Kickstarting Application Development with Enterprise Java
A Spring Boot Case Study

Lennart Cockx
Guangming Luo

May 3, 2017

Contents

Abstract

In this report we will explore various Java technologies and analyse what features they offer. The focus of this project is to find which frameworks are most suitable for quick prototyping of a business application. Constraints and other requirements will be defined and are then applied to a specific case study. At the end of this project we should have an excellent understanding of the technologies involved to make a online cash register application that handles orders and payment in restaurants. [WIP]

Index terms— JavaEE, Spring, Cash register, Java

1 Project Objective

1.1 Proposal

This master thesis is commissioned by Faros, a Cronos group company specialized in the development of Java web applications and primarily focused on designing Rich Internet Applications. Development within Faros is done mainly with Vaadin, Spring, HTML5, ExtJS, JavaScript, JSP, JSF, NoSQL and more. This means that during this thesis, special attention is given to these technologies in regards to design choices. Naturally, comparisons with other possible frameworks will still be done and valid arguments will be set forward to confirm these decisions.

The following case study was proposed by Faros: designing an online cash register that can be used in restaurants or at events. Existing cash register systems are expensive and for smaller establishments or temporary events, this investment is not worth it. The idea is to use existing devices from employees and customers to handle the ordering and payment process.

This project is an opportunity for us to learn about the frameworks and technologies used for the development of rich, interactive web applications. Additionally, we want to explore the best methods to quickly build a web application from scratch. Some key aspects that we will look out for are: Documentation availability, speed of new project setup, customizability, integration with other frameworks and security features. When given the corresponding documentation, it should be possible to reproduce this application. During the design and development of this project we want to describe the problems we faced and how we solved them. This will give other students the option to learn from our mistakes and hopefully help them increase their understanding of Java EE concepts.

1.2 Concept

At the start of the project we discuss the key aspects of this project with our coaches at Faros. Based on the requirements provided by them, combined with our own interests, we will define the general field we will be working in.

After we have a good overview of what aspects must be included in this project (or should be avoided), we can start deciding core features that will determine which technologies will be appropriate for this project. These choices are based on the aspects we think are

needed for developing a web application based on the aforementioned requirements. This should give us a list of possible systems and frameworks that could potentially be used during this project.

Now that we have a list of possible systems, we can start to think about the case study we want to develop. Given the general outline of the project, we can begin selecting the appropriate tools and frameworks that fit this case study. This will shorten the list to technologies specifically applicable to this project. For certain frameworks it will be difficult to obtain a shorter list, as they might all comply with the requirements we set. In this case we will make a selection of items to keep the scope of the project realistic. Usually the most popular frameworks will be selected. If a deviating set is chosen this will be elaborated on in the corresponding chapter.

Now that the tools are known, the standard procedure for application development will be followed. Relevant documentation will be constructed including a use case analysis, feature description (Nice-to-have versus Must-have), navigational models, hierarchical task analysis, UML diagrams and market research. After this we can start working on getting familiar with the chosen enterprise Java frameworks.

As soon as we are sufficiently proficient with our chosen tools, we will start working on a case study where we use the knowledge we obtained to build a functional prototype for a realistic use case scenario. During the project we might come to the conclusion that certain choices we made were suboptimal or simply do not allow further progression. In such cases, we will adjust our trajectory and change to the appropriate tools. Any time such a decision is made we will mention it in this report and provide the matching argumentation as to why we made the switch.

1.3   Project Requirements

1.3.1   Constraints

As this project is appointed by Faros, there are certain guidelines we will follow during this study. The primary constraint is the usage of Java. As Faros is a Java consultancy company, the technology stack we use should match what the employees there utilize. Within Java we are still free to choose what frameworks will be used. Another constraint is that we should not aim to create a perfectly working application with extensive styling and fail-safes, but rather focus more on experimenting with various different systems to get a better overview of the big picture and compare these systems with each other. There are also certain project-specific constraints that will be detailed in the case study.

1.3.2   Design focus

1.4   Existing systems

Before creating our own Online cash register application we searched for similar systems that already exist. One of such systems is Gloriafood [1]. This is an online system that allows any restaurant owner to sign up and start receiving orders online. A extensive layout is available where you can add a menu and payment options.

Similarly, there is ChowNow [2]. This service also handles online ordering and payment with no extra input required from the restaurant manager. ChowNow will also create a custom branded mobile application to showcase the brand and food of the restaurant.

Both of these systems are focused on the online ordering aspect and not so much on allowing people to use their devices to order food on location. In this project we want to put the emphasis on the latter part, creating an application that is meant to be used at the location itself.

2   Research and Methods

2.1   Approach

We will take a quantitative approach in this paper, exploring and comparing frameworks for each of the features required for this project. This includes: General frameworks providing a way to couple web resources to Java code. Persistence providers, SQL or noSQL databases to store customer information. Object relational mapping tools to generate mappings from existing data-stores. Security systems to allow authentication and authorization in a safe way. Front-end frameworks, providing styling options and client side scripting.

2.2   Java EE Landscape

We will not discuss every existing tool in the Java EE world, but it might be interesting to have a look at the current state of Java EE in the professional world. We will use the numbers from a survey published by Zeroturnaround to do our analysis.[9] This website has a blog for Java web developers and might display a certain bias. For more general statistics please refer to Stackoverflow's developer survey.[5]

Currently the majority of Java developers work on full-stack web applications (67%), a smaller group works on back-end code only (18%) with the rest of the developers divided into batch scripting, mobile apps, desktop applications, etc... This shows us that only in a fragment of the projects specialization is beneficial. Usually developers know and work with both front-end and back-end technologies. This is also what we will do during this project, implementing both the front-end and back-end of the application.

So which front-end and back-end frameworks are used by developers? The most commonly used web framework is Spring MVC (43%), followed closely by Spring Boot (29%). Practically, this means that over 70 percent of respondents use the Spring framework. Spring Boot itself is not really a framework, but rather a streamlined entry point to the Spring ecosystem with a "Opinionated Defaults Configuration" approach. [3][10] Other common web frameworks are JavaServer Faces (19%), Vaadin (13%) and Struts (7%). Other frameworks exist but are too numerous to list. For this project we will limit ourselves to analysing the 3 most popular frameworks. This should give us a good idea of the advantages that web frameworks offer, and how they compare. Spring Boot in particular looks promising as it provides autoconfiguration for many properties which will be advantageous for people new to the Java EE platform. It is also worth mentioning that 17 percent replied that they do not use any framework. It seems that for certain

projects using a framework is not worthwhile.

When working with frameworks, developers will make use of build tools to avoid having to manually invoke the steps in the compilation process. The most popular build tools for Java are Apache Maven (68%), Gradle (16%) and Apache Ant(11%). While using a different build tool won't directly affect the result of a project, it is important to make an appropriate choice from the start. This build tool will be shared by all teammates in a project.

To deploy the application, an application server is needed. For production, Tomcat (42%) is the primary choices for running Java web applications. The rest of the users are quite evenly distributed over the other application servers with the most popular being Jetty (12%), Wildfly (10%) and JBoss EAP (7%). Developers have the choice of running their own application server and deploying their apps to it or making an application with an embedded server. For us, using an embedded server will get our project up and running faster. We will also run an external server to learn how this changes the development cycle.

While not directly related to specific technologies, the survey also asked participants to indicate whether they are using

n/a

Finally, what types of databases are used to store information? The most used database are OracleDB (39%), MySQL (38%) and PostgreSQL (29%). All three are relational databases. The most used NoSQL databases are MongoDB (15%) and Redis (8%). Note that these results are multiple choice, as some projects use multiple databases simultaneously. For the case study, the selection of database will depend on the type of data we want to store. Most of our experience is with relational databases, so trying out a NoSQL database will be valuable experience.

Hopefully this section gave some insight into the technologies used in Enterprise Java Development and how we will approach them in our project. We want to try out various technologies and determine those most suitable for the use case and our development style. In the next section a more detailed explanation of the technologies will be given. During the exploration of these frameworks we will already create a prototype for the case study. This will give us a better idea of which features would be beneficial for this specific project.

2.3 Comparison of technologies

2.3.1 General framework

2.3.2 Persistence providers

2.3.3 Object relational mapping tools

2.3.4 Security systems

2.3.5 Front-end and Styling frameworks

2.4 Implementation

3 Case study: Online cash register

3.1 Introduction

The goal of this case study is to create a service that can be used at events and in restaurants to process orders made by customers, relay these orders to the waiters and chefs and handle the payment. This service should include an application that clients can use to view the menu and order items from it. Waiters also have their own application that can place and acknowledge orders. These orders will be send to a central system that manages all the required information. See fig. 1 for a use case diagram.

3.2 Use Cases

In this case study we can identify several distinct actors that will interact differently with the application.

Customer   The customer is the primary user of this application. They access the menu of a specified restaurant or event. They can place orders, either via the application itself or from a waiter with their own device. Finally, they can pay for the orders that they placed. Again, either via the application or a waiter.

Waiter   Like the customers, the waiter can access the menu of their restaurant and place orders for their clients. In addition to this they can display an overview of the currently queued orders, with the corresponding table numbers or location. They will also handle payments from customers through the application, or accept payment in other forms of currency.

Chef   The chef can access similar functions as the waiter. Additionally, they can modify the status of an order and mark it as completed or in progress.

Manager   The manager can modify the menu and change prices. They can also create accounts for employees and give them the corresponding privileges.
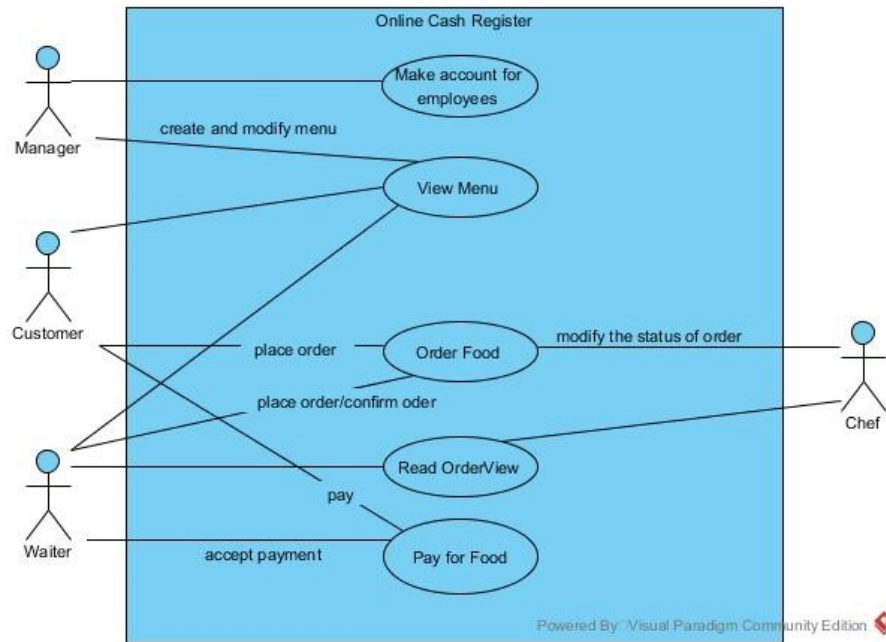
Figure 1: Use Case Diagram

3.3 Must-haves and nice-to haves

As soon as we had the use cases defined, we can specify what features are absolutely essential and which are optional. This was done in collaboration with our coaches at Faros and gave us the following specifications.

Must-Haves

- The system can be used by multiple managers or event-organizers simultaneously. (*Multi-tenancy*)

- A central application will handle orders and payment. (*Central application*)

- Waiters can use an application to send the orders of customers to the kitchen and accept payment. (*Employee interface + Payment system*)

- Customers can view the menu of the restaurant/event with their mobile device. (*Customer interface*)

- An order overview is available for the waiters and chefs. It shows the completed orders and orders which have to be made. (*Order overview*)

- Customers can add comments to certain orders. E.g. no tomato (*Special requests*)

- Customers can place their own orders with their mobile device. (*Customer Mobile ordering*)

- Prevent customers from making ridiculous orders or misusing the system. (*Misuse prevention*)

8

Nice-to-Haves

- Event organizers can run the central application on their own local network. (*Local network hosting*)

- After an order is finished, the completed order number is shown on a big screen. (*Take-a-number/Now serving System*)

- The manager can find statistics on his restaurant and orders via a web app. (*Order Statistic*s)

- Customers can split the bill or combine orders (*Bill splitting*)

- Provide an easy way for customers to use our system at an event/restaurant. Even if they dont know it yet. (*Easy access*)

- Customers can order additional food or drinks. This will be added to the total bill. (*Multiple orders*)

3.4 Technology choices

3.4.1 Application type

3.4.2 Framework choices

3.5 Design Choices

3.5.1 Client

We want to make an application for customers of restaurants and events. The customer can use this application to get an overview of the available menu items and prices. If the customers have decided, they can order the desired dishes straight from the application. Finally, the customers can pay for their meal with this application. We can differentiate a couple of different scenarios when the customer goes to a restaurant or event that supports this system. We assume that a web app is used. If a native or hybrid application is used, installing the app will be an additional step before using the system.

After (or before) the meal is finished, customers should be able to pay directly from within the application or pay a waiter. When using the application to place an order for multiple people, the customers should have the option to order and pay separately or order together and split the cost. For example: Three people want to place an order, two of which have access to the application.

- One person can place the order for the entire group.

- One person can order for himself and the person without the application. While the third person makes his own choice with his own device.

- The two persons with the application order for themselves, while the other person orders something from a waiter.

In all three cases the order will be combined into one order for that table. After the meal the bill can be paid for.

- One person decides to pay for the entire bill. This can be paid to a waiter or with his application.

- Each person pays for the part he ordered separately, with the price corresponding to what they ordered.

- Everyone decides to split up the bill equally, dividing the total cost over three people.

Typically, when people eat at a restaurant, they will place multiple orders. During the meal additional drinks might be ordered and at the end of the meal some people might want a dessert or coffee. We must design the application so it can accept multiple orders and display the sum of these orders at the end of the meal.

3.5.2   Employee

To be able to interact with customers using this system, the waiters and chefs will need their own version of this application. They will need to get an overview of the current orders placed and they should be able to modify and confirm these orders.

The order overview will have all the relevant information of the order: The different dishes and drinks ordered by the customer, the table from which the order was placed, the price and the status of the order. Additional information such as a time and date and an order number can be added for statistical purposes. The order overview could look something like displayed in fig. 2.

Order	Table		Status
2x Mac 'n cheese – 1x Coke – 1x Sprite	2	✓	Ready to serve
1x Java coffee - 1x Nougat pie	5	!	Processing
1x Chicken Spicy - 1x Beef Indi - 1x Red wine hs bottle	13	!	Processing
1x Mussel menu 2per	7	✗	In queue
1x Spr water bottle	13	✗	In queue

Figure 2: Example Order Overview

Whenever an order is served, it will be removed from the overview and the next item will be moved up. When the bill is requested, all orders from the corresponding table will be added together. After payment the orders for that table are reset to allow the next customers to place their order.

In some cases, an order might be wrong or a mistake with the number of dishes is made. The application should be flexible and allow a waiter to make small modifications to the order to correct a mistake. Additionally, a customer might make a special request for his order. This can be due to allergies or plainly because of a certain preference. It might be useful to provide a bit of space to the order overview for extra remarks.

An order might contain a lot of different items. The full names of these items will take a lot of space in the order overview. Typically, a shorthand notation is used by waiters to more efficiently note down and read orders. A similar system could be used for the order overview. Also, for showing the menu to the customer we will display a vertical list with the name of every menu item. This list will contain detailed information like the name of

the dish, the ingredients, pricing and possibly even a picture. These menu items can be sorted into categories for easy navigation. On the other side, the waiter does not need all the detailed information. Instead a compact and efficient layout is preferred to be able to easily take an order. This could be a grid layout with simple icons or abbreviations for each dish. See fig. 3b.
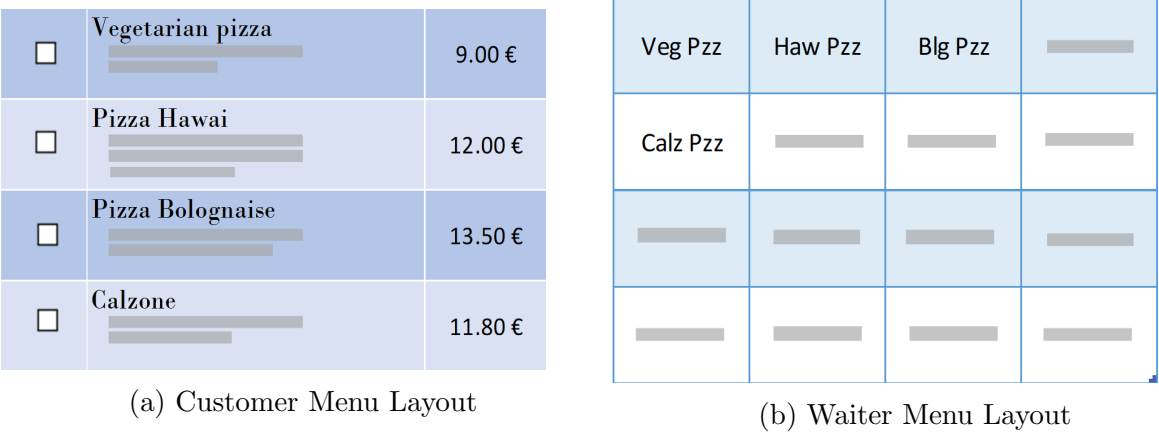

(a) Customer Menu Layout

(b) Waiter Menu Layout

Figure 3: Menu Layouts

It is possible that certain people with bad intent might try to place orders for ridiculous amounts of food, repeatedly place small orders to fill the system or even place orders for another table. Some of these situations can be prevented with proper programming like limiting the amount of orders one table can make each minute. Another way to prevent some of these issues is to have waiters confirm orders on their device before adding them to the kitchen. In day to day use, these scenarios will be rare, but its still important to think about the possible problems that might occur. While a user account for the customer might be desirable to track certain statistics and prevent misuse of the system (like mentioned above), requiring an account might not be interesting, as this can make the process of using the application slow and unintuitive for the customer.

3.5.3 Manager

The manager will be able to create new accounts for employees that work in their restaurant. They can modify properties of the restaurant and alter the menu and prices.

One of the most important aspects to consider here is multi-tenancy. The application should not only provide an interface for a certain restaurant but should be usable by different managers who own different restaurants. In other words the application should be generic and allow for customization for each restaurant separately.

3.5.4   Remarks and challenges

3.6   Major issues and their solutions

4   Results

4.1   Final Prototype

4.2   User tests

5   Conclusion

blablabla [8] [6] [**?**] [7] [4]

6 References

References

[1] Gloriafood - free online ordering system for restaurants. `https://www.gloriafood.com/`. (Accessed on 04/28/2017).

[2] Online food ordering system & app - chownow. `https://www.chownow.com/`. (Accessed on 04/28/2017).

[3] Spring boot tutorial best complete introduction — dinesh on java. `http://www.dineshonjava.com/2016/06/introduction-to-spring-boot-a-spring-boot-complete-guide.html#5`. (Accessed on 04/28/2017).

[4] What is good about the spring framework that made it among the most popular java web frameworks in 2014? - quora. `https://www.quora.com/What-is-good-about-the-Spring-Framework-that-made-it-among-the-most-popular-Java-web-frameworks-in-2014`. (Accessed on 04/27/2017).

[5] Stackoverflow developer survey. `http://stackoverflow.com/insights/survey/2017`, 2017. (Accessed on 05/03/2017).

[6] Alex Antonov. *Spring Boot Cookbook*. Packt Publishing Ltd, 2015.

[7] Faros. Faros master thesis proposal. 2016.

[8] George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.

[9] Simon Maple. Java tools and technologies landscape report 2016. `https://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-2016/`, July 2016. (Accessed on 05/03/2017).

[10] Phil Webb. Spring blog - spring boot introduction. `https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone`, August 2013. (Accessed on 05/03/2017).