, reduce, filter simplify loop control in functional programming. Loop control abstraction also bring benefit of using lambda expression or arrow functions (EcmaScript 6), which is a succinct alternative way of wring anonymous function passing in as a function argument. Functional program also aims for statelessness and immutability, which can be achieved by using pure functions. A pure function only depends on provided input and not on any hidden or external state that changing during its evaluation or between function calls. In addition, a pure function does not cause changes outside of its scope such as changes to a global object or pass in parameters. It is proven that many benefits arise if return result of a function is consistent and predictable, which make the “referential transparency” property of functional programming. [26 pp. 8-9.]

```
var counter = 0;

function increment_imp(){
    return counter++;
}

function increment_dec(input_var){
    return input_var++;
}
```

Figure 24. Referential transparency examples – modified from [26, p. 9].

In figure 24, the result of “increment_imp” is difficult to predicted as it depends on how many time the function was called; yet the function “increment_dec” always return a predictable result base on a pass-in parameter value. Passing parameters with scalar values and defined those beforehand would help to avoid side effect of a function, however when “objects are passed by reference”, programmers must be careful not to accidentally change them. The passing parameters to a function should be immutable. [26 pp. 14-15.]

Immutability is an important concept in functional programming, meaning that data cannot be change after it was created. In JavaScript (as well as other programming languages) primitive types are immutable, yet other objects such as array are not. One example is the Array.sort function in JavaScript causing sort operation in the original array, which is often an undesirable side effect to programmers. JavaScript developers often encounter difficulty in maintaining large functions that rely on external shared variables, with many branching and unclear structure. Functional programming style is promised to solve the difficulty to overcome modern software complexity. This section serves as an introduction to functional programming as the subject is complex and consists of great depth to be explored further. [26, p. 15.]

2.11 Software development in a start-up environment

Startups are new companies with small history of record, high uncertainty and rapid evolution. Following recent studies, startups demonstrate an important position in a country economy. In 2014 in the US, 476000 new businesses are created each month, contributed to 20% of new job creation [27]. In 2016 in Finland, 383 million Euros are invested in 400 new start-ups, a new record in Finnish history [28]. This section of the thesis concentrates to explore software engineering practices in start-ups' environments.

Doing software engineering in a startup means that "done is better than perfect" and people "move fast and break things". A startup contains the flexibility to accept frequent changes, therefore the most viable process is agile methodology. The time from new ideas to production must be short requiring fast releases with iterative and incremental approach. Lean methodology is a variation agile development, which targets to pinpoints the riskiest part of an application and implements a minimum viable product (MVP) for testing and moving on to the next iteration. In that way, any change will introduce less costs to the product and a testable version is available early to see feasibility of continuing development. [27.]

Table 2. Typical characteristics of startups – modified from [27].

Testing type	**Description**
Lack of resources	Economical, human, and physical resources are limited.
Highly reactive	In comparison to more established companies, startups can react quickly to changes in markets, technologies and products.
Uncertainty	Startups deal with highly uncertain environment of market, product features, competition, people and finance.
Rapid evolving	Startups aim to grow and scale fast.
Time pressure	All constraints force startups to release new features fast and frequent. Constant pressure from customers and investors appears in daily work.
Third-party dependency	The lack of resources makes startups rely on external solutions to build products: open source software and API, outsourcing etc.
Small team	Startups start with small number of people.
One product	Only one product/service exists to be built.
Low-experienced team	Team is often formed by people with less than 5 years of experience and fresh graduated students.
New company	The company was created recently.
Full organization	Startups consist of low hierarchical structures with no need for upper management, the team is centered around the founder and everyone owns multiple responsibilities.
Highly risky	Startup contain high failure rates.
Not self-sustained	In the early stage, startups need funding from external sources such as venture capitalist, angle investors, personal/family funds etc.
Limited working experience	The short existing time make organizational culture non-existence.

Table 2 reveals recurrent traits of startups, which many points resonate with the author's personal experience. Startups need to gain new customers fast, by concentrating on serving customers, making them act as "designers". Feedbacks from customers are critical and often used by developers to drive requirements. In addition, the team is the central of development, by empowering each member of the team, manager hopes to counterbalance the lack of resources. Finally, startups contain the advantage of choosing newest technologies without concerning legacy codebase. Startups attempt to generate revenue and find funding fast, hence software quality is not the most critical element. In that environment, developers own freedom to choose what to work on, stop to take measure when anything is wrong and accept that failures are inevitable for faster learning or pivoting when necessary (skip unnecessary complex features with small

impacts). Few common practices that should be mentioned from startups including: using well-known frameworks to quickly change to product following market needs; using of prototyping and experimentations with existing components; using easy-to-implement tools to increase development speed. [27.]

3 Requirement analysis and architecture overview

3.1 Use cases analysis

This section aims to review AuroraX from an architecture point of view regarding its financial application nature. AuroraX is a start-up, meaning the company software product must constantly change and new features need to be implemented fast. For a financial commercial application, a relational database for saving data is essential as it is difficult to achieve the same performance and data integrity in comparison to other non-relational database solutions. A backend language is needed and on the front-end styling effort must be minimized by utilizing font-end frameworks. User experience on the UI can be enhanced by JavaScript with a suitable framework to handle user interaction and input data validation. In addition, the back-end part must contain the robustness, correctness and security of a financial application as well as extensibility and flexibility to interact with external APIs. In addition, the application allocates most of computing work to the backend, as the platform is not built in style of a real-time trading platform (for financial products such as: stocks, commodities, electric etc.). It is better to first understand important use cases to gain a holistic view of the application.

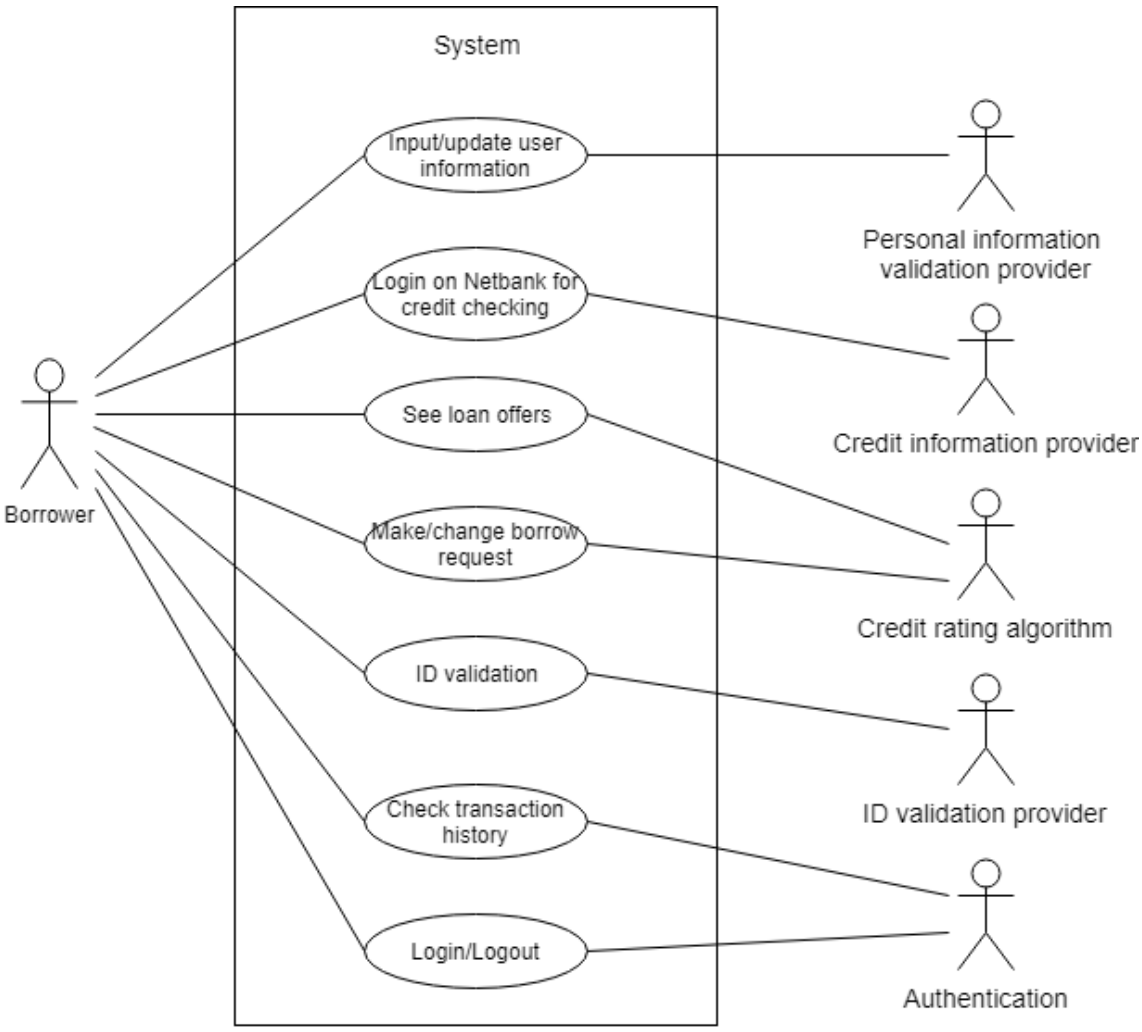


Figure 25. UML use cases of a borrower.

From the constraints of requirements, the use case diagram here reveals core functionalities of the application and how different entities achieve goals with helps from the system. Figure 22 illustrates how different actors in the system to interact with a borrower. The system needs to constantly know about who the borrower is and on which page and which action is the user trying to perform with the help from the authentication system. The borrower needs to provide personal information such as social security number; in Finland, the number would reveal information about a person's age, other personal details and personal payment remarks (with the help of a third-party API, which will impact the person's credit rating). The credit rating provider provides information about recent banking transactions of a person which are used to determine a person's risk rate level. An offer to a person is determined by the credit rating algorithm with different parameters of maximum loan amount, interest and repayment time. The person can modify a loan request to see if other options are available and commit to a loan when feeling satisfied

with an offer. After ID validation, the system will automatically send money to the user account. Invoices must be sent, processed and recorded in the system for borrowers to see. Furthermore, borrowers can make early repayment for any amount that is larger than the amount of the next payment plan item. In addition, the borrower needs to be able to see transaction history and perform login/logout.

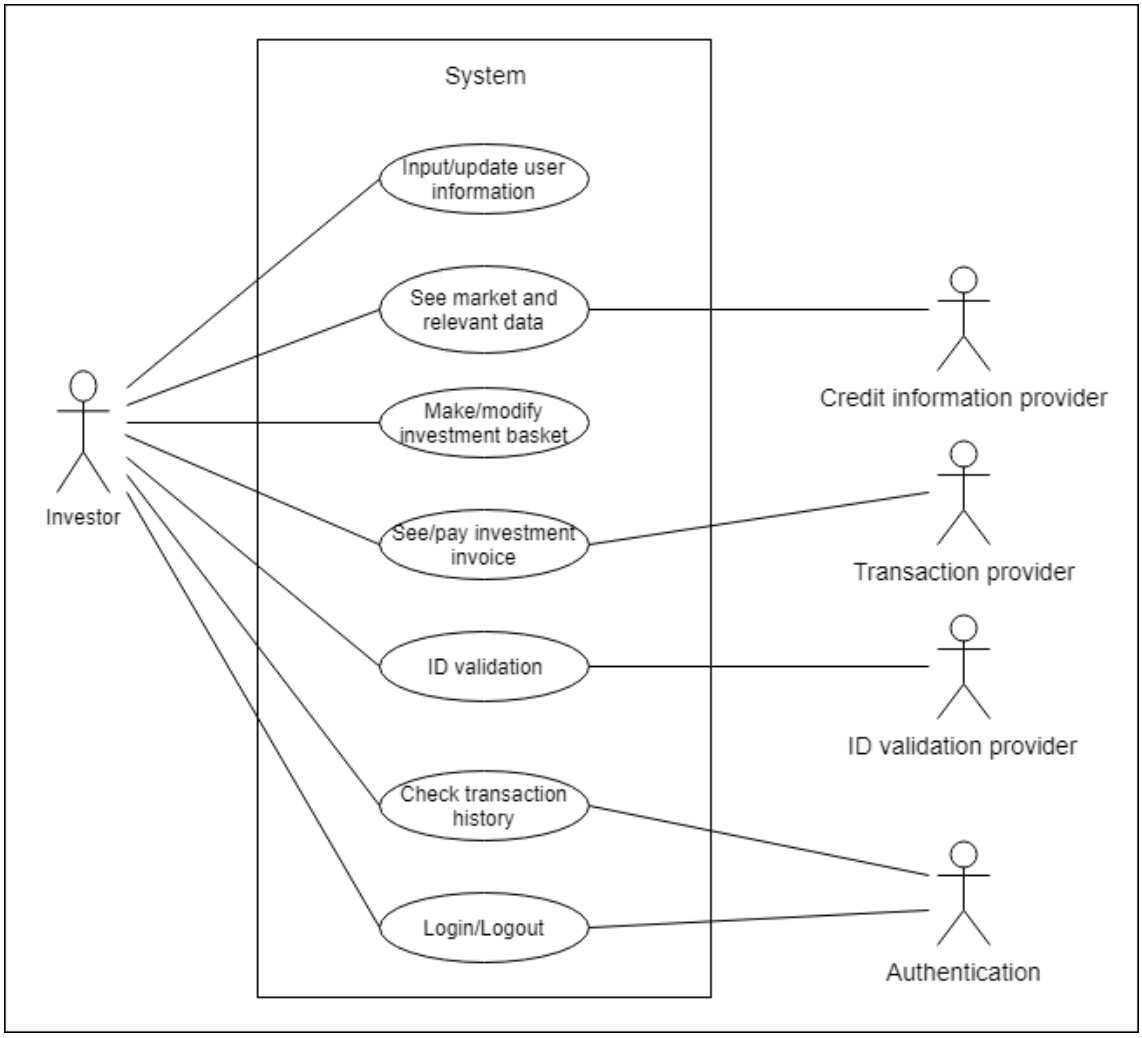


Figure 26. UML use cases of an investor.

Figure 26 demonstrates how an investor can achieve his goals by interacting with the system's exposed functionalities. The system needs to know about additional required information for an investor to guide the user to input more information and the category of the investor: individual or corporate. The investor must be able to input data to for creating an account with (minimum) information requesting from the system including: email, password, living country, social security number, the first name and the last name. The person needs to see market information and statistics such as what are most recent loan requests, what is the open loans request's total volume, how much is the amount of

a loan request and what was the interest rate. The investor needs to see projection of how much s/he will earn in the future and be able to modify to see real-time answer of projected returns. The investor can proceed in the investor flow and system will generate investment invoice for the user to pay; afterwards, system batch will acknowledge the investment and the investment will be displayed on the investor's investment dashboard.

The investor must be able to see investment details of each investment basket and breakdown of each of those details demonstrating how much had been paid, and how much left, and interests (paid/unpaid) for each of the breakdown; a transaction list illustrates information for each transaction must be available in a view for the user.

Another important user in the system is the admin user, who can search borrowers and investors in the systems and perform different actions for customer service operation. In addition, daily batch operations with various functionalities such as processing invoices of investment and repayments, validating ID, cancelling outdated loans, etc. can impact users in the system directly when it runs.

3.2 The borrowing process flow

The borrowing process flow or in short, "the borrower flow" is how the application designer calls the process including all steps for a new borrower to register and complete the borrow process. The flow has been changed many times with trial-and-error method to find the ideal one to optimize the borrower funnel.

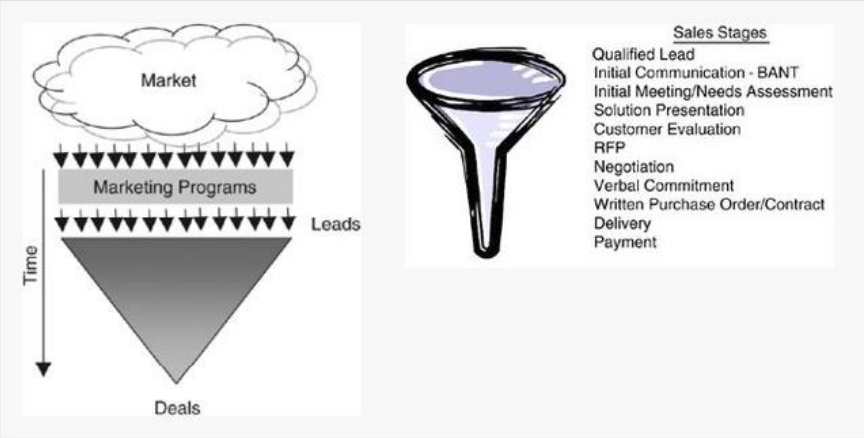


Figure 27. The stages of a sale process - modified from [29].

Figure 27 reveals steps before purchasing of a customer for a product. For an online financial product as AuroraX, the funnel is normally shorter for a customer: a customer can find AuroraX through email marketing or Google Ads, Facebook Ads etc. digital marketing campaign, communication can be short or only one-way through video introduction and FAQs (Frequent Ask Questions) text, and meeting may not be needed, contract can be viewed online and is sent by email, product is delivered as a solution by using the application, and payment is carried out by paying invoice through online bank service.

The cost per customer from a loan comparison site such as “vertaaensin.fi” is currently 60 to 80 euros, and similarly from other affiliate sites in Finland. Using their services is one way of booting sales and increasing brand awareness and brand value, however in the long term, the most sustainable strategy is improving user experience on the site, optimizing the borrower flow, together with using “word-of-mouth” marketing method.

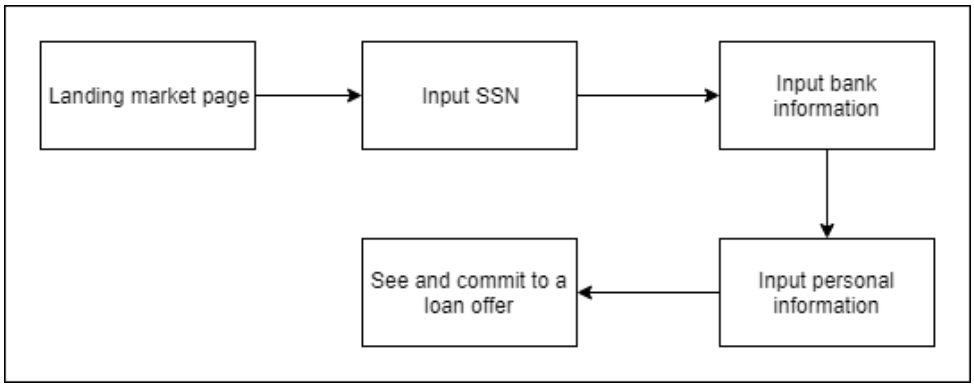


Figure 28. Optimized borrower flow.

Figure 28 illustrates a simplified borrower flow for a borrower, deriving from requirements of the borrower funnel analysis.

3.3 The investing process flow

The investing process flow (the investor flow) includes all steps for a new investor to register and make an investment, and steps for an existing investor to make new investment.

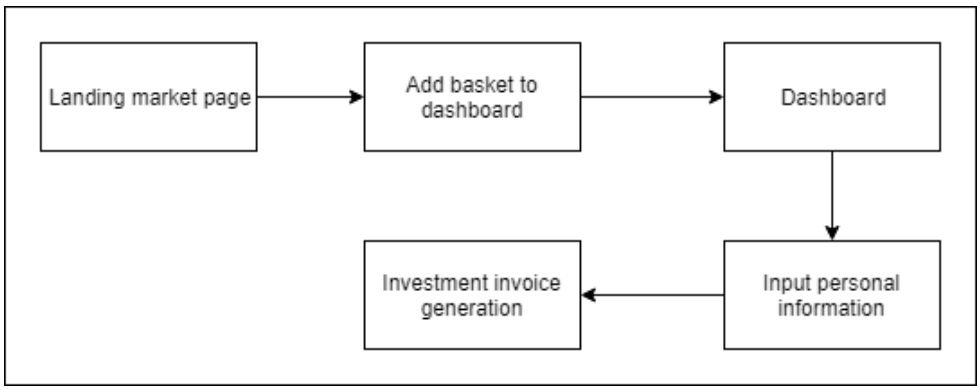


Figure 29. Optimized investor flow.

Figure 29 demonstrates the shortest path possible for a new investor to complete investment action. For an existing investor, inputting personal information step is not needed. In here a web-shop experience is built for investor when s/he must spend minimum effort to preview potential purchases (funding baskets) and only need to provide personal information when a purchase decision is made.

4 Implementation

4.1 Implementation objective

This section aims to review the process of building the new investor dashboard which is one of the main views for investor user in the system with modern styling and smooth user experience. The dashboard stands in the system as one of critical selling point to attract new investors and the center piece to present investment data to existing investor. Therefore, the page contains high complexity and high dependencies for other parts of the investor flow. Building the dashboard takes time and effort to rethink base on the designer's designer what essential elements to keep and how to construct the on the code level in a SPA (Single Page Application) style.

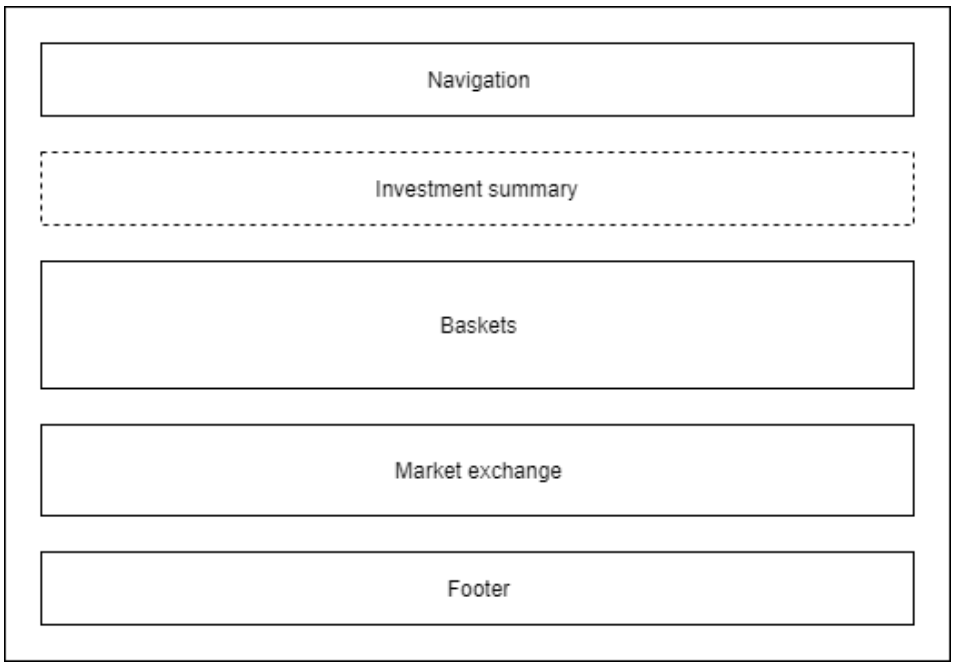


Figure 30. Component structure of an investor dashboard.

Figure 30 reveals the component-based structure when the page was planned further after receiving design materials from a designer. The navigation and footer components can be reused in all other pages of the application and the market exchange component can be reused for the borrower's UI. The core element of the page is the baskets section where each basket with all its functionalities appears. To reuse existing authentication system, a new user with a random-generated email and a default password will be generated whenever a user proceeds in investor path from the landing market page. The user's information will be asked and updated in the system on another page when the user decided to proceed to the invoice generation page. The "baskets" component needs to be built by reusing and improving existing source code. The investment summary component will only reveal when two baskets were created (can be in any state financed or not-financed) or when one financed basket was realized in the baskets section. Other components on the page needs minimum changes to work together with the "baskets" component.

4.2 Implementation details

The concept of scope is important in JavaScript, as scope protects accidentally rewriting variables and difficult-to-track bugs. On the investor dashboard, different JavaScript scope for each non-related component needs to be constructed. Navigation bar and

footer are two separate components that each consists of its own scope. Investor summary, baskets and market exchange contain the same scope, as some data of the basket's component are used for both market exchange and investment components.

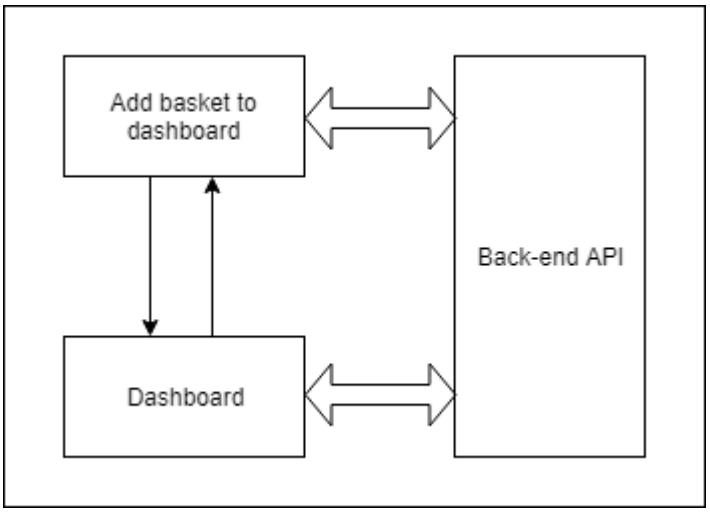


Figure 31. Investment basket creation mechanism.

Figure 31 illustrates a loop between two views (components) in which an investor can adds as many baskets as s/he wants to the investor dashboard for previewing. The "add basket to dashboard" component should be a subcomponent of the "baskets" component in the dashboard to reduce duplicated business logic; however, to lower complexity of the dashboard component, the other component was implemented as a separate component on another view. With this design, an investor will experience a similar UI to the UI when buying products from an e-commerce site (Amazon, Ebay etc.). Each basket consists of two sections: investment details on the left side and projected return on the right side. A basket contains two states: financed (investment invoice was paid) and unfinanced (investment invoice was not paid).

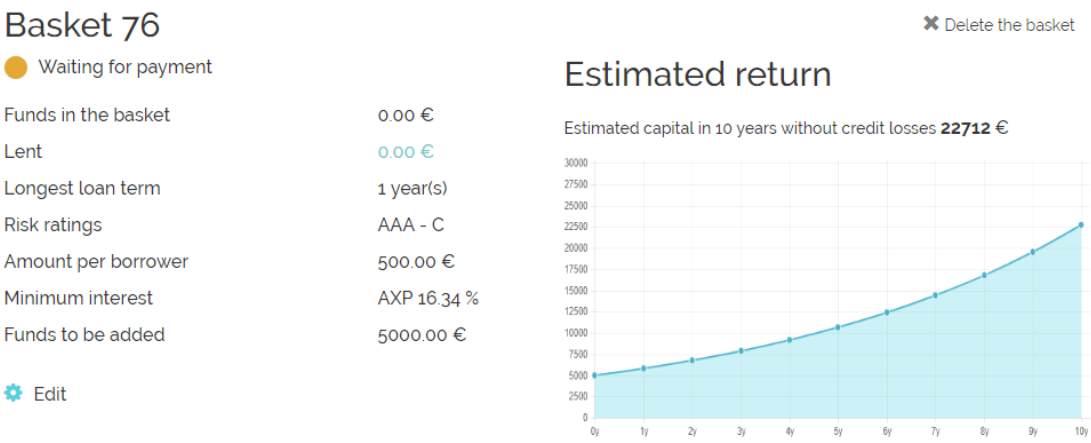


Figure 32. An unfinanced basket.

Figure 26 demonstrates an unfinanced basket for an investor, all necessary information about the basket was gathered from another component “add basket to dashboard”. Base on the amount of investment and yearly interest, the chart on the right side of the basket component will represent corresponding computed data with the compound interest formula.

```
1   <div class="col-md-11 ...">
2       <div data-bind="foreach: moneyinvestedintable">
3           <!-- Bootstrap Modal dialog for allowing the investor to pay the investment invoice. -->
4           <div data-bind="bootstrapShowModal: $data.modalPayInvoice" class="modal fade modal-scroll">
104         <!--Ending-Bootstrap Modal dialog for allowing the investor to pay the investment invoice. -->
105         <!----------------------------------Starting of adding funds section for each basket----------->
106         <!----------------------------------Starting of adding funds section for each basket----------->
107         <div data-bind="visible: $data.basketView()=='add-funds-basket'">
160         <!---------------------------------Starting of moving funds section for each basket----------->
161         <!---------------------------------Starting of moving funds section for each basket----------->
162         <div data-bind="visible: $data.basketView()=='move-funds-basket'">
231         <!---------------------------------Starting of editing section for each basket---------------->
232         <!---------------------------------Starting of editing section for each basket---------------->
233         <div data-bind="visible: $data.basketView()=='modify-basket'">
387         <!----------------------------------Ending of editing section for each basket----------------->
388         <!----------------------------------Ending of editing section for each basket----------------->
389         <div data-bind="visible: $data.basketView()=='basket'" class="row m-row">
491     </div>
492 </div>
```

Figure 33. Structure markup of a basket (code folded).

Figure 33 reveals how a basket component was structured in HTML markup. Line one uses Bootstrap CSS class to style the whole division taking 11 columns in a 12 grid columns system for a responsive layout. The JavaScript (JS) object “moneyinvestedintable” contains an array of all basket objects that belong to an investor. Each basket component owns four components (views) inside and only one component is revealed at a time (the “modalPayInvoice” is a legacy component here). Constructing the basket in this way requires minimum loading/waiting time for each action on the page after all resources were loaded once. On the same position of each basket, different views can be

rendered depending on the defined logic. KnockoutJS framework makes adding business logic to HTML friendlier (than using PHP) with built-in dynamic rendering functionalities for HTML elements on a browser DOM (Document Object Model). In the figure, the “$data” notation is a special notation which defines the accessing data is belonged to each object scope in the “moneyinvestedintable” array in the “foreach” loop.

```
1  <div data-bind="visible: $data.basketView()=='basket'" class="row m-row">
2      <div class="col-sm-6 col-xs-12 m-fz16">
3          <h2 data-bind="text: $data.BasketName"></h2>
4          <div data-bind="visible : $data.amount_financed > 0 || $data.status > 1">
5              <label class="switch">
6                  <input type="checkbox" data-bind="enable : $data.amount_financed > 0, checked: basketActive, click:
                   $root.changeActiveBasketStatus">
7                  <span class="m-slider round"></span>
8              </label>
9              <span class="m-ml10 m-text-slider-fix">Reinvesting</span>
10         </div>
11         <div class="m-mb10 m-flex-vertical" data-bind="visible : !($data.amount_financed > 0)&& $data.status == 1">
12             <span class="m-dashboard-round-circle m-inline-block"></span><span class="m-ml10 m-inline-block">Waiting for
                   payment</span>
13         </div>
14         <div class="row">
15             <div class="table-responsive m-mt10">
16                 <table class="table borderless m-paddding0">
17                     <tbody>
18                         <tr>
19                             <td class="col-xs-4">
20                                 <p><span data-bind="text : $data.cash_on_basket"></span></p>
21                             </td>
22                         </tr>
23                         <tr>
24                             <td class="col-xs-8">
25                                 <p>Lent</p>
26                             </td>
27                             <td class="col-xs-4">
28                                 <p><a href="#" data-bind="click: $root.getRequestDetailsForInvestment"><span data-bind="text :
                                   $data.amount_invested"></span>
29                                         <span data-bind="text : $data.symbol"></span></a></p>
30                             </td>
31                         </tr>
32                         <tr>
33                             <td class="col-xs-8">
34                                 <p>Longest loan term</p>
35                             </td>
36                             <td class="col-xs-4">
37                                 <p><span data-bind="text: $data.Loantime">3</span> year(s)</p>
38                             </td>
39                         </tr>
40                         <tr>
41                             <td class="col-xs-8">
42                                 <p>Risk ratings</p>
43                             </td>
44                             <td class="col-xs-4">
45                                 <p>
46                                     <span data-bind="visible : $data.Risk ==5">AAA</span>
47                                     <span data-bind="visible : $data.Risk ==4">AAA - AA</span>
48                                     <span data-bind="visible : $data.Risk ==3">AAA - A</span>
49                                     <span data-bind="visible : $data.Risk ==2">AAA - B</span>
50                                     <span data-bind="visible : $data.Risk ==1">AAA - C</span>
51                                 </p>
52                             </td>
53                         </tr>
54
```

Figure 34. Structure HTML markup of a basket - part1.

Figure 34 demonstrates a part of markup structure of a basket component for the basket detail section (the left section of the figure 26) when a normal view is visible (when $data.baketView() == ‘basket’). The “basketView” observable can own different values after declaration and the DOM will see the variable’s changes to update accordingly. Line 2 styles the division to takes half of the immediate parent (container) division’s width in screen size medium and above (from Ipad to larger screen size devices) and takes the whole parent’s division width in screen extra small (mobile). Line 3 illustrates name of the basket. Line 4-10 defines a division which only demonstrates when financed money

by an investor exists, meaning the “amount_financed” is greater than 0 and the basket status is more than 1. Line 6 defines an input of type checkbox which is styled by CSS with a slider style; the checkbox will toggle “reinvestment” functionality of a basket (meaning the earned interests of the basket can be reinvested in other loans or not). Line 11 constructs a division which reveals state of waiting for payment of a basket, the division will not be demonstrated when the basket is financed. Line 15 uses the CSS “row” class of Bootstrap framework to styles another division underneath the other divisions on the UI. The CSS class “table-responsive” on line 16 helps to make a table takes full-width of a parent division; if any overflow due to higher table width than a screen exists, a scroll bar will be illustrated for browsing each part of the table. Line 21 reveals how much cash in a basket is available for investment. Line 29 illustrates “amount_invested” and symbol of a currency for a basket; when a user clicks on the amount, a breakdown view will be demonstrated to see more details. Line 38 reveals loan repayment time for an investor. Line 42-54 constructs a table row, which contains five risk ranges to be chosen for five risk categories: AAA, AA, A, B, C ranging from the least risky to the riskiest (eg. if minimum required risk is A, the UI demonstrates AAA-A risk range).

```
1                   <tr>
2                       <td class="col-xs-8"><p>Amount per borrower</p></td>
3                       <td class="col-xs-4"><p><span data-bind="text : $data.amount_per_borrower_with_symbol"></span></p></td>
4                   </tr>
5                   <tr>
6                       <td class="col-xs-8"><p>Minimum interest</p></td>
7                       <td class="col-xs-4"><p><span data-bind="text : $data.showInterestRate()"></span></p></td>
8                   </tr>
9                   <tr data-bind="visible : $data.status > 1">
10                      <td class="col-xs-8">
11                          <p><strong>Interest earned</strong></p>
12                      </td>
13                      <td class="col-xs-4">
14                          <p><strong><span data-bind="text : $data.total_earned"></span></strong></p>
15                      </td>
16                  </tr>
17                  <tr data-bind="visible : $data.FundsToBeAdded > 0 && $data.status > 1">
18                      <td class="col-xs-8">
19                          <p>Funds to be added</p>
20                      </td>
21                      <td class="col-xs-4">
22                          <p data-bind="click:$root.removeFundsToBeAddedFromBasket">
23                              <span data-bind="text : $data.FundsToBeAdded_with_symbol"></span>
24                              <a href="#">
25                                  <span class="glyphicon glyphicon-remove remove_basket" aria-hidden="true"></span>
26                              </a>
27                              <a data-bind="bootstrapPopover : {content:$root.FundsToBeAddedRemove_hover}">
28                                  <span class="m-ml4 glyphicon glyphicon-info-sign" aria-hidden="true"></span>
29                              </a>
30                          </p>
31                      </td>
32                  </tr>
33                  <tr data-bind="visible : $data.amount_financed == 0 && $data.status ==1">
34                      <td class="col-xs-8">
35                          <p>Funds to be added</p>
36                      </td>
37                      <td class="col-xs-4">
38                          <p>
39                              <span data-bind="text : $data.FundsToBeAdded_with_symbol"></span>
40                          </p>
41                      </td>
42                  </tr>
43              </tbody>
44          </table>
45      </div>
46      <div class="col-xs-12 m-pointer m-flex-vertical">
47          <span data-bind="click:$data.modifyBasket"><span class="glyphicon glyphicon-cog m-dashboard-symbol-c m-mr10"></span><span>Edit
            </span></span>
48          <span data-bind="click:$root.basket_selling , visible : $data.amount_financed > 0 || $data.status > 1"><span
49                  class="fa fa-money m-ml10 m-mr10 m-dashboard-symbol-c"></span><span>Sell</span></span>
50          <span data-bind="click:$data.addFundsToBasket, visible : $data.amount_financed > 0 || $data.status > 1"><span
51                  class="glyphicon glyphicon-euro m-ml10 m-mr10 m-dashboard-symbol-c"></span><span>Add funds</span></span>
52          <span data-bind="click:$data.moveFunds, visible : $data.amount_financed > 0 || $data.status > 1"><span class="glyphicon
            glyphicon-refresh m-ml10 m-mr10 m-dashboard-symbol-c"></span><span>Move
53                  funds</span></span>
54      </div>
55  </div>
56 </div>
```

Figure 35. Structure HTML markup of a basket - part2.

Figure 35 illustrates part 2 of HTML markup for the detail section of a basket component. Line 3 structures amount per borrower with currency symbol. Line 7 forms text on the UI for a basket's minimum interest rate which can be manually define by an investor or can be set for default market rate. Line 9-16 defines a table row illustrating the earned interest which is only visible if a basket contains financed money. Line 17-31 constructs another row in the table if an investor initiated adding funds to a basket; the funds can be deleted if an investor changes his/her mind by clicking on a delete button. For an unfunded basket, line 33-42 demonstrates the amount to be financed. Line 46-53 creates several symbols to trigger other functionalities of a basket: editing, selling, adding funds to basket and move funds between baskets in which only editing function is available for an unfunded basket.

```
1   <div class="col-sm-6 col-xs-12">
2       <h2 class="btn pull-right" data-bind="click:$root.removeBasket, visible: $data.amount_invested==0 && $data.amount_financed==0 &&
        $data.status==1">
3           <span class="glyphicon glyphicon-remove remove_basket" aria-hidden="true"></span> Delete the basket </h2>
4       <h2 class="invisible">Estimated return</h2>
5       <h2 class="m-mt0">Estimated return</h2>
6       <p class="m-mt20">Estimated capital in 10 years without credit losses <span class="m-fw700" data-bind="text :
        $data.v1InvestorEarningTrialDashboard"></span>
7           <span data-bind="text : $data.symbol"></span></p>
8       <div>
9           <canvas class="m-ml0" width="750" height="320" id="canvas1" data-bind="chart: { type: 'Line', data: $data.basketChartDashboard,
            options: { observeChanges: true, throttle: 0} }"></canvas>
10      </div>
11  </div>
12  <div class="col-xs-12 m-ml15 v2-border-bottom v2-border-bottom-fix1 m-mt15 m-grey"></div>
13 </div>
```

Figure 36. Structure HTML markup of a basket - part3.

Figure 36 reveals markup of a chart of return on investment for a basket, each chart is draw on a canvas with the help of “chart.js” library. Data of a chart is computed by Javascript and input to the variable “basketChartDashboard”. The basket can be deleted if it was not funded as in line 3.

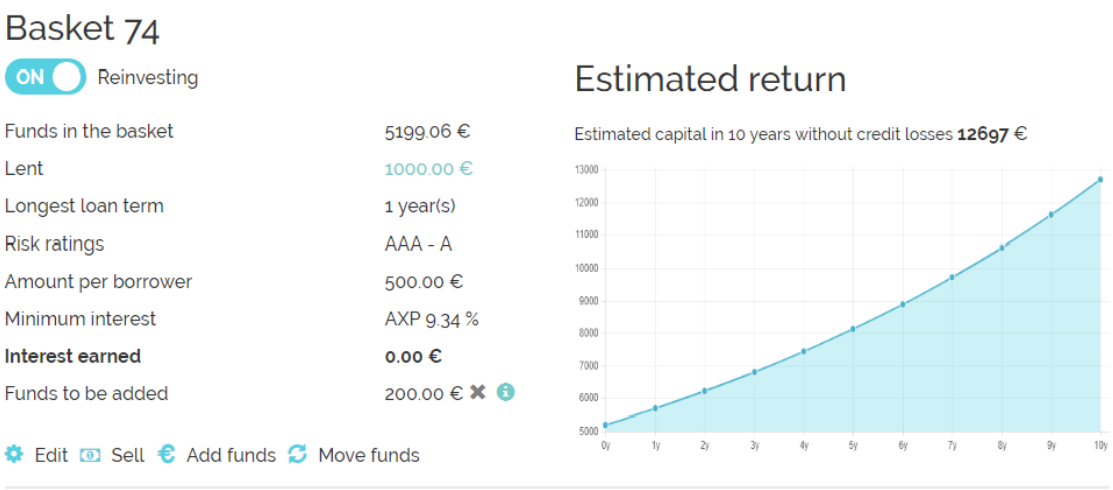


Figure 37. A financed basket.

Figure 31 illustrates a basket when it is financed. In the basket, 1000€ was lent out to borrowers and 200€ is going to be added to the basket. Delete option is not available for this basket as the logic in HTML markup hides it.

Figures 34, 35, 36 demonstrate HTML structure of a basket component. However, for each investor, data will be different, and the representing data of a basket must be dynamic revealing in real-time corresponding to the investor setting such as changing interest rate, adding funds, changing amount to be invested. This is the place when JavaScript and AJAX are needed. With KnockoutJS framework (JavaScript), a developer

can plug in different computed variables to a HTML markup, and AJAX will help to trigger a call to backend to obtain additional data.

```
1  function AppViewModel(extensionViewModel) {
2      var self = this;
3      ...
4      self.getMoneyInvested = function () {
61
62     self.getCurrencyOptions = function(){
63         $.ajax({
64             type: 'GET',
65             url: BASEURL + 'index.php/main/getCurrencyOptions/',
66             contentType: 'application/json; charset=utf-8'
67         })
68         .done(function(data) {
69             $.each(data, function (index, item) {
70                 item.url = BASEURL + "images/" + item.Abbreviation.toLowerCase() + ".png";
71             });
72             self.offeringCurrency(data);
73             if(data.length > 1){
74                 self.multipleCurrency(true);
75             }
76             self.selectedCurrency(data[0]);
77         })
78         .fail(function(jqXHR, textStatus, errorThrown) {
79             alert("There is an error, please reload the page");
80         })
81         .always(function(data){
82             if (self.loggedIn() == 'true') {
83                 //Excecute this function only on dashboard page
84                 if (typeof addingBasketPage == 'undefined') {
85                     self.getMoneyInvested();
86                 }
87             }
88         });
89     }
90
91     self.getCurrencyOptions();
92 }
93
94 $(document).ready(function () {
95     var dashboardViewModel = new AppViewModel(extensionViewModel);
96     ko.applyBindings(dashboardViewModel, document.getElementById('dashboard_wrapper'));
97     extensionViewModel.parenViewModel(dashboardViewModel);
98 });
```

Figure 38. Investor dashboard view model.

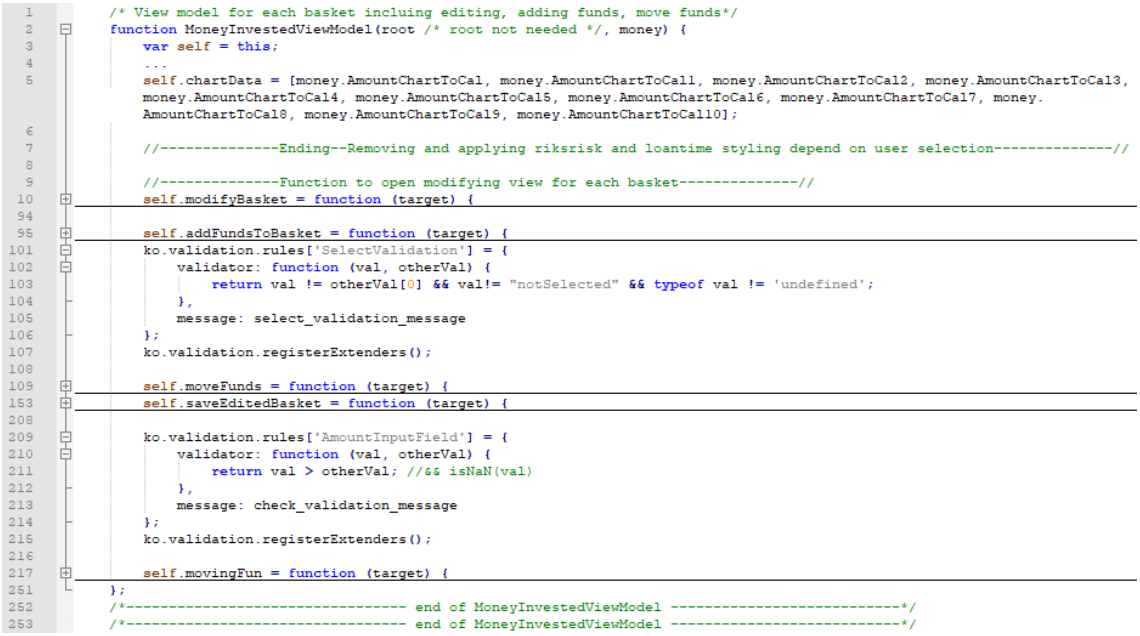
Figure 38 reveals how a JS file for the dashboard view was constructed. Line 1-91 define a function in KnockoutJS style for initializing on line 95. The function requires an object (“extensionViewModel”) which contains any additional needed data fetching from the backend. Line 96 binds the ViewModel to a division with id “dashboard_wrapper”. Only one ViewModel can bind to a DOM element at a time; therefore, header and footer on this dashboard page contain their own ViewModels which possess different scopes to the dashboard ViewModel. The separation of scopes helps to avoid problems of accidentally-redeclared variables and duplicated function names. Line 94 using jQuery library to ensure that the DOM is fully loaded when a user lands on the page to start executing the code inside by calling an anonymous JS function which will be executed immediately. The dashboard page now contains all needed variables which will be constructed inside the ViewModel. Line 91 call function “getCurrencyOption” which is defined

in line 62-89. In the function, an AJAX function written in jQuery syntax is defined. The first part of the AJAX function define which method to call to the backend server (GET method), which URL to call (to getCurrencyOptions function in the controller main, with BASEURL of the environment: development, test or production) and which type of data is expected in return (JSON).The function expects at least one object to be returned, hence line 69-71 contains a loop to add image URL for each currency for loading on the UI, after that all data is assigned to an observable “offeringCurrency” which will be used in the dashboard UI. A default currency which is the first element of return data array will be assigned to the observable “selectedCurrency”. If the AJAX function fails, an alert message will be illustrated to the user. In addition, the AJAX function when completed will call function “getMoneyInvested” if the user is on the dashboard page. This code excerpt demonstrates an example of how to construct safe asynchronous calls with jQuery library.

```
1   self.getMoneyInvested = function () {
2       $.ajax({
3           type: 'GET',
4           url: BASEURL + "index.php/investmentcontroller/getInvestedInvestments/" + auth + "/investor_dashboard",
5           contentType: 'application/json; charset=utf-8'
6       })
7       .done(function (investedmoney) {
8           if(investedmoney.length > 0){
9               self.user_symbol(investedmoney[0].user_symbol);
10              self.user_abbreviation(investedmoney[0].user_abbreviation);
11          }
12          // gets the data for the top 4 boxes in invest summary page
13          self.EstimatedTotalEarning();
14
15          self.totalInvestedMoney(investedmoney);
16          self.moneyinvested.removeAll();
17          if (typeof paymentBasketPage == 'undefined') {
18              $.each(investedmoney, function (index, money) {
19                  self.moneyinvested.push(new MoneyInvestedViewModel(self, money));
20              });
21              self.numBasket(self.moneyinvested().length);
22              var filtered_basket = self.moneyinvested().filter(function( obj ) {
23                  return (obj.amount_financed == 0 && obj.status == 1) ||
24                         (obj.FundsToBeAdded > 0.00 && obj.status > 1);
25              });
```

Figure 39. Function to obtain all baskets.

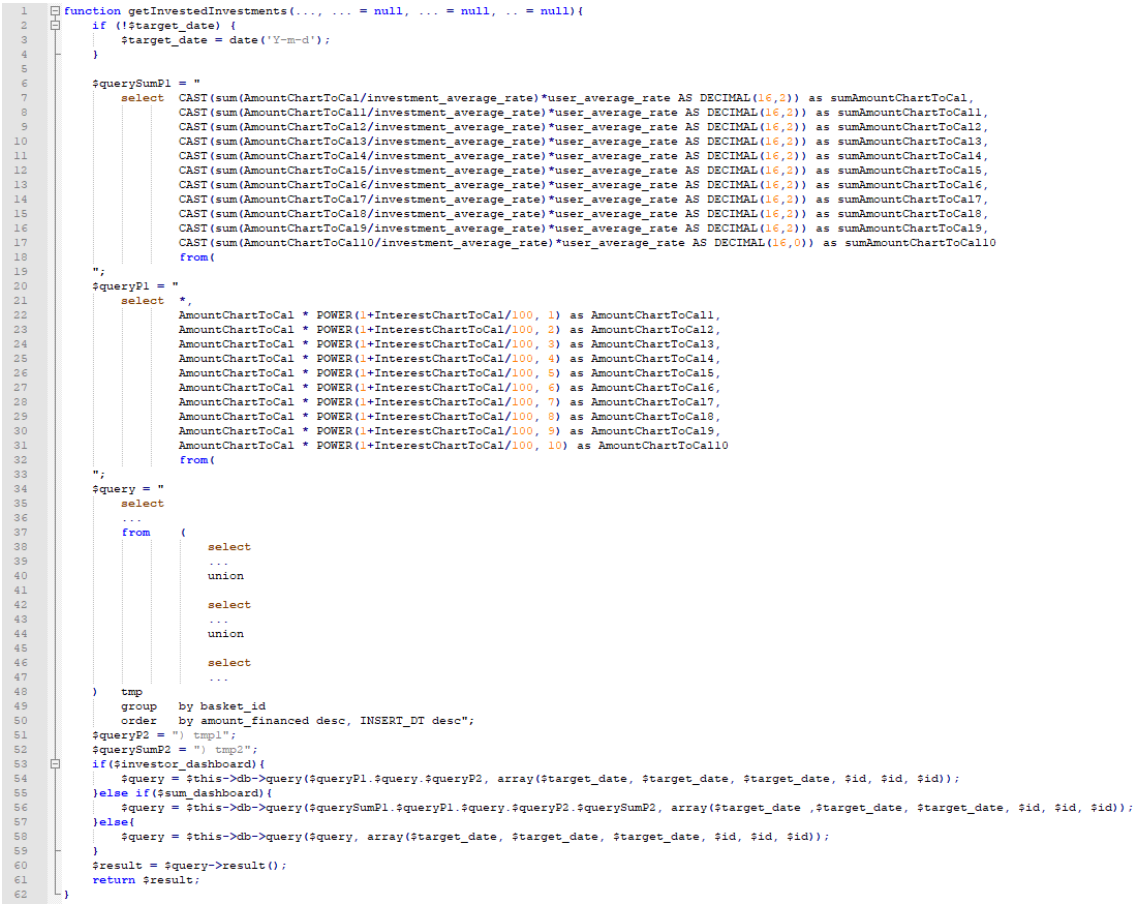
Figure 39 demonstrates an AJAX function to initialize all baskets and their functionalities. The AJAX call to backend expects to obtain an object in JSON format. If one or more added basket exists, the user preferred currency symbol and abbreviation are assigned to two observables in line 9 and 10. Line 13 calls a function to obtain summary data of return on the summary section of a dashboard. For each object in the return data array, a new basket object with its data are initialized in line 19. Line 22-25 reveals a functional way of processing array in immutable way, as the “moneyinvested” object was not mutated and a desired result set containing basket objects for payment was obtained.



```
1    /* View model for each basket incluing editing, adding funds, move funds*/
2    function MoneyInvestedViewModel(root /* root not needed */, money) {
3        var self = this;
4        ...
5        self.chartData = [money.AmountChartToCal, money.AmountChartToCal1, money.AmountChartToCal2, money.AmountChartToCal3,
         money.AmountChartToCal4, money.AmountChartToCal5, money.AmountChartToCal6, money.AmountChartToCal7, money.
         AmountChartToCal8, money.AmountChartToCal9, money.AmountChartToCal10];
6
7        //--------------Ending--Removing and applying riksrisk and loantime styling depend on user selection--------------//
8
9        //--------------Function to open modifying view for each basket--------------//
10       self.modifyBasket = function (target) {
94
95       self.addFundsToBasket = function (target) {
101      ko.validation.rules['SelectValidation'] = {
102          validator: function (val, otherVal) {
103              return val != otherVal[0] && val!= "notSelected" && typeof val != 'undefined';
104          },
105          message: select_validation_message
106      };
107      ko.validation.registerExtenders();
108
109      self.moveFunds = function (target) {
153      self.saveEditedBasket = function (target) {
208
209      ko.validation.rules['AmountInputField'] = {
210          validator: function (val, otherVal) {
211              return val > otherVal; //&& isNaN(val)
212          },
213          message: check_validation_message
214      };
215      ko.validation.registerExtenders();
216
217      self.movingFun = function (target) {
251  };
252  /*--------------------------------- end of MoneyInvestedViewModel ---------------------------*/
253  /*--------------------------------- end of MoneyInvestedViewModel ---------------------------*/
```

Figure 40. A basket ViewModel.

Figure 40 demonstrates the shortened code of a basket ViewModel with the main functions highlighted. In the code snippet, several core functions of a basket such as “modifyBasket”, “addFunsToBasket”, “moveFunds”, “saveEditedBasket” define the main interactions of an investor with a basket. Line 3 illustrates a clever way to reuse the “this” key word in JavaScript as all observable and functions start with “self” will be only bind to context of the ViewModel. Line 5 defines data for return on investment chart with all data points were already calculated in a query, therefore minimize computing on a user’s browser. Line 101-107 and 209-215 demonstrate two examples of how to use knockout validation library, which help to bind validation to an observable and keep the validation event if an element containing the observable is removed and re-added to the DOM. In fact, each basket is a complex JavaScript object which is defined in nearly 900 lines of code with more than 50 variables that directly fetched from the backend database, therefore computing work must be conducted as much as possible from backend, specifically by constructing complex and efficient SQL queries for improving user experience.



```
1  function getInvestedInvestments(..., ... = null, ... = null, .. = null){
2      if (!$target_date) {
3          $target_date = date('Y-m-d');
4      }
5
6      $querySumP1 = "
7          select  CAST(sum(AmountChartToCal/investment_average_rate)*user_average_rate AS DECIMAL(16,2)) as sumAmountChartToCal,
8                  CAST(sum(AmountChartToCal1/investment_average_rate)*user_average_rate AS DECIMAL(16,2)) as sumAmountChartToCal1,
9                  CAST(sum(AmountChartToCal2/investment_average_rate)*user_average_rate AS DECIMAL(16,2)) as sumAmountChartToCal2,
10                 CAST(sum(AmountChartToCal3/investment_average_rate)*user_average_rate AS DECIMAL(16,2)) as sumAmountChartToCal3,
11                 CAST(sum(AmountChartToCal4/investment_average_rate)*user_average_rate AS DECIMAL(16,2)) as sumAmountChartToCal4,
12                 CAST(sum(AmountChartToCal5/investment_average_rate)*user_average_rate AS DECIMAL(16,2)) as sumAmountChartToCal5,
13                 CAST(sum(AmountChartToCal6/investment_average_rate)*user_average_rate AS DECIMAL(16,2)) as sumAmountChartToCal6,
14                 CAST(sum(AmountChartToCal7/investment_average_rate)*user_average_rate AS DECIMAL(16,2)) as sumAmountChartToCal7,
15                 CAST(sum(AmountChartToCal8/investment_average_rate)*user_average_rate AS DECIMAL(16,2)) as sumAmountChartToCal8,
16                 CAST(sum(AmountChartToCal9/investment_average_rate)*user_average_rate AS DECIMAL(16,2)) as sumAmountChartToCal9,
17                 CAST(sum(AmountChartToCal10/investment_average_rate)*user_average_rate AS DECIMAL(16,0)) as sumAmountChartToCal10
18                 from(
19     ";
20     $queryP1 = "
21         select  *,
22                 AmountChartToCal * POWER(1+InterestChartToCal/100, 1) as AmountChartToCal1,
23                 AmountChartToCal * POWER(1+InterestChartToCal/100, 2) as AmountChartToCal2,
24                 AmountChartToCal * POWER(1+InterestChartToCal/100, 3) as AmountChartToCal3,
25                 AmountChartToCal * POWER(1+InterestChartToCal/100, 4) as AmountChartToCal4,
26                 AmountChartToCal * POWER(1+InterestChartToCal/100, 5) as AmountChartToCal5,
27                 AmountChartToCal * POWER(1+InterestChartToCal/100, 6) as AmountChartToCal6,
28                 AmountChartToCal * POWER(1+InterestChartToCal/100, 7) as AmountChartToCal7,
29                 AmountChartToCal * POWER(1+InterestChartToCal/100, 8) as AmountChartToCal8,
30                 AmountChartToCal * POWER(1+InterestChartToCal/100, 9) as AmountChartToCal9,
31                 AmountChartToCal * POWER(1+InterestChartToCal/100, 10) as AmountChartToCal10
32                 from(
33     ";
34     $query = "
35         select
36         ...
37         from    (
38                     select
39                     ...
40                     union
41
42                     select
43                     ...
44                     union
45
46                     select
47                     ...
48     )   tmp
49         group   by basket_id
50         order   by amount_financed desc, INSERT_DT desc";
51     $queryP2 = ") tmp1";
52     $querySumP2 = ") tmp2";
53     if($investor_dashboard){
54         $query = $this->db->query($queryP1.$query.$queryP2, array($target_date, $target_date, $target_date, $id, $id, $id));
55     }else if($sum_dashboard){
56         $query = $this->db->query($querySumP1.$queryP1.$query.$queryP2.$querySumP2, array($target_date ,$target_date, $target_date, $id, $id, $id));
57     }else{
58         $query = $this->db->query($query, array($target_date, $target_date, $target_date, $id, $id, $id));
59     }
60     $result = $query->result();
61     return $result;
62 }
```

Figure 41. An example SQL query.

Figure 41 reveals a shortened query for providing data to the investor dashboard. Different parameters can be passed in to the function to obtained different result sets. Line 56 reveals that if data is needed for the summary section of the dashboard, data points will be calculated based on the existing data set considering currency conversion rate for the user. Line 54 illustrates a query to provide data for all baskets with additional chart data points calculated. Line 58 demonstrates another way of calling the function for reporting purpose. The calculation of data points for projected earnings can be carried out in JavaScript from a result data set; however, calculation in this way brings computing process to the SQL server and using JavaScript objects for only displaying purpose on the user UI. Writing in syntax style of lines 54, 56 and 58 in CodeIgniter ensures the passing-in variables will be prepared (escaped special characters) before substituting for the “?” placeholders in the query string (“$query”), to prevent SQL injection. This query function should be in a model class (in MVC design pattern) and be called by controller(s) to provide relevant data to the UI.

5 Results

Building, maintaining and improving AuroraX's core product is a continuous process. For the investor dashboard feature, the project can be considered successful. The dashboard was rewritten to be more compelling both in visual effect and usability. No question about how to invest to AuroraX's arrives in support mailbox since released time on production on beginning of 2018, meaning the process is straightforward with an intuitive UI. Before the project, investing in AuroraX is a lengthy process with many views and irrelevant details. To achieve this result, it takes time for the development team to brainstorm about how to improve the existing feature, rewrite investor flow with existing technical constraints, obtain a design and start implementing.

In writing code for this investor dashboard feature, the author must understand every line of the existing codes and how different components are pieced together by reverse engineer method (as documentation is not available at the time). In addition, it requires courage and an innovative mind to break existing working codes, modify and improve them to deliver the final product on time. It is great to look at a coherent and functional UI with well-structured codes but difficult to say the process takes more than a month including testing and fixing. From business point of view, the CEO and the chairman were happy to see the final product, though it would be excellent if that can be delivered in one month (which author found difficult as the other developers were busy working on other features and were not able to help at the time).

6 Evaluation of Results

The completed dashboard is valuable for AuroraX as every new investor will use it in investment process. If the system contains no invested money in due to inability of investing from investors (not-functional UI), borrowers will not see any loan offers and leave the platforms for other competitors, despite the costs of (online) marketing to attract them. Solving investor problems (improving investing flow) first is the correct decision (colloquially it is the chicken and egg problem) as at the time, AuroraX started gaining traction of attracting new borrowers after the heavy costs of the previous marketing activities. Therefore, the success of the investor dashboard is critical for further steps of development of the company.

The project contains several challenges as the code-base was written in non-SPA style. It takes time to modify backend function to be callable by AJAX as well as handle properly asynchronous nature of AJAX calls. In addition, more effort was spent on styling (in CSS) the dashboard from scratch as the new design is different from the previous existing UI. Many parts of backend were not written by the author, so time must be consumed for understand the existing codebase and decide what to rewrite and what to reuse.

In the future, many improvements can be conducted to the dashboard. Unit tests should be written for every computing functions which guarantee no unexpected/new bugs appear in existing functionalities when new codes are introduced. The dashboard currently only illustrates minimum needed information for investors; it should demonstrate more informative data such as: infographic information of borrowers, historical records of recovery (recovering) in each risk rate category, multiple currencies handling (implementing). In addition, front-end code needs to be fully separated from back-end code with back-end (following REST architecture style), acting as an API server. After that, ReactJS framework can be gradually migrated to replaced KnockoutJS framework, to take advantage of reusable front-end component capability from ReactJS. Furthermore, Webpack or other front-end build tools can be used to automate the front-end build processes such as checking code format, compressing web assets (images, files etc.), removing comments, running front-end tests etc.

7 Conclusion

This thesis paper seeks to find relevant theory behind implementation of a commercial product feature. Based on this study it seems that the difference between theory and implementation is not that substantial, as the software industry standards are seen everywhere in the final product. The project was implemented in agile methodology, the software features were built on top of more-than-a-decade-age patterns and a mature backend language with reliable performance and small difficulty in using frameworks. The project helps the author to consolidate understanding of the principle: Don’t Repeat Yourself (DRY), which aims to reduce repetitive codes or technical debts by using software design patterns, software frameworks and following industry standards.

The author found that software is fragile when it is built without a modern build process and automation tools such as automated testing, automated deployment and continuous integration. Despite all the difficulties in building, software is essential to technological

development when the costs of building it are surpassed by the future values of reusability and scalability and if the software is robust, reliable and brings added-value to users' lives.

References

1 Pick-up consumer credit growth [Online]. Suomen Pankki.
URL: https://www.suomenpankki.fi/en/Statistics/mfi-balance-sheet/older-news/2017/kulutusluottojen-kasvu-kiihtynyt. Accessed 15 August 2018.

2 Finland GDP 1960-2018 [Online]. Trading Economics.
URL: https://tradingeconomics.com/finland/gdp. Accessed 15 August 2018.

3 Yadav, Subhash Chandra. An Introduction to Client Server Computing. New Age International Pvt Ltd Publishers; 2009.

4 HMTL5 – Developer guides. MDN Web Docs community [Online].
URL: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5. Accessed 15 June 2018.

5 An introduction to JavaScript [Online]. Javascript.info.
URL: https://javascript.info/intro. Accessed 15 June 2018.

6 Mike McQuillan. Introducing SQL Server. Apress; 2015.

7 DB-engine ranking [Online]. Solid IT.
URL: https://db-engines.com/en/ranking. Accessed 15 August 2018.

8 What is Software framework? [Online]. Techopedia.
URL: https://www.techopedia.com/definition/14384/software-framework. Accessed June/2018.

9 Crookshanks, Edward. Practical Enterprise Software Development Techniques. Apress; 2015.

10 A typical sprint, play-by-play [Online]. Scrum.org.
URL: https://www.scrum.org/resources/blog/typical-sprint-play-play. Accessed 15 June 2018.

11 AngularJS Tutorial [Online]. W3schools.
URL: https://www.w3schools.com/angular/tryit.asp?filename=try_ng_data-binding_two-way. Accessed 15 August 2018.

12 Knockout: The “value” binding [Online]. Knockoutjs.com.
URL: http://knockoutjs.com/documentation/value-binding.html. Accessed 15 August 2018.

13 Human Computer Interaction – brief intro [Online]. Interaction Design Foundation.
URL: https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/human-computer-interaction-brief-intro. Accessed 15 August 2018.

14 Why Software Fails [Online]. IEEE Spectrum.
URL: https://spectrum.ieee.org/computing/software/why-software-fails. Accessed 15 August 2018.

15 The ROI of User Experience [Online]. Human Factors International.
URL: https://www.youtube.com/watch?v=O94kYyzqvTc. Accessed 15 August 2018.

16 What is User Centered Design? [Online]. Interaction Design Foundation.
URL: https://www.interaction-design.org/literature/topics/user-centered-design. Accessed 15 August 2018.

17 ROI of User Experience [Online]. Human Factors International.
URL: http://www.humanfactors.com/whitepapers/ROI_of_user_experience.asp. Accessed 15 August 2018.

18 Applying REST architecture to archetype-based electronic health record systems [Online]. BMC.
URL: https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/1472-6947-13-57. Accessed 15 August 2018.

19 Server-side web frameworks [Online]. MDN Web Docs community [Online].
URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks. Accessed 15 August 2018.

20 Express history. Expressjs.com [Online].
URL: https://github.com/expressjs/express/blob/master/History.md. Accessed 15 August 2018.

21 Spring Framework Overview [Online]. Tutorials Point.
URL: https://www.tutorialspoint.com/spring/spring_overview.htm. Accessed 15 August 2018.

22 PHP: History of PHP [Online]. Php.net.
URL: http://www.php.net/manual/en/history.php.php. Accessed 15 August 2018.

23 Comparing frameworks trend [Online]. Google Trends.
URL: https://trends.google.com/trends/explore?date=today%205-y&q=laravel,CodeIgniter,Symfony,CakePhp,Zend. Accessed 15 August 2018.

24 CodeIgniter Features [Online]. British Columbia Institute of Technology.
URL: https://www.codeigniter.com/user_guide/overview/features.html. Accessed 15 August 2018.

25 CodeIgniter Working with Database [Online]. Tutorials Point.
URL: https://www.tutorialspoint.com/codeigniter/working_with_database.htm. Accessed 15 August 2018.

26 Luis Antencio. Functional programming in JavaScript. Manning; 2016.

27 What Do We Know about Software Development in Startups? [Online]. IEEE Software.
URL: https://www.infoq.com/articles/what-do-we-know-about-software-development-in-startups. Accessed 15 August 2018.

28 Finnish startups attract the most venture capital in Europe — but there's one big problem [Online]. Business Insider Nordic.
URL: https://nordic.businessinsider.com/finnish-startups-attract-the-most-venture-capital-in-europe-2017-6. Accessed 15 August 2018.

29 Marketing and sales alignment for improved effectiveness [Online]. Patterson, L. J Digit Asset Manag (2007) 3: 185.
URL: https://link.springer.com/article/10.1057/palgrave.dam.3650089. Accessed 15 August 2018.