MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Interactive HR Onboarding Application

LICENSE THESIS

Graduate: Csaba-László Gábor
Supervisor: Prof. Asist. Ing. Cosmina Ivan

2020

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,	HEAD OF DEPARTMENT,
Prof. dr. eng. Liviu MICLEA	**Prof. dr. eng. Rodica POTOLEA**

Graduate: **Csaba-László Gábor**

Interactive HR Onboarding Application

1. **Project proposal:** *The purpose of the project is to build an HR system for big companies where employees can log their time, send business trip requests or go through an onboarding process where they learn how to use the application.*
2. **Project contents:** *Table of contents, Introductions, Project Objectives, Bibliographic Research, Analysis and Theoretical Foundation, Detailed Design and Implementation, Testing and Validation, User's manual, Conclusions, Bibliography and Glossary.*
3. **Place of documentation:** Technical University of Cluj-Napoca, Computer Science Department
4. **Consultants: Prof. Asist. Ing. Cosmina Ivan**
5. **Date of issue of the proposal:** November 1, 2019
6. **Date of delivery:** July 8, 2020

Graduate: Csaba-László Gábor

Supervisor: Prof. Asist. Ing. Cosmina Ivan

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY

OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a) **Csaba-László Gábor** legitimat(ă) cu carte de identitate seria CJ nr. ______________ CNP ______________, autorul lucrării **Interactive HR Onboarding Application** elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la Facultatea de Automatică şi Calculatoare, Specializarea Calculatoare din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea Iulie a anului universitar 2019-2020, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

În cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă.*

Data	Nume, Prenume
______________	______________________
	Semnătura

Contents

Chapter 1

Introduction - Project Context

The purpose of this project is to create an easy to use application which eases HR tasks in an enterprise environment such as submitting business tasks which can be approved/rejected, filling timecards (users can submit the duration of time they've worked on a task) and providing training for employees by using automated ways to show how the HR management application works (Onboarding process). Also, metrics will be provided for several aspects so managers/company owners can have an overview of how their company is working. People are the real assets of a company, so they should be educated and their opinion has to be taken into account. That's why several solutions will be implemented that help the company to improve itself and to be more efficient. In the end this solution will lower costs, will improve productivity and will give the company's employees a better day to day life.

1.1 Project context

In big companies, manually managing labour work is next to impossible. Because of this, an automated way needs to be developed to carry out this task. The problem is that such systems are fairly complex and hard to use. Also, large enterprise companies waste a lot of time by training their newcomers. These new employees need time to become familiar with the company's processes. But in today's world where employees come and go, this process should be made as efficient as possible by using automated ways.

1.2 Motivation

Big companies have a lot of employees and everybody has to track their time. Different companies solve this problem differently but almost all of them have some system built for it. The problem is that those systems are legacy systems in a lot of cases or they are really complex systems which are really hard to use and new employees find it difficult to learn them. So we need an efficient solution which is easy to use and has an intuitive user interface. In the long run such a simple system will save huge amounts of money for the company and make it more efficient. Also, even if the system is simple to use, employees still need to learn it. This is where an automated learning experience such as an interactive Onboarding tutorial would shine which shows the users, in an interactive fashion, how to use the system in the correct way. This project was my own personal idea because I have seen and heard how big companies struggle with such easy tasks as managing labour work. In the beginning, the project goal was not clear, but it was refined during development.

1.3 Project content

In the second chapter of the document we are going to discuss the main objectives of the system, the secondary objectives and all the necessary resources to build such a system and to be able to deploy it.

In the third chapter we are going to dive deep into related systems and see how they are implemented. We are going to see how user-friendly they are, how intuitive they are and most importantly how easy is to learn to use them. We are going to compare those systems between them, creating tables which will show the differences between the systems but also the similarities. We are going to focus on the first introductory part of every HR system namely the onboarding process.

In the fourth chapter we are going to analyse what we are trying to be build. We are going to specify the functional and non-functional requirements. Also we'll take a look at some functional requirements and most importantly we are going to define our architecture.

In the fifth chapter we are going to design the system in more depth including the whole architecture. We are going to enumerate all of the technologies that we need to implement the system and also specify where those technologies are going to be used in the system. We will also define how the deployment will be done. In this chapter the database design will be presented.

In the sixth chapter all the testing methods will be described. This includes manual testing, unit testing and integration testing. All this testing effort contributes to the reliability of the application.

No application or system can be made without the user manual so in the seventh chapter, a throughout presentation will be made about the system: how it is used, what are the flows, also some alternate flows. And all these descriptions will include some screenshots about the application so the regular user will be able to easily understand it.

In the final chapter, namely the eighth chapter we are going to discuss our results and also take a look at ideas for future development.

At the end of the document a glossary will be provided with all the terms which are hard to understand for those people who are not familiar with these types of systems. Also the bibliography, list of figures and list of tables are shown at the end of the document.

Chapter 2

Project Objectives and Specifications

2.1 Main Objectives

The main objectives are to:

- Provide a way to track how employees spend their time on internal activities or external training events. To do this, provide an easy to use interface for employees to **fill their timecards** so that they don't waste too much time filling them, at the end of each month employees submit their timecards and project managers can review them. If the review process is over, the timecard information goes to the Payroll team which manages the budget of the company and sends the salary to the employees from different budgets based on the work done on different tasks/projects. The Payroll team can also export this information as a pdf.
- Provide a way for employees to **send business trip requests** which will be reviewed by moderators.
- Provide **real time notifications** for employees if their business trip requests or timecards were approved/rejected
- Provide several **real-time metrics** which can help to identify weaknesses in the company's structure and help business owners improve their company. For example, with the help of these metrics the company can tackle down cost related issues and solve them. Metrics include: number of business trip requests per month, per city, duration of these requests etc. For these functional requirements, charts with the metrics will be shown in the UI
- Make the **Onboarding process** as smooth as possible by providing a smartphone application which acts as an **interactive tutorial** for the users. The app is like a remote controller for the Desktop application so whenever a user wants to do something but doesn't know how to do it, it just presses the corresponding button in the app, and the Desktop app becomes interactive (the cursor moves from itself) and shows how to do that task like showing how to submit a business trip request. In this way, users quickly learn how to use the company's HR software and in the

end it is a win-win for both parties. The company doesn't have to provide trainings for the newcomers to learn the software and newcomers are not frustrated.

2.2 Secondary Objectives

Some Secondary objectives are to:

- Make the application **highly secure** so that personal information would not be leaked.
- **Authentication**: provide a way for users to log into the system by providing their email and password.

2.3 Necessary Resources

This project needs these software dependencies:

- A relational database system like MySQL to store relational data.
- A fast and reliable NoSQL database system to store the metrics, this database has to have very fast read access (to be real-time) - MongoDB
- A web application framework in Java for the backend part (Spring)
- A frontend application framework like Vue.js
- Some highly reliable asynchronous messaging system like RabbitMQ to be able to send messages from one module to the other
- Remote procedure call library for different modules to be able to log in themselves in the central application via a single sign-on mechanism (gRPC)
- Push notifications library like SockJS to notify users if their request has been accepted

As hardware requirements, a powerful PC is needed with at least 12 gigs of RAM, a quad core CPU. A dedicated GPU is not needed for this project. I don't need a dedicated server but the project will be able to be deployed on one.

Chapter 3

Bibliographic research

3.1 HR Management Software

The general approach to developing an HR Management software is to build several modules. A module is built for resume uploading and training material, another one for time management, another one for performance management etc. ([1], [2]). By having multiple subsystems we can create a decoupled application which is easier to debug, and if one module fails, the other ones can still work afterwards.

It is important to make a distinction between HRM and HRIS systems[3]. HRM means Human Resources Management while HRIS means Human Resource Information System. An HRM is *"an attempt to regulate the relationship between the employee and the employer as part of a foundation"* [4]. But this definition does not say anything about a system which holds data about employees or uses that data to speed up the process. This is where HRIS comes into play: *"a human resources information system is a system used to acquire, store, manipulate, analyze, retrieve, and distribute pertinent information about an organization's human resources"* [5].

Several such solutions exist, each with its niche features. [3] presents some open source solutions like: Sentrifugo ([6]) with features like interview schedule, time management, expenses calculation, background checks etc., Orange HRM([7]) with features like time off management.

[1] presents new features of such a system like scheduling trainings for employees, tracking resumes of employees (storing them). A history with the past trainings of the employees can be stored in a database and this data can be used later for things like performance management or salary raise calculation. The previous solutions presented a way to store only specific user data but [8] says that by storing a lot of data about users (like telephone number, photos, medical information, resume, training records) helps to efficiently utilize all the resources of a company. In this way, we don't end up with fragmented databases used for different software components. Instead, we can have multiple modules using the same database and in this way, synchronizing databases doesn't become a problem. Also, it is important to automize as many parts of the process as pos-

sible. According to [9], *"people-related costs now constitute the majority of total corporate expenditures"*.

Automating these processes is not enough, it is also important to measure performance and align the processes if necessary after a throughout evaluation. Having relevant, accurate and timely information is of great importance [10]. This means that information needs to be collected from all the modules of the system and gather them into one single place where it can be processed and analyzed. Both internal information like basic employee information (age, gender etc.) and external information like socialcultural information need to be taken into account. Also, according to [10], this data can help in several functional requirements: can provide information for operational routines like workforce management, tactical processes like training decisions or really important strategic decisions. [11] shows that we can have a data mining approach with several pipelines where each pipeline has a specific function, and this way, our huge dataset can be categorized and relevant information and patterns can be extracted.

As seen in [3], current solutions are focusing too much on operational routines, whereas tactical and strategic processes are not taken into account. The result is that these solutions only help to lower the costs of the company but don't let the company improve itself, its processes and its employees. There are some articles talking about these important processes but a real implementation is missing (only theory is provided but not a real application).

3.1.1 Common aspects

Regarding technologies used, these types of applications are developed mostly using PHP [1]. The problem is, PHP is known for its security issues and such an important system needs security. Another aspect is that PHP is slow so more powerful servers are needed which cost money.

3.1.2 Onboarding Experience

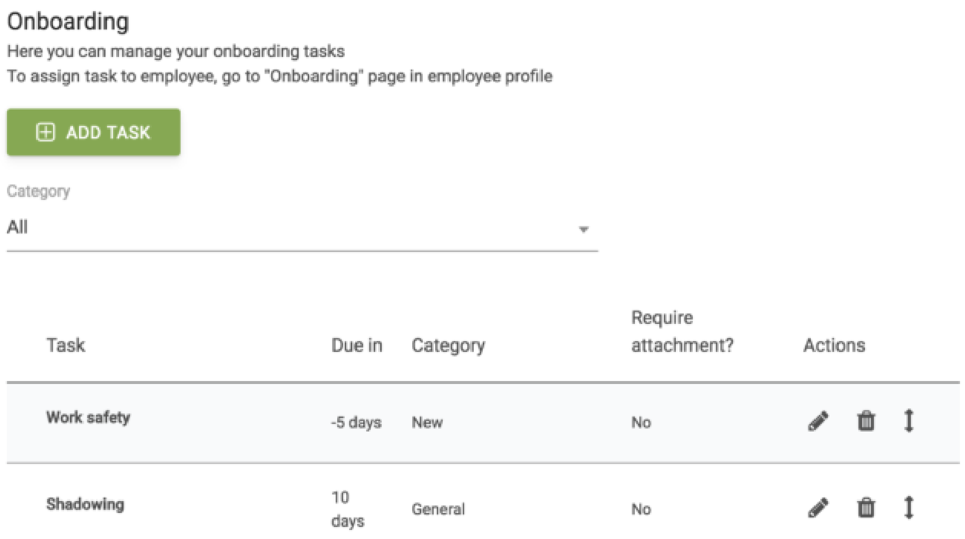
In this solution we are going to focus on the onboarding experience which is really important for new employees because, according to [12] *"Employees will decide within 10 days if they intend to stay with the organization or begin looking for a different job"*. This means, that this process should be as smooth as possible and shouldn't cause any confusions in the mind of the newcomer. Also, according to [12], *"Roughly 25% of our permanent staff have been at UNC Charlotte for two years or less and that, with expected retirements and growth, we face a growing challenge to properly orient and train hundreds of new employees"*, this means that we must take seriously people who just came to the company and utilize them as efficiently as possible knowing that most of them won't stay for a long time in the company. In today's world, employees come and go, every hour

[1] https://en.wikipedia.org/wiki/PHP

wasted because of our incompetent systems or training materials wastes a lot of money of the company. In most of the companies, when a new employee arrives, people from the company spare some of their time and train these people. And this happens every time a newcomer arrives. This process should be automated. According to [13] *"36% of companies have insufficient technology to automate or organize the onboarding process."*. In [13] a study has been carried out with the conclusion that only 57% of new employees can carry out tasks on their own. For the rest, help is needed.

Let's see some examples of Onboarding experiences:

Cake HR[14] assigns tasks to each newcomer with some attached training materials(videos, pdfs etc.) and they work on them like on real tasks. This approach is interesting but has several issues: the materials need to be updated constantly and more importantly the newcomers can be shy to ask questions from other colleagues because they think that everything is written down in the documents and they may seem dumb to others if they ask questions. Also, new-comers feel stressed on their first day of work because they got some tasks that they have to complete.


Onboarding

Here you can manage your onboarding tasks

To assign task to employee, go to "Onboarding" page in employee profile

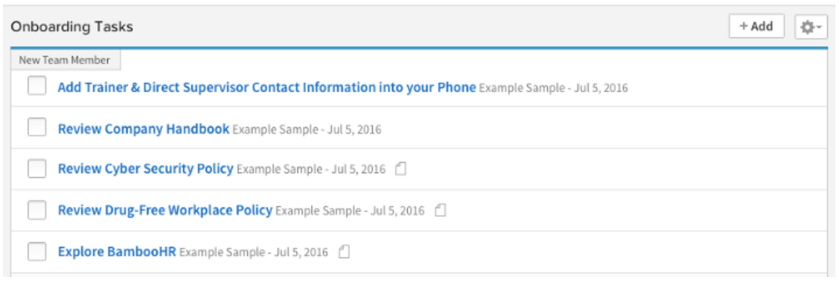
Category: All

Task	Due in	Category	Require attachment?	Actions
Work safety	-5 days	New	No	
Shadowing	10 days	General	No	

Figure 3.1: Onboarding new Employees from [14]

Observe that the new employee works on these tasks like on real ones.

Interestingly, *BambooHR* ([15]) has the same approach:


Onboarding Tasks

+ Add

New Team Member

- Add Trainer & Direct Supervisor Contact Information into your Phone Example Sample - Jul 5, 2016
- Review Company Handbook Example Sample - Jul 5, 2016
- Review Cyber Security Policy Example Sample - Jul 5, 2016
- Review Drug-Free Workplace Policy Example Sample - Jul 5, 2016
- Explore BambooHR Example Sample - Jul 5, 2016

Figure 3.2: Onboarding Experience from [15]

A newcomer receives some tasks with attached training materials and needs to complete them. *Zoho* ([16]) has a similar approach. I could only find a single interactive approach. *Namely*[2] presents a virtual way to preview the company's floors and where each room is located. This is still just a beginning in providing an interactive onboarding experience but it is a good start.

3.1.3 Comparison to other alternatives

There are several such software in the International Market, so a comparison table has been made:

Based on basic HR tasks:

Table 3.1: Comparison between systems taking into account their components

Name	Onboarding	Performance Man.	Time Off Man.	Payroll Man.	Recruitment Man.	Time and Attendance Man.	Employee Profiles
Zoho Recruit	X	-	-	-	X	-	-
Cake HR	X	X	X	-	X	X	X
Conrep [3]	X	-	X	X	X	X	X
Kronos workforce management [4]	-	X	X	X	-	X	X
Bitrix24 [5]	X	X	X	-	-	X	X
SAP SuccessFactors [6]	X	X	-	X	-	X	-
BambooHR	X	X	X	-	X	X	X

[2] https://www.namely.com/employee-onboarding-software
[3] https://www.conrep.com/products/human-resource-management/human-resource-management.php
[4] https://www.kronos.com
[5] https://www.bitrix24.com/
[6] https://www.sap.com/romania/products/human-resources-hcm/hxm-suite.html

Based on price and platform:

Table 3.2: Comparison between systems taking into account their components

Name	Price	Android	iOS	WEB/CLOUD	Windows	Mac	Linux
Zoho Recruit	$25/month/user	X	X	X	-	-	-
Cake HR	100EUR/month/user	X	X	X	-	-	-
Conrep [7]	40$/month/user	-	-	X	-	-	-
Kronos workforce management [8]	Not disclosed	X	-	X	-	-	-
Bitrix24 [9]	159,20EUR for all users	X	X	X	X	X	X
SAP SuccessFactors [10]	Not disclosed	-	-	X	-	-	-
BambooHR	8.25$/month/user	X	X	X	X	X	X

It can be seen that there isn't a bulletproof solution for every business need. This is why

- Big companies invest in multiple solutions (they buy multiple products), but the disadvantage is that they pay twice or even three times for the same functionality
- Companies developing these kinds of software partner with each other and develop a common solution, this is the case with Kronos Workforce Management and SAP SuccessFactors (the new application after the partnership is called SAP Time Management application by Kronos), but these solutions are very expensive

The problem is that having multiple solutions means having a segmented user experience and user data storage as well. So, employees need to learn multiple tools, but the biggest disadvantage is that users' data is stored in different places and because of this:

- It is harder to securely store them and comply to policies
- Data mining is not efficient on user data, so if predictions and metrics need to be carried out on this data, it will be a lot harder than it would have been if the data was stored in a single place
- It is awfully hard to update data (it needs to be updated in every single app)

If we take a look at the National, Romanian Market, then the number of solutions is not so large. But, the prices are lower somewhat: some of the solutions cost 5.90 Ron/user/month which is roughly 1.5 Eur/user/month. And even the number of features is satisfactory in these solutions but there are several things to note when choosing a solution:

[7] https://www.conrep.com/products/human-resource-management/human-resource-management.php
[8] https://www.kronos.com
[9] https://www.bitrix24.com/
[10] https://www.sap.com/romania/products/human-resources-hcm/hxm-suite.html

- Are there going to be future updates such as bug fixes, or integrations with other systems such as Slack, Microsoft Teams etc.
- Will prices go up in the future? Switching to another platform is really expensive that's why a good one has to be chosen from the beginning.

Another alternative when one needs a solution only for a national market is to choose an international product with good regional translation. Cake HR ([14]) is a good example for this. The problem is the same again: usually prices are higher.

3.2 Security in applications

Our system will consist of a frontend component (smartphone app is a part of it) and backend component. This means that the frontend component will run on client side and communicates with the backend. This is how a usual UI application works. But for this reason, strong security measures should be made like authentication and authorization.

Authentication means that the user must provide his credentials to enter the system. Authorization means that specific actions can be only carried out by specific users having specific roles.

There are several ways to authenticate a user. The most simple one is basic HTTP Auth which means that the username and password of the user are sent through the wire each time. This is really bad from a security perspective because the password should be stored in an encrypted form even in the database, sending it all the time in clear-text form is really bad because an eavesdropper can steal it.

This is where Token based Authentication comes into play. Basically, when the user authenticates for the first time, he sends his password and username, and gets back a token in exchange. He then stores that token and uses it for consecutive calls. Even in this case, the token can be stolen and used by somebody else but the biggest difference is that a token has to be renewed periodically and also, it can be easily invalidated. And most importantly, most of the people use the same or similar passwords for different sites and in this case a token helps a lot because the attacker gets access only to a single site and not to all of them.

Working mechanism of the token based Authentication:

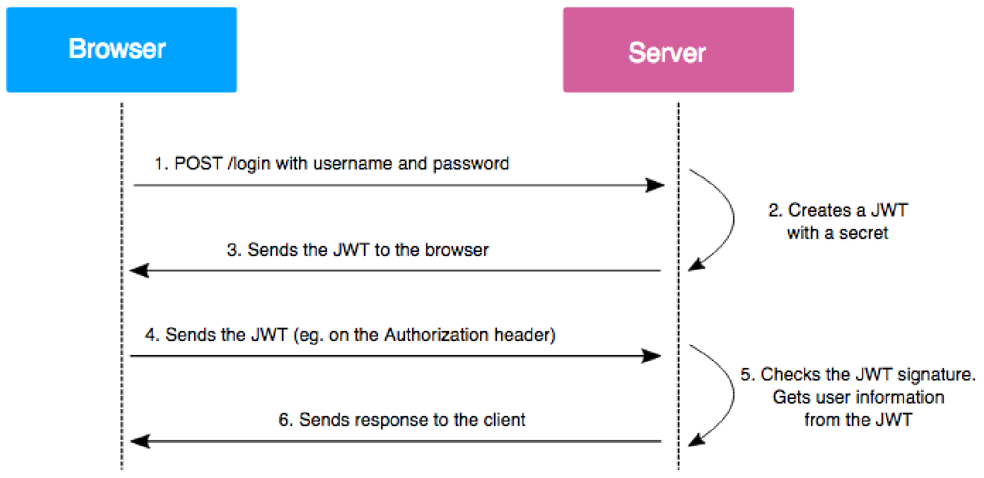

Browser
Server
1. POST /login with username and password
2. Creates a JWT with a secret
3. Sends the JWT to the browser
4. Sends the JWT (eg. on the Authorization header)
5. Checks the JWT signature. Gets user information from the JWT
6. Sends response to the client

Figure 3.3: Token based Authentication, from [17]

It is a two step process, where the browser first sends the credentials, then gets back the token, then it attaches the token for all the consecutive calls:

3.3 Push notifications

Unfortunately, HTTP in itself doesn't let us send data directly from the server to the client. A lower level protocol is needed. But before getting to the final solution, let's see how push notifications could be implemented.

This is how HTTP 1.1 works:

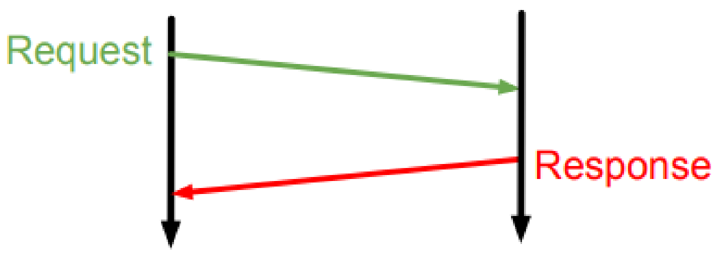


Figure 3.4: HTTP 1.1 from [18]

Later versions of HTTP allow multiple streams of data to be sent at the same time, but the concept remains the same: the client sends something, the server responds. So even if the server cannot directly send data to the client, the client can ask for data from time to time, and the server simply responds with *no* if no data is present or with the data.

So the most simple solution is:

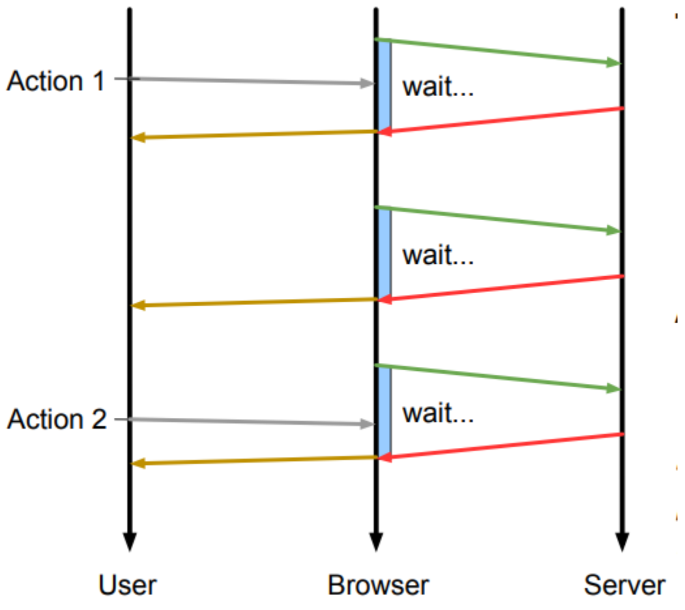


Figure 3.5: Short polling from [18]

To be able to use this solution, you need Ajax (Asynchronous JavaScript and XML). It allows you to send a request to the server **without** refreshing the webpage.

The problem with short polling is that it is not efficient at all. Every request needs to be processed on server side even if the server doesn't have anything to respond with. This wastes server resources but also wastes bandwith because every request sent (even if empty) must contain some required data/headers which take up some space. So a more efficient solution would be to just send a request to the server, the server hangs the request (doesn't respond immediately) and when it has data to respond with, it responds. Until then the client does something else and waits for the incoming data asynchrounously. This is called Long Polling.

These are the advantages of long polling:

- Almost real-time
- Efficient bandwidth utilization

Some disadvantages of long polling:

- Results in a complex backend implementation (needs to keep connection open)
- Still resource hungry

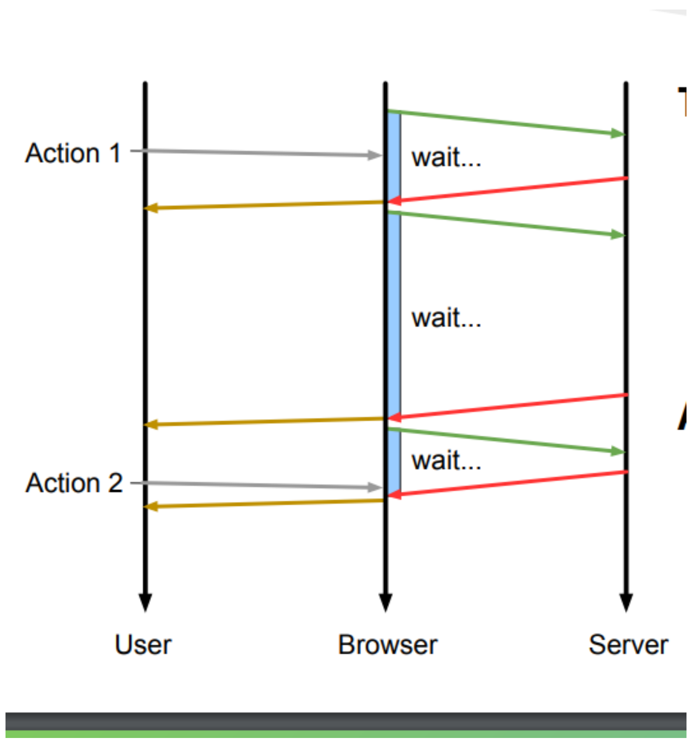


Figure 3.6: Long polling from [18]

So, building on the lower level protocol TCP, a final solution was adopted by all the major browser vendors called: WebSocket.

It works in the following way:

- client sends request to server (handshake)
- server responds and a real two-way TCP channel will be created between the client and server
- the server can directly push data to the client even if he didn't ask for it
- the only thing the client needs to do is to subscribe to the channel and listen for incoming messages

Advantages of WebSocket:

- two-way connection: client and server both can send data
- really low latency - even can be used for real-time games

- efficient bandwidth usage - HTTP headers are not sent with every request
- easy to implement in modern frameworks

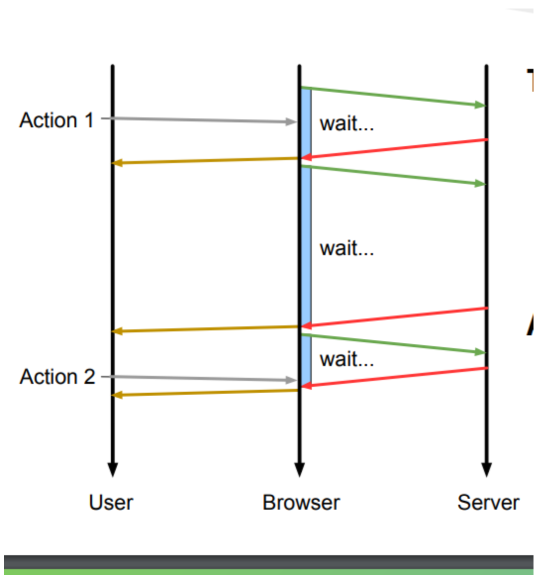


Figure 3.7: WebSocket [18]

3.4 Conclusions

Taking a look at related HR systems we can conclude that none of them have a great Onboarding experience. This is the part which needs to be improved as much as we can. Feature wise, all the systems supported most of the features needed in an HR system like timecard filling, vacation request etc. but the problem is that most of these tools are paid ones, there are very few free solutions and those lack many functionalities and only provide one, two functionalities like time management or employee management. They are not complete systems in any way. My application will be free and would provide many functionalities missing from these free systems such as onboarding experience, handling business trip requests, having a multi-role access to the system where different users can do different things etc.

All of the existing systems for which I could find the programming language in which they were developed, were developed in PHP. Now, PHP is an amazing scripting language but it is known for its security issues. Such an important system must have very good security measures which is why I have chosen Java as a programming language and Spring framework for the backend.

Also, another conclusion is to build such a system using several modules. All the modules are extensible by design.

Chapter 4

Analysis and Theoretical Foundation

In this chapter we are going to take a look at the overall architecture of our system and the use cases of the system. Also, we are going to specify the functional and non-functional requirements.

4.1 Client Server architecture - Conceptual Architecture

The application will be based on the client-server model. This means that the user interacts with the client, which gets data from the server. There are two clients: the frontend web client and the smartphone client. The user uses both clients.

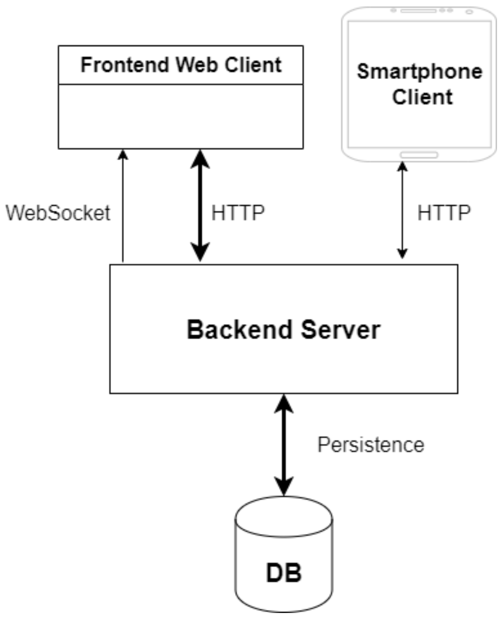


Figure 4.1: Client-Server Architecture

The frontend and backend will communicate with HTTP, and the live pushes will be carried out using WebSocket.

4.2 Onboarding Component - Conceptual Architecture

The Onboarding component will be developed separately so that later it can be reused for other modules as well. In this scenario, the user uses only the smartphone client directly and views the existing frontend web client.

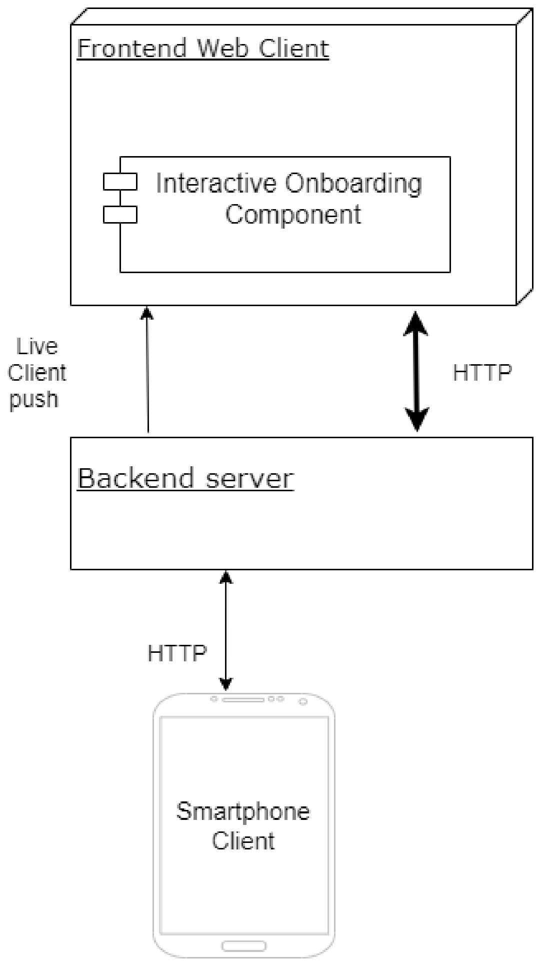


Figure 4.2: Conceptual Architecture of the Interactive Onboarding Component

Basically, the Interactive Onboarding Component will be a separate component on the UI side which gets live notifications from the main backend and acts on the main UI through Javascript DOM events.

The Smarthphone app will be developed separately and will use the same authentication mechanisms as the main UI. When the user wants to see a tutorial, the smartphone app contacts the server, which sends a live event to the Onboarding component.

4.3 Development platform

The backend is developed using Java, so it will be able to run on any operating system. Although the development platform which was used was Windows 10. The smartphone application is built using a hybrid approach. This means that it can run on both Android and iOS. The biggest advantage for this approach is that some code (like the log in flow) can be shared between the frontend component and app.

4.4 Functional requirements

There are 5 types of users in the system based on their role:

- Regular user which corresponds to the regular employee
- Moderator which can be a member of the Facility Department or other similar departments in a company
- Admin which is usually a member of the IT team
- Payroll which is the Payroll team
- PM which is a project manager

FR-1: User authentication:

Table 4.1: Authentication related FR

FR	**User**	**Moderator**	**Admin**	**Payroll**	**PM**
FR-1.1: Log in with password and email	X	X	X	X	X
FR-1.2: Log out	X	X	X	X	X
FR-1.3: View Account information	X	X	X	X	X
FR-1.4: Change password	X	X	X	X	X

FR-2: CRUD actions on resources:

Table 4.2: CRUD based FR

FR	**User**	**Moderator**	**Admin**	**Payroll**	**PM**
FR-2.1: CRUD on users/roles			X		
FR-2.2: CRUD on tasks					X
FR-2.3: CRUD on projects		X			
FR-2.4: CRUD on cities, countries, transportation means		X			

FR-3: send, approve, reject Business Trip Request:

Table 4.3: Business Trip Request related FR

FR	**User**	**Moderator**	**Admin**	**Payroll**	**PM**
FR-3.1: Send Business Trip Request	X				
FR-3.2: Get real time notifications for the status of business trip requests	X				
FR-3.3: Approve Business Trip Requests		X			
FR-3.4: View metrics related to business trip requests		X			

FR-4: fill, send, approve, reject Timecard:

Table 4.4: Timecard related FR

FR	**User**	**Moderator**	**Admin**	**Payroll**	**PM**
FR-4.1: Fill timecard	X				
FR-4.2: Approve timecards					X
FR-4.3: Export timecard related information for a given user as PDF				X	
FR-4.4: Export all the timecards from the previous month as PDF				X	
FR-4.5: Get real time notifications for the status of timecard	X				

FR-5: view interactive tutorials during the onboarding process:

Table 4.5: Interactive tutorial related FR

FR	**User**	**Moderator**	**Admin**	**Payroll**	**PM**
FR-5.1: See Interactive demo for submitting business trip request	X				
FR-5.2: See Interactive demo for filling timecard	X				

4.5 Non-functional requirements

The system must have:

- Security: passwords need to be encryped, access to resources needs to be restricted to certain roles
- Extensibility: the system should be extensible. This means that components can be removed at any time, or new components can be added at will.

Scalability is not a requirement in this application, because, although there are many employees in a large company, their number doesn't exceed ten thousand usually which is still a small number. And we are not talking about ten thousand concurrent users. Users usually fill their timecard at the end of the month, but there are three or four days left to do that and each timecard filling process takes a maximum of 10 minutes. So on average there are 10000/(3*8*6) ˜ 70 concurrent users (3 days, 8 hours working days, the duration of a session is 10 minutes so there can be 6 sessions in one hour).

4.6 Use Cases of the system

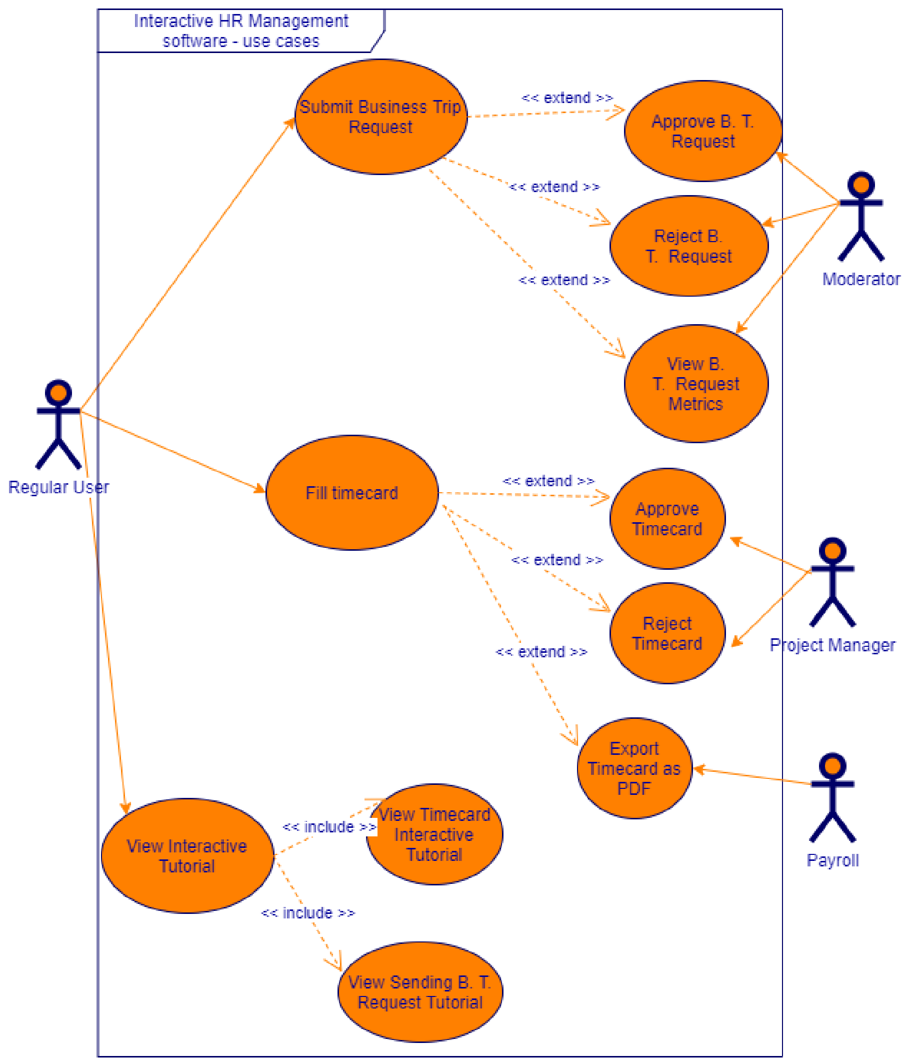


Figure 4.3: Use Case Diagram for common use cases

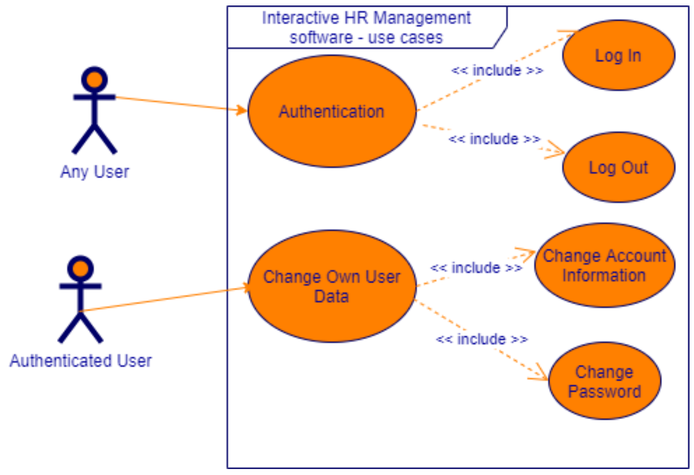


Figure 4.4: Authentication Related Use Case Diagram

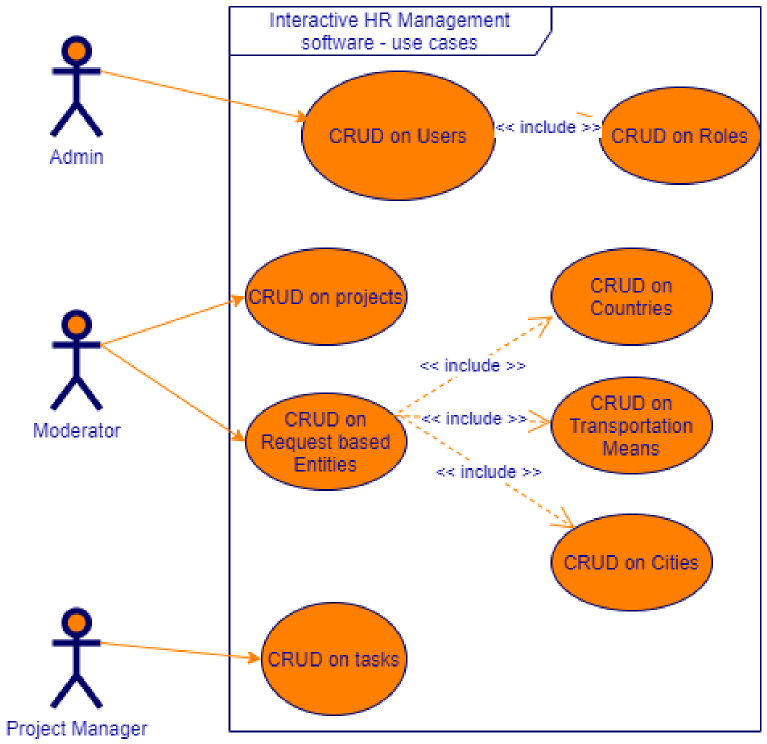


Figure 4.5: CRUD Related Use Case Diagram

4.6.1 Logging in the system

Actor logs in the system after providing an email and password.

Primary Actor: any User

Basic Flow:

1. User provides email and password.
2. System verifies user's password.

Alternative Flow:

- 2a. Password is incorrect:
    1. System rejects password.
    2. System goes back to Step 1.

Preconditions:

1. User must have a valid account.
2. Application must be open.

Postconditions:

1. User should be authenticated.

Extension Points:

- 2b. Other Type of validation error happens:
    1. System displays error message.

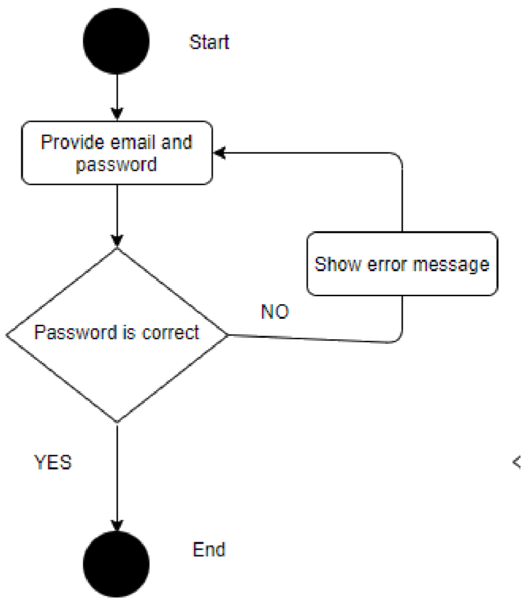


Figure 4.6: Use Case Diagram of Log In Flow

4.6.2 Log out from the system

Actor logs out from the system. Client system erases user data.

Primary Actor: any User

Basic Flow:

1. User logs out.
2. Client System deletes user data.

Alternative Flow: -

Preconditions:

1. User is logged in.

Postconditions:

1. User is logged out.
2. Client System doesn't have any information stored about the user.

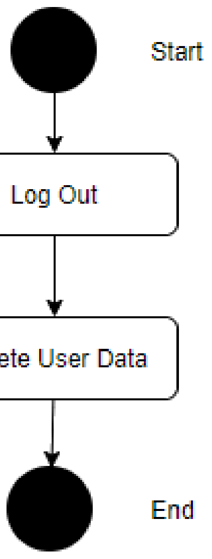


Figure 4.7: Use Case Diagram of Log Out Flow

4.6.3 Change Account information

Actor changes his firstname, lastname or email.
Primary Actor: any User
Basic Flow:

1. User changes firstname, lastname or email.
2. System verifies user provided data.

Alternative Flow:

- 2a. Data is invalid:
    1. System displays error message.
    2. System goes back to step 1.

Preconditions:

1. User is logged in.

Postconditions:

1. User data remains valid.

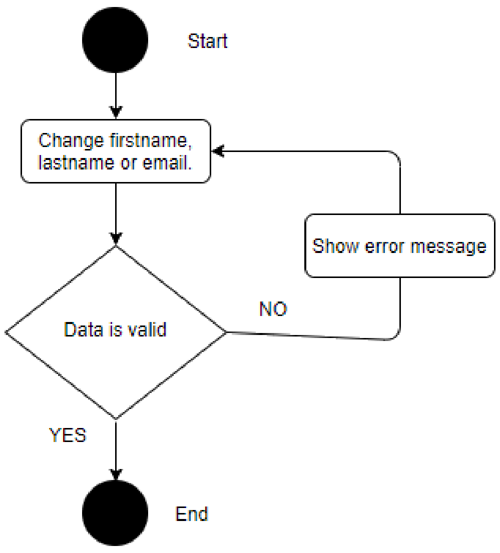


Figure 4.8: Use Case Diagram of Change Account Info Flow

4.6.4 Change password

User provides old password and new password.
Primary Actor: any User
Basic Flow:

1. User provides old password.

2. User provides new password twice.
3. System validates both passwords.

Alternative Flow:

- 3a. Old password is not valid:
  1. System displays error message.
  2. System goes back to step 1.
- 3b. New passwords do not match:
  1. System displays error message.
  2. System goes back to step 1.
- 3c. New password is not valid:
  1. System displays error message.
  2. System goes back to step 1.

Preconditions:

1. User is logged in.

Postconditions:

1. User has valid password stored in the system.

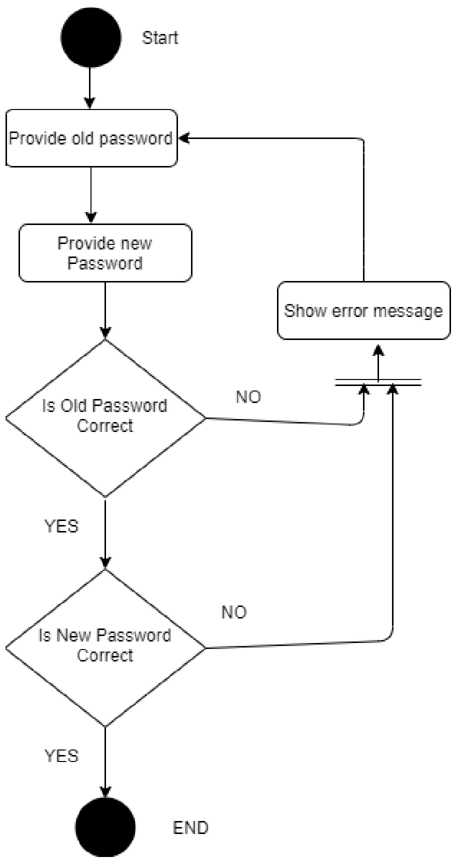


Figure 4.9: Use Case Diagram of Changing Password Flow

4.6.5 Send Business Trip Request

Regular user sends a business trip request after providing title, description, need of laptop, start date and end date.

Primary Actor: Regular User

Basic Flow:

1. User completes form with data about the request.
2. System verifies data.

Alternative Flow:

- 2a.Entered data is invalid:
  1. System displays error message.
  2. System goes back to step 1.
- 1a.User cancels:
  1. System goes back to step 1.

Preconditions:

1. User is logged in.

Postconditions:

1. Request is saved in the system for a success scenario.

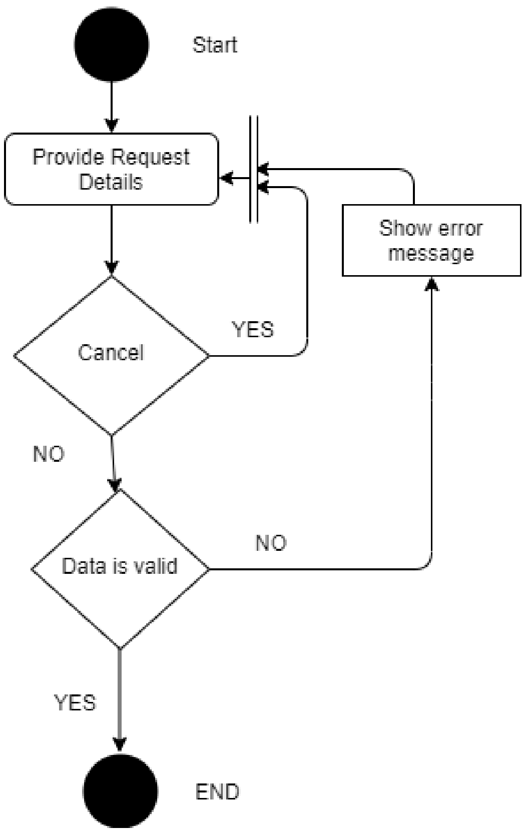


Figure 4.10: Use Case Diagram of Sending a B. T. Request Flow

4.6.6 Approving Business Trip Request

Moderator searches through sent requests and approves them.

Primary Actor: Moderator

Basic Flow:

1. Moderator searches for sent business trip requests.
2. Moderator accepts request.
3. System changes status of request to Accepted.

Alternative Flow:

-

Preconditions:

1. Moderator is logged in.

Postconditions:

1. Approved/Rejected Request is not displayed.

Extension:

- 2a. Moderator rejects request
    1. System changes status of request to Rejected.

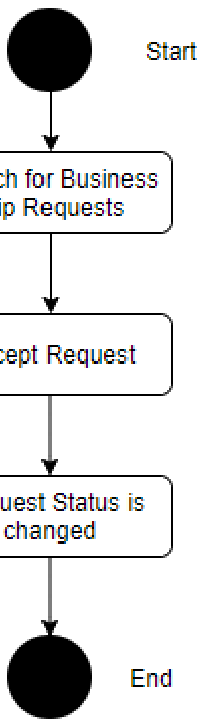


Figure 4.11: Use Case Diagram of Accepting a B. T. Request Flow

4.6.7 View Business Trip Request Metrics

Moderator view different types of metrics related to business trip requests: how many requests were done per city, per country, per transportation mean, per year and month.

Primary Actor: Moderator

Basic Flow:

1. Moderator views Business Trip Request Metrics.
2. Moderator changes Metrics Parameters.

Alternative Flow: -

Preconditions:

1. Moderator is logged in.

Postconditions: -

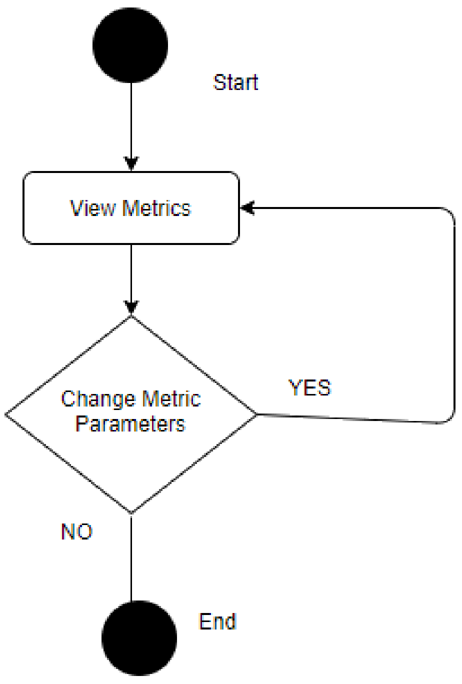


Figure 4.12: Use Case Diagram of Viewing Metrics Flow

4.6.8 Fill Timecard

Regular User fills his timecard and sends it for approval.

Primary Actor: Regular User

Basic Flow:

1. Regular User logs worked hours by selecting date, hours and task. Regular User repeats this step until done.
2. Regular User sends timecard for approval.

Alternative Flow: - Preconditions:

1. Regular User is logged in.

Postconditions:

1. Timecard is in a valid state.

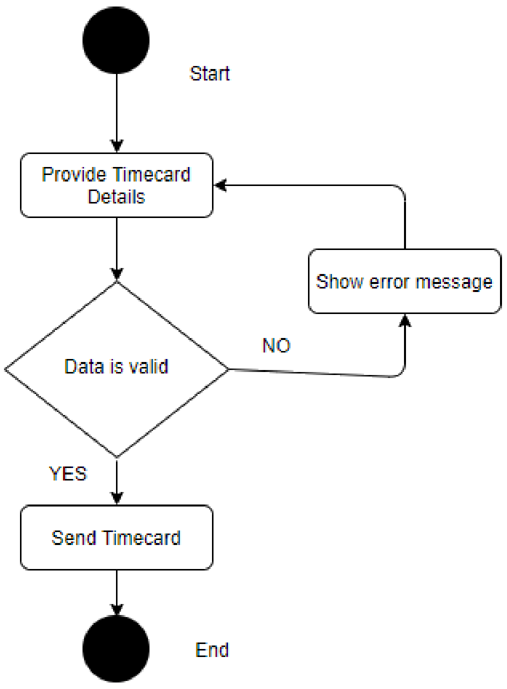


Figure 4.13: Use Case Diagram of Filling a Timecard Flow

4.6.9 Approving Timecards

Project Manager checks and approves a timecard.

Primary Actor: Project Manager

Basic Flow:

1. Project Manager selects a specific timecard.
2. Project Manager approves the timecard.
3. System changes timecard's status to Accepted

Alternative Flow: - **Preconditions:**

1. Project Manager is logged in.

Postconditions:

1. Approved/Rejected Timecard is not displayed.

Extension:

- 2a. Project Manager rejects the timecard.
    1. System changes timecard's status to Rejected

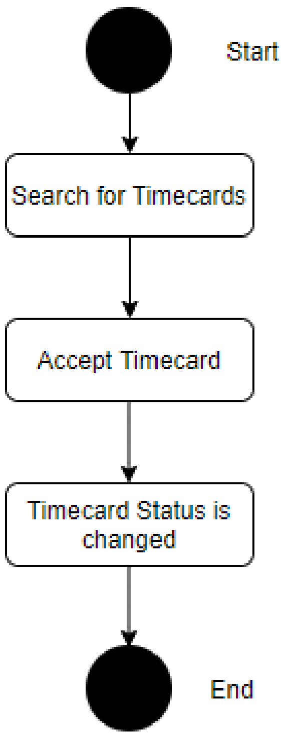


Figure 4.14: Use Case Diagram of Accepting a Timecard Flow

4.6.10 Export Timecard as PDF

Payroll member selects a timecard and exports it.

Primary Actor: Payroll

Basic Flow:

1. select timecard
2. export timecard as PDF

Alternative Flow: -

Preconditions:

1. Payroll is logged in.
2. Timecard exists.

Postconditions:

1. PDF file with Timecard information exists.

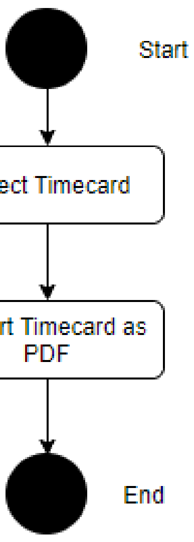


Figure 4.15: Use Case Diagram of Exporting a timecard as PDF Flow

4.6.11 View interactive tutorial

Regular user views an interactive tutorial.
Primary Actor: Regular User
Basic Flow:

1. Regular User selects tutorial.
2. Regular User views interactive tutorial.
3. System reverts all the data stored during the tutorial phase.

Alternative Flow:

- 2a. Tutorial is skipped:
  1. System goes to Step 3.

Special Requirements:

1. Smartphone device which is capable of connecting to the internet.

Preconditions:

1. Web Application is open.
2. The primary actor is authenticated and authorized in both the smartphone application and in the Web application for this use case.
3. Home screen of the smartphone application is in front of the primary actor.

Postconditions:

1. Timecard entity should remain in the same state as it was before the use case regardless of the outcome of the use case. The use case being only a simulation, shouldn't modify the primary actor's timecard.
2. Web Application should remain open regardless the outcome of the use case.

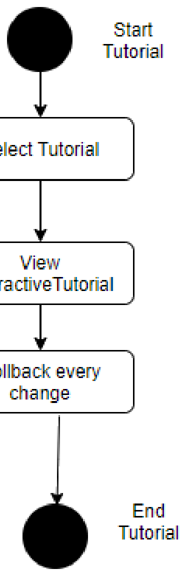


Figure 4.16: Use Case Diagram of Viewing an Interactive Tutorial Flow

The sequence diagram shows the interactions between the regular user and the system during the interactive tutorial use case:

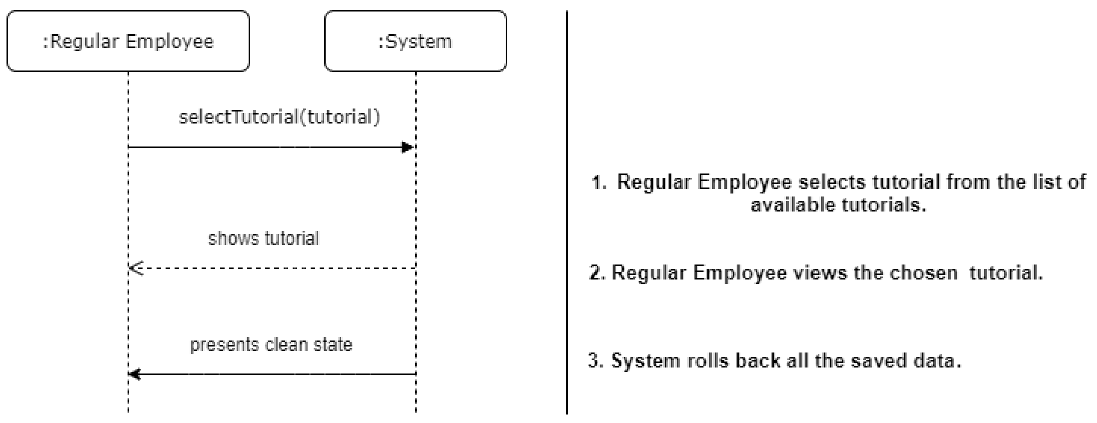

:Regular Employee
:System
selectTutorial(tutorial)
shows tutorial
presents clean state
1. Regular Employee selects tutorial from the list of available tutorials.
2. Regular Employee views the chosen tutorial.
3. System rolls back all the saved data.

Figure 4.17: System Sequence Diagram of Viewing an Interactive Tutorial Flow

Chapter 5

Detailed Design and Implementation

5.1 System Architecture

The conceptual architecture was already presented in the previous chapter. Now, the whole system architecture is presented along with all the technologies involved.

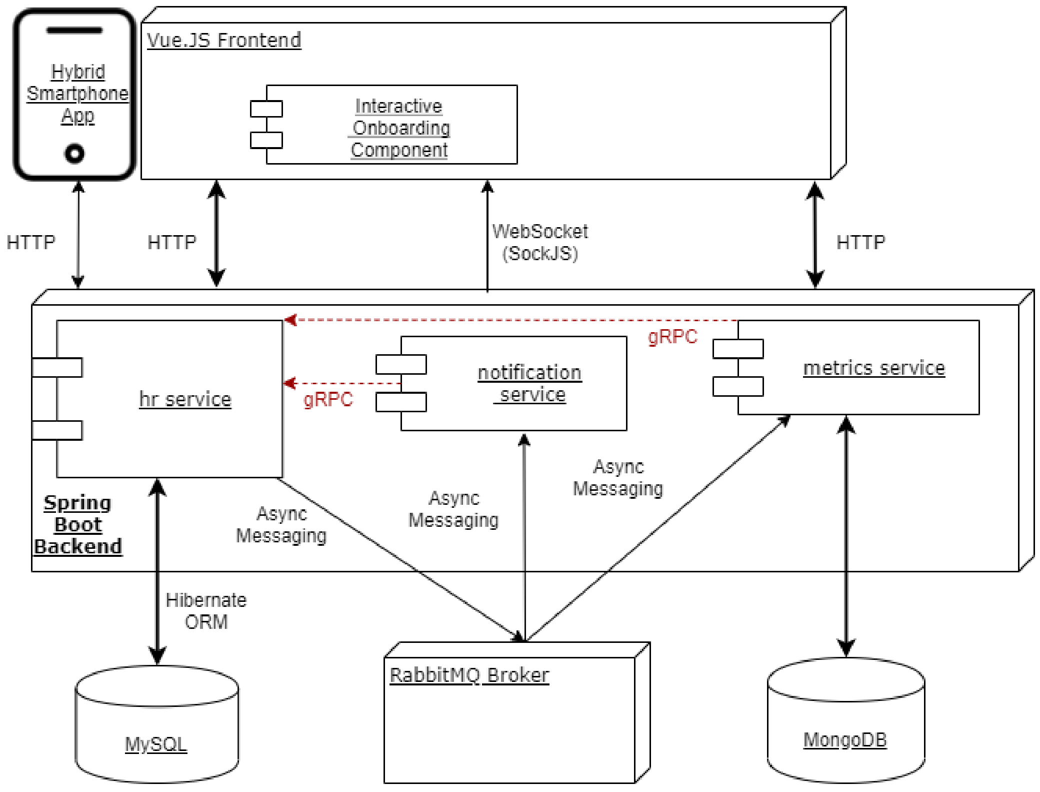


Figure 5.1: System architecture

A backend with several modules will be developed:

- The **hr service** is the core of the backend. It communicates with the main database holding all the user related data and sends that data to the frontend component.
- The **metrics service** module will save/get data from the NoSQL database and send it to the UI. It carries out all the calculation needed for metrics.
- The **notification service** module is responsible for sending live events to the UI when something important happens: the timecard or the business trip request of the user is accepted/rejected.

The communication between the modules will be carried out in both synchronous and asynchronous ways:

- Synchronous RPC will be used for authentication. The submodules will contact the main module(*hr service*) which authenticates the user so that he can access the services provided by the submodule.
- An asynchronous messaging broker will be used to send events from the main module to submodules. In this way, modules become decoupled between each other, the only common thing between them is the name of the of the queues on which data gets transferred. Asynchronous messaging is used instead of synchronous because it is non-blocking in nature. This means, that the main module can operate even if some of the submodules fail.

A relational database system is needed for the relational data in the system and a NoSQL database will be used for storing the metrics.

Spring Boot [1] already has an embedded Tomcat [2] container so there is no need for a separate deployment container. Also, we need a static file serving container, for this, Node was used.

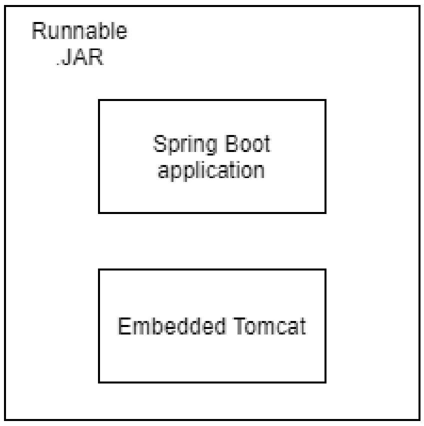


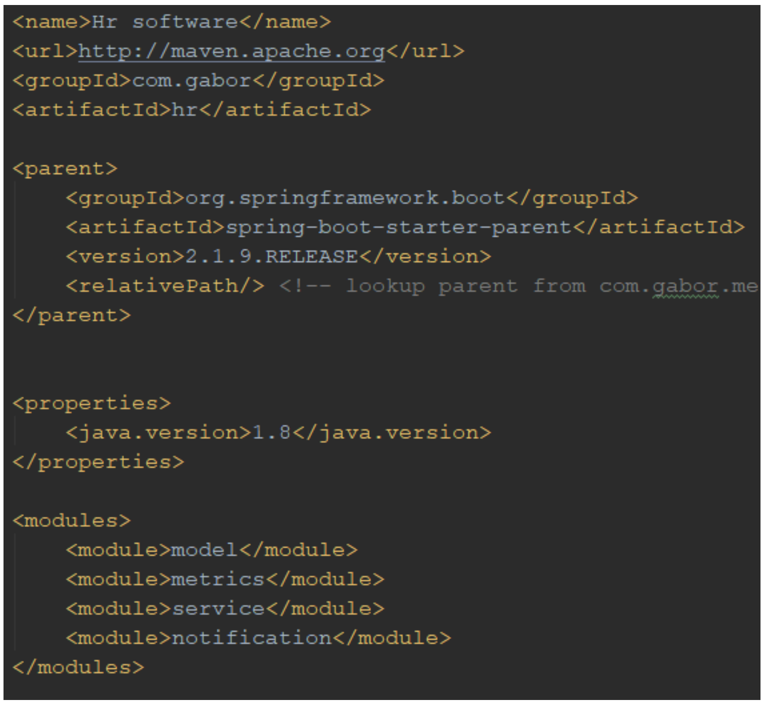
Figure 5.2: Embedded Tomcat

The interactive Onboarding component is not a separate application, meaning that

[1] https://spring.io/projects/spring-boot
[2] https://tomcat.apache.org

it is deployed together with the Frontend component, it is just a separate module/component in it. It is activated when a specific Push Notification is received from the backend. This notification contains the type of the tutorial which needs to be shown. The Interactive Tutorial Component runs on the existing UI and it has knowledge about the elements present on the page and activates their corresponding events like click event, press event etc. The Smartphone application uses the same authentication flow as the the frontend component so the main backend doesn't have to be modified. The Smartphone application calls the existing backend presented before via HTTP and the backend transmits a live WebSocket notification to the Onboarding component. The existing HR server needs to be extended to support the new push notification flow where it receives a request from the smartphone application and sends a specific websocket event to the Onboarding component.

Regarding the backend modules, they were developed as Maven[3] modules. So there is a parent module named hr and 4 submodules:


```xml
<name>Hr software</name>
<url>http://maven.apache.org</url>
<groupId>com.gabor</groupId>
<artifactId>hr</artifactId>

<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.9.RELEASE</version>
    <relativePath/> <!-- lookup parent from com.gabor.me
</parent>


<properties>
    <java.version>1.8</java.version>
</properties>

<modules>
    <module>model</module>
    <module>metrics</module>
    <module>service</module>
    <module>notification</module>
</modules>
```

Figure 5.3: Maven parent pom.xml

The **model** module is a common module between all the other modules. It only contains some Java POJO classes which common in all modules. This module could have been developed separately as a Maven library and then imported by all other components.

One of the submodules is **service**. The application cannot work without this module. All the other modules just provide additional functionalities like metrics or real time notifications but this module hosts the business logic of the app.

[3] https://maven.apache.org/

5.2 Technologies used

In this section we are going to see how and why some technologies were used in the project. This section doesn't just provide theoretical background about various technologies but also shows how they are applied in the project.

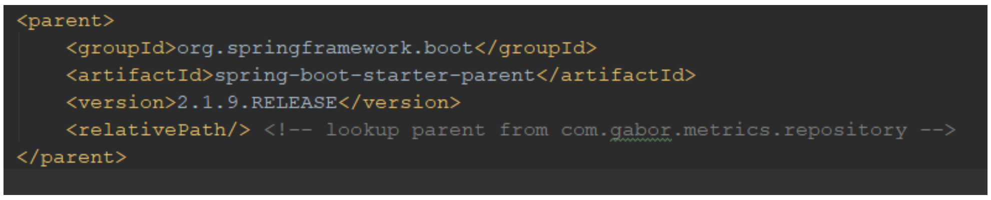
5.2.1 Spring Boot

Spring[4] is the most widely used web frameworks used since the 2000's. Initially, a container was needed to deploy the application on it, but later on in 2014, Spring Boot was introduced with the *"convention over configuration"* motto, this meant that a lot of things were already preconfigured as defaults, so it takes less time to boostrap a starter application. On the other hand, if configurations need to be changed, they can still be changed.

The biggest improvement for Spring Boot was the introduction of the embedded Tomcat container. This means, that the container is packaged together with the app and can be started as a standalone application. Advantages:

- Faster startup
- Container version can be easily changed because it is just a library of the application
- Smaller size
- Faster and easier deployment

Using Spring Boot in Maven is as simple as including it as a parent:

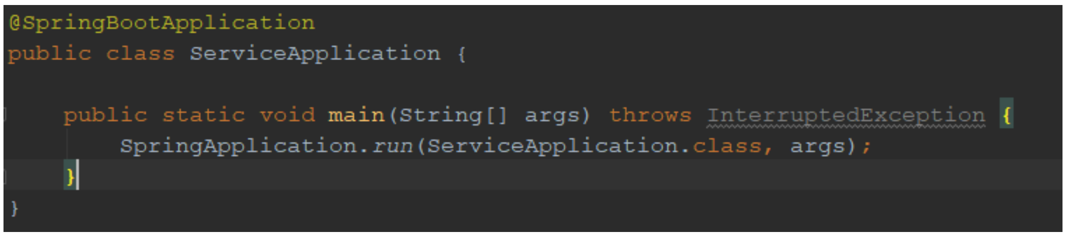


```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.9.RELEASE</version>
    <relativePath/> <!-- lookup parent from com.gabor.metrics.repository -->
</parent>
```

Figure 5.4: Maven's pom.xml in the parent module

[4] https://spring.io

The entry point of the application (it can be started as a standalone app as stated before):



```
@SpringBootApplication
public class ServiceApplication {

    public static void main(String[] args) throws InterruptedException {
        SpringApplication.run(ServiceApplication.class, args);
    }
}
```

Figure 5.5: The entry point of the Spring Boot application

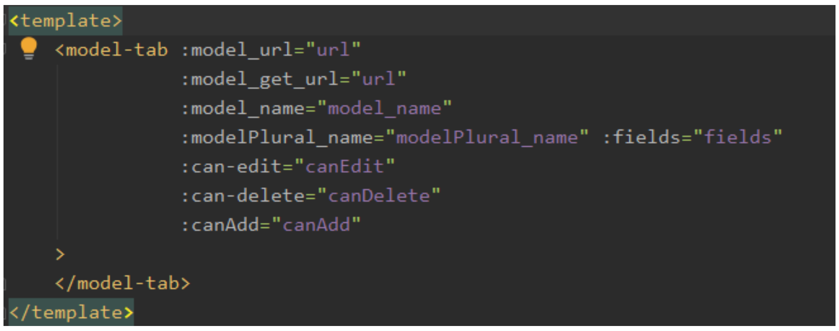
5.2.2 Vue.js

JavaScript is still the language of choice for web UI development because it is widely supported by all major browser vendors. But Javascript in itself is not enough for rapid application development. This is where frameworks such as Vue.js[5] shine.

Advantages of Vue.js over pure JavaScript:

- Reactive: this means that one-way binding can be made which updates the view if the model changes
- Virtual DOM: updating the real DOM is a costly process, Vue.js stores a virtual DOM and manipulates it, and when needed, updates the real DOM
- Code reuse: Vue.js uses templates which are basic UI components and can be parametrized and reused
- Low learning curve

A template with some parameters:



```
<template>
    <model-tab :model_url="url"
               :model_get_url="url"
               :model_name="model_name"
               :modelPlural_name="modelPlural_name" :fields="fields"
               :can-edit="canEdit"
               :can-delete="canDelete"
               :canAdd="canAdd"
    >
    </model-tab>
</template>
```

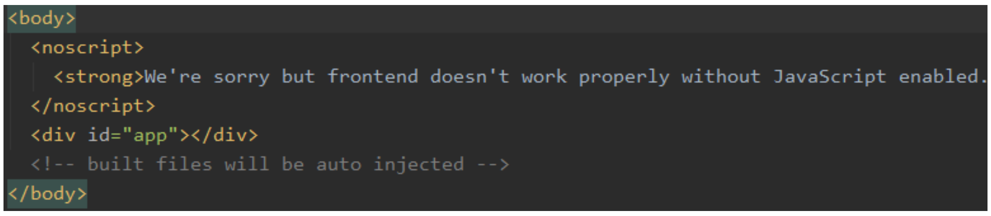
Figure 5.6: Vue.js template

Vue.js allows us to create SPA (Single Page) applications. This means that all the

[5] https://vuejs.org

components are wrapped inside a single HTML file but only those components are rendered which are necessary.

`App` is the the parent of all the components in a Vue.js app, and only a single such instance exists:

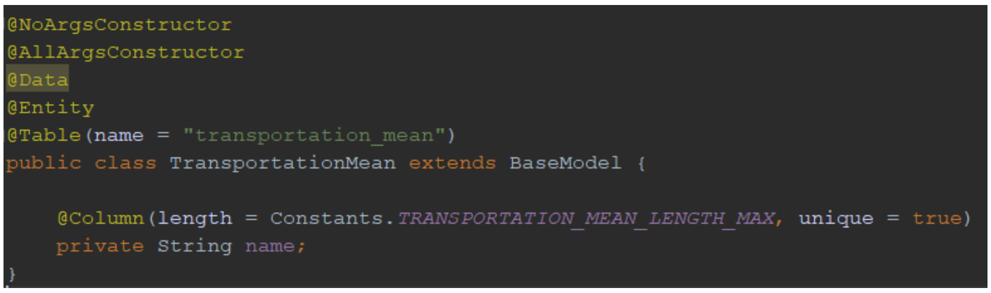


```
<body>
  <noscript>
    <strong>We're sorry but frontend doesn't work properly without JavaScript enabled.
  </noscript>
  <div id="app"></div>
  <!-- built files will be auto injected -->
</body>
```

Figure 5.7: Parent Vue.js component

5.2.3 Hibernate ORM

Hibernate[6] was used as ORM(Object relational mapper) to be able to work with objects and still be able to use a relational database without much hassle. To use Hibernate, you just need to annotate your DO class with `@Entity` and `@Table`

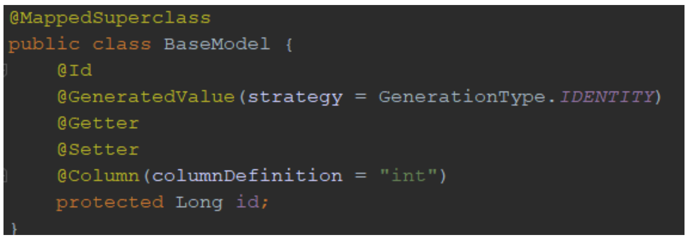


```
@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@Table(name = "transportation_mean")
public class TransportationMean extends BaseModel {

    @Column(length = Constants.TRANSPORTATION_MEAN_LENGTH_MAX, unique = true)
    private String name;
}
```

Figure 5.8: TransportationMean entity

You can notice, that this class extends a common superclass called `BaseModel` which is there to provide a unique primary key for every type of model object:



```
@MappedSuperclass
public class BaseModel {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Getter
    @Setter
    @Column(columnDefinition = "int")
    protected Long id;
}
```

Figure 5.9: BaseModel Hibernate parent entity

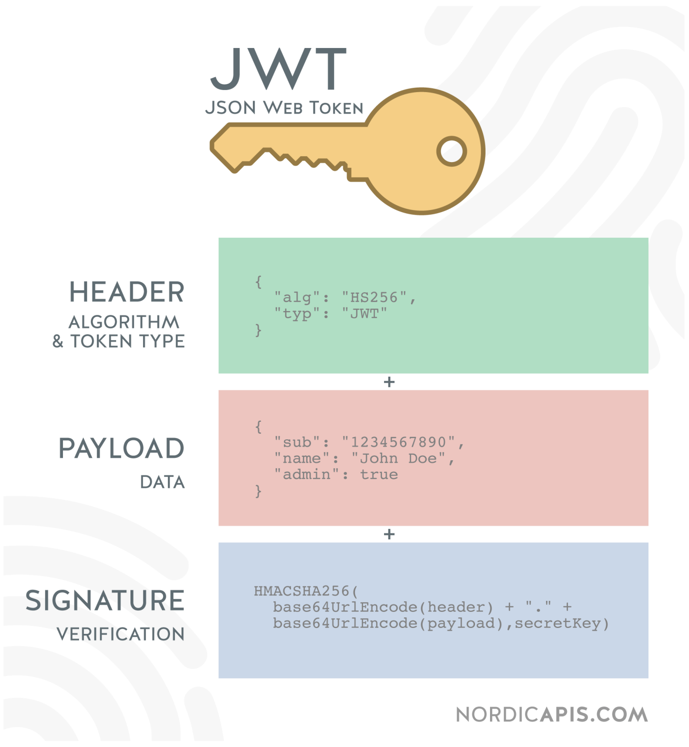
[6] https://hibernate.org

5.2.4 JWT

JWT(Json Web Token)[7] is an open source standard. It allows to create a token in such a way that it cannot be modified by any malicious user. When it is first created, it is just a JSON String but it is cryptographically signed using a **secret key**. This key is stored on the backend and nobody should have access to it. The token has 3 parts:

- The header contains basic info about the type of the token, about the algorithm used to encrypt it
- The Payload contains the actual data about the user
- the Signature is created by signing the first two parts, the signature cannot be modified, it ensures the validity of the token, if it is modified, then it becomes invalid

As mentioned before, the JWT token has 3 parts:



```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),secretKey)
```

Figure 5.10: JWT structure[8]

So, the user provides his password and email and these values are sent to the backend's **/api/auth/login** API where they are checked against the values stored in the DB(the password is hashed) and if they are valid, a JWT token is sent back to the fron-

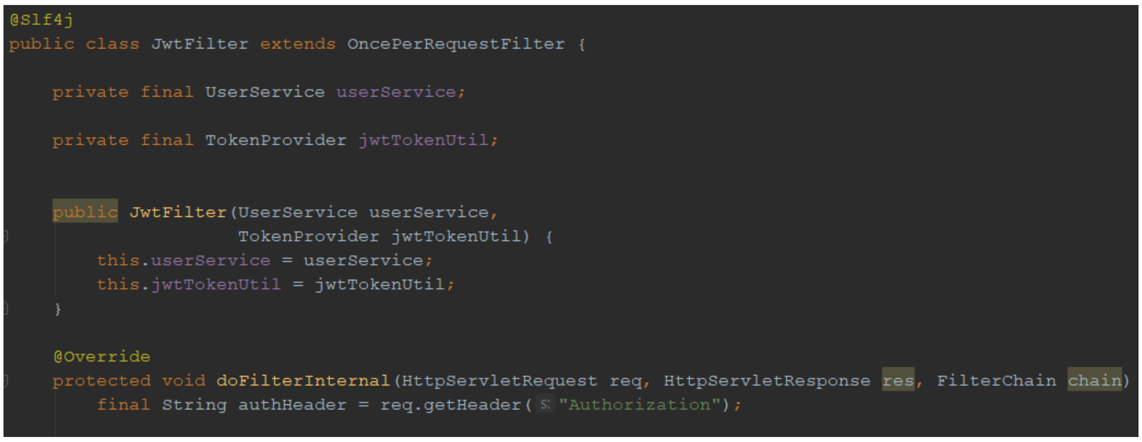
[7] https://jwt.io
[8] https://nordicapis.com/why-cant-i-just-send-jwts-without-oauth

tend containing the email and role of the user. This token is then stored in the localstorage area of the browser. Later on, every time when a new request is made to the backend, this token is attached as a header.

Advantages of JWT over other authentication mechanisms such as sessions, cookies etc.:

- can be used for mobile apps as well, cookies only work in a browser
- stateless, the server doesn't need to store any information about the user in its local cache

Before every request to the backend, the **Authorization header** is checked if it contains a valid token. The class *JwtFilter* extends Spring's *OncePerRequestFilter* meaning that it is called exactly once before each request:



```java
@Slf4j
public class JwtFilter extends OncePerRequestFilter {

    private final UserService userService;

    private final TokenProvider jwtTokenUtil;


    public JwtFilter(UserService userService,
                     TokenProvider jwtTokenUtil) {
        this.userService = userService;
        this.jwtTokenUtil = jwtTokenUtil;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest req, HttpServletResponse res, FilterChain chain)
        final String authHeader = req.getHeader( s: "Authorization");
```

Figure 5.11: JwtFilter class

5.2.5 MySQL

MySQL version 5 was used to store all the relational data including users, timecards, tasks, so MongoDB was only used to store metrics.

InnoDB was used as a database engine. It has several must have features such as foreign key constraints, support for transactions, also it is very robust when it comes to data corruption. Another widely used database engine would be MyISAM but it has neither foreign key constraints nor transaction support.

5.2.6 MongoDB

For metrics we needed a schemaless representation where lists can be stored inside objects(to calculate metrics related to business trip requests sent in a specific year and month).

MongoDB[9] was the best NoSQL DB for this usecase because it allows us to store JSON objects and also we needed a simple solution.

MongoDB is organized in **collections** which are like tables in the SQL world and **documents** which are like individual rows in a table.

`number` is present in every MongoDB collection, it stores the number of business trip requests which have that value in common. Say, you have 3 different cities, then there are 3 different documents in the collection `city_metric` and every such document stores how many business trip requests go to that city in the `number` field. `city`, `country`, `laptop`, `status`, `tansportationMean`, `year` and `duration`(expresses in days) are all indexed fields, this means that search on them is really efficient. This is important because metrics are calculated incrementally. So, if a new business trip request is saved, its parameters are broken down and the necessary MongoDB documents are updated. If a business trip request is deleted, on the other hand, the necessary documents are changed again, but this time, `number` is decremented. So, before incrementing or decrementing the documents, a search is needed to find the correct document based on the city, country, duration etc. of the request.

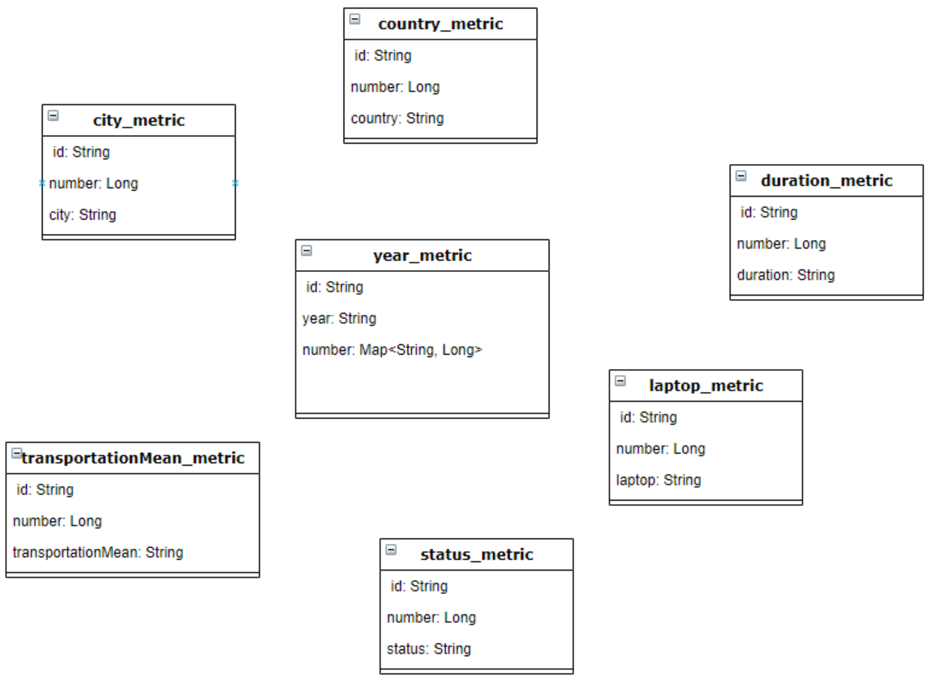


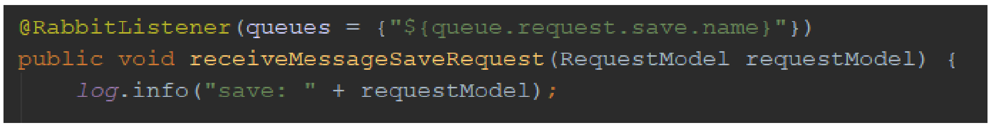
Figure 5.12: MongoDB ”schema”

[9] https://www.mongodb.com

5.2.7 RabbitMQ

RabbitMQ[10] is a messaging broker capable of providing several types of messages like fanout, direct or topic.

In the project, the Topic based exchange mechanism was used which means that a Topic can include multiple queues... The most basic element of RabbitMQ is a queue and when a message is sent to a topic, a routing key is specified and all those queues which match the routing key, receive the message. On the other end of the queue, listeners are waiting to receive messages:



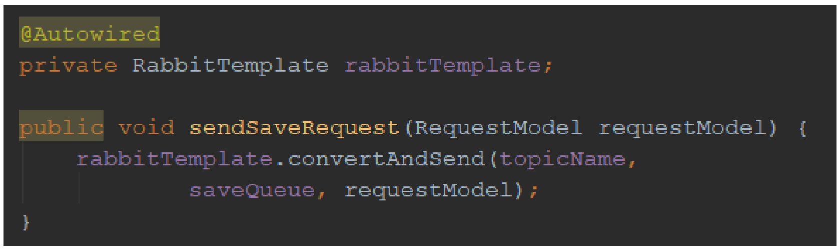
```
@RabbitListener(queues = {"${queue.request.save.name}"})
public void receiveMessageSaveRequest(RequestModel requestModel) {
    log.info("save: " + requestModel);
```

Figure 5.13: RabbitMQ listener

For example, when a a business trip request is accepted, 2 things need to happen:

- The status based metrics need to be updated, so the *metrics* module subscribes to the `metric.request.accept` queue whose routing key is `*.request.accept`
- The user whose request was accepted needs to get a live notification, so the *notification* module subscribes to the `notification.request.accept` queue whose routing key is `*.request.accept`

This means that if a message is sent to either `notification.request.accept` or `metric.request.accept`, both queues receive the messages, which is exactly what we want. This implementation may seem unnecessary but it's not. A single message can be only be delivered to one consumer at a time. This means, that we must have two different queues for our use case.



```
@Autowired
private RabbitTemplate rabbitTemplate;

public void sendSaveRequest(RequestModel requestModel) {
    rabbitTemplate.convertAndSend(topicName,
            saveQueue, requestModel);
}
```

Figure 5.14: Sending a RabbitMQ message

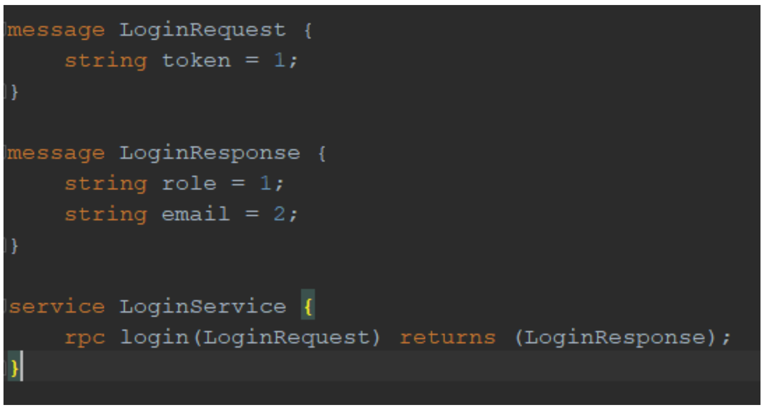
5.2.8 gRPC

We used RabbitMQ for asynchronous communication between our modules. But our *metrics* and *notification* modules need a way to contact the main service to provide

[10] https://www.rabbitmq.com

authentication. This should be done in a blocking way and also reqlly quickly. This is where gRPC[11] comes into play. gRPC is built on top of HTTP 2 and provides a language independent remote procedure call mechanism.

You just create a protobuf file having a .proto extension with the help of an IDL(Interface description language) and the gRPC compiler will compile it to whatever language you are using:



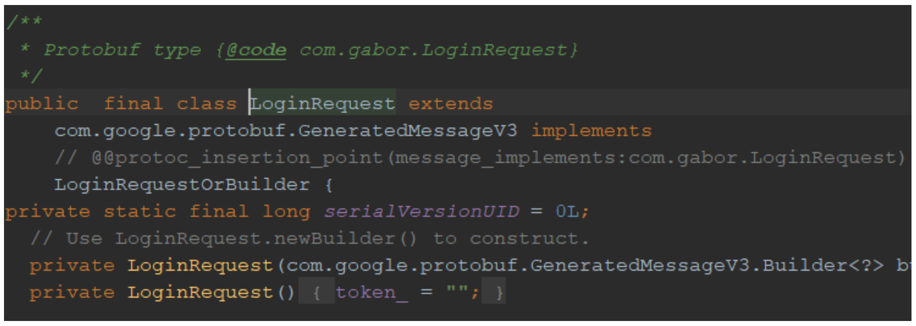
```
message LoginRequest {
    string token = 1;
}

message LoginResponse {
    string role = 1;
    string email = 2;
}

service LoginService {
    rpc login(LoginRequest) returns (LoginResponse);
}
```

Figure 5.15: Our .proto file used for login

This .proto file, after compiling it, will be turned into specific Java classes such as:

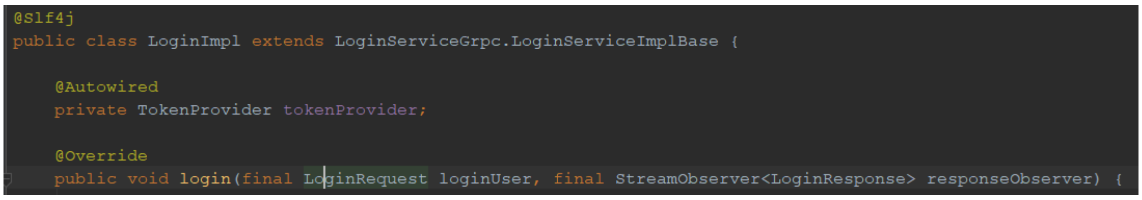


```
/**
 * Protobuf type {@code com.gabor.LoginRequest}
 */
public  final class LoginRequest extends
    com.google.protobuf.GeneratedMessageV3 implements
    // @@protoc_insertion_point(message_implements:com.gabor.LoginRequest)
    LoginRequestOrBuilder {
private static final long serialVersionUID = 0L;
  // Use LoginRequest.newBuilder() to construct.
  private LoginRequest(com.google.protobuf.GeneratedMessageV3.Builder<?> b
  private LoginRequest() { token_ = ""; }
```

Figure 5.16: Generated Java Class

The `LoginService` is generated in the same way, but only in an interface is generated. To implement the interface:

[11] https://grpc.io



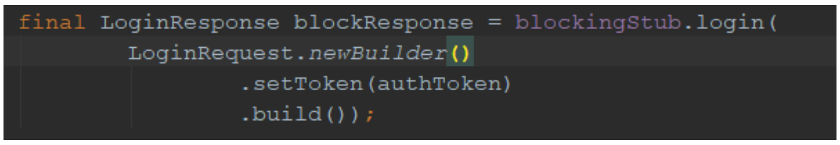
```
@Slf4j
public class LoginImpl extends LoginServiceGrpc.LoginServiceImplBase {

    @Autowired
    private TokenProvider tokenProvider;

    @Override
    public void login(final LoginRequest loginUser, final StreamObserver<LoginResponse> responseObserver) {
```

Figure 5.17: Implementing LoginService

To use the interface on client side, we can create an instance of it (called a stub which doesn't have an implementation - under the hood it just calls the object on the server side) and then call our method:



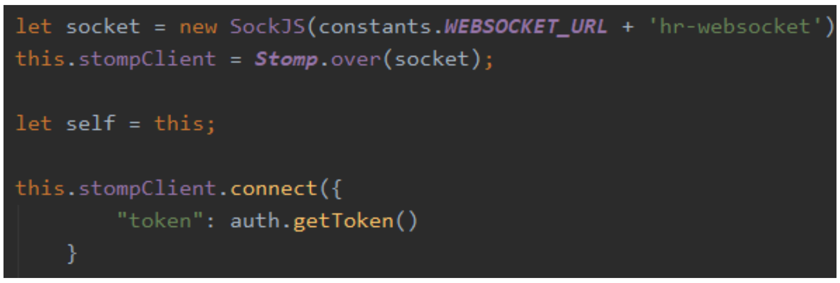
```
final LoginResponse blockResponse = blockingStub.login(
        LoginRequest.newBuilder()
                .setToken(authToken)
                .build());
```

Figure 5.18: Calling the generated method

5.2.9 SockJS

SockJS[12] is a library to ease our work with WebSocket. WebSocket itself is a protocol, so we need a wrapper around it.

Its usage is really simple. Just connect to the endpoint where Websocket is hosted on the backend side:



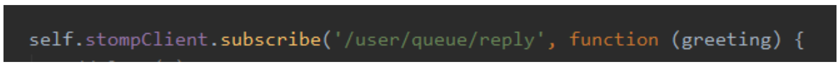
```
let socket = new SockJS(constants.WEBSOCKET_URL + 'hr-websocket')
this.stompClient = Stomp.over(socket);

let self = this;

this.stompClient.connect({
        "token": auth.getToken()
    }
```

Figure 5.19: Connect with SockJS

After that, we need to subscribe to the endpoint from where data is going to come. Also, we need to provide a callback function which is called every time a new request comes.



```
self.stompClient.subscribe('/user/queue/reply', function (greeting) {
```

Figure 5.20: Subscribe with SockJS

[12] https://github.com/sockjs

5.2.10 WebView

WebView[13] is a technology which allows us to package a website into a mobile application. It works on both iOS and Android and even Windows Phone. This means that a Responsive website has to be designed which adjusts itself to various sizes and devices.

The advantages of Webview are numerous:

- Can reuse code from the frontend code like login code, logout etc.
- No need to create separate applications for iOS and Android, can have a single codebase which works fine on both systems

5.3 Layers

Layers correspond to packages in the project. The DO -> DTO and DTO -> DO mapping is done by the *ModelMapper* library. Every Entity in the project has the corresponding Controller which supports CRUD operations on the entity, a service and a repository. Controllers return DTOs to the frontend instead of returning DOs.

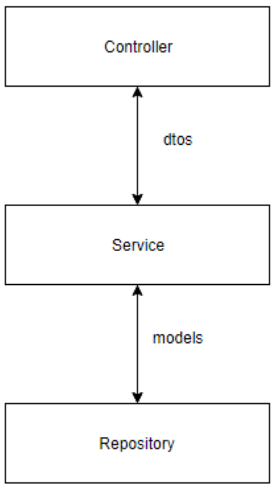


Figure 5.21: Layers

5.4 Class diagram

Not every class is represented on these diagrams because there are so many classes in the project.

[13] https://developer.android.com/reference/android/webkit/WebView

5.4.1 Controllers

All the controllers extend a main *CrudController* class that supports CRUD operations. Also, every controller uses the corresponding service e.g. *CityController* uses *CityService*.

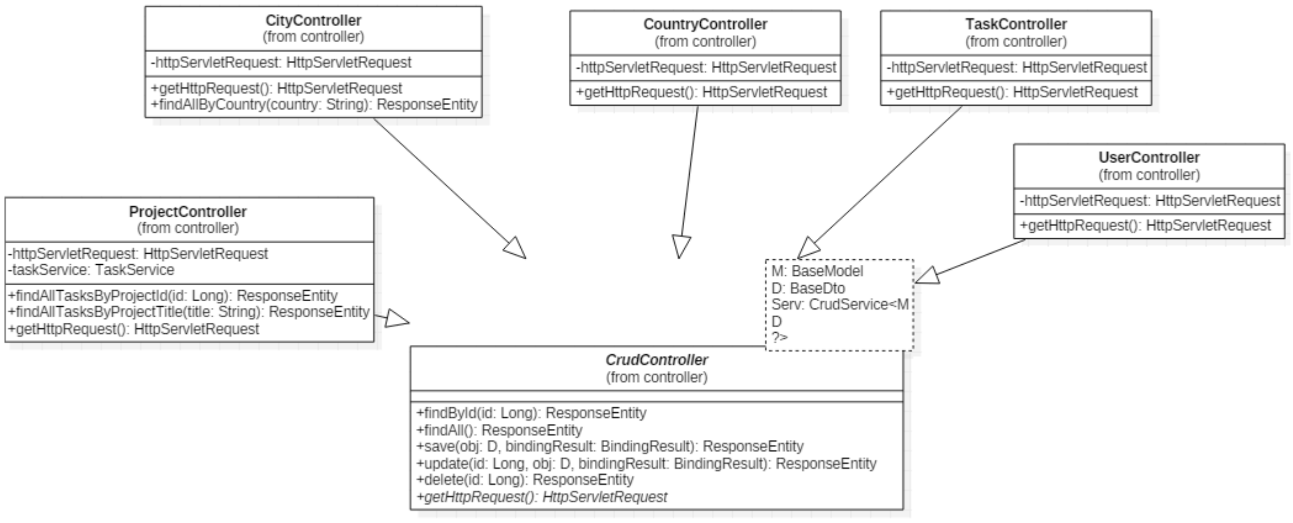
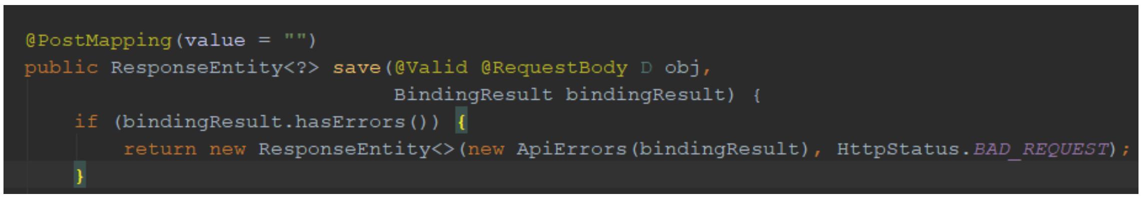

CityController
(from controller)
-httpServletRequest: HttpServletRequest
+getHttpRequest(): HttpServletRequest
+findAllByCountry(country: String): ResponseEntity
CountryController
(from controller)
-httpServletRequest: HttpServletRequest
+getHttpRequest(): HttpServletRequest
TaskController
(from controller)
-httpServletRequest: HttpServletRequest
+getHttpRequest(): HttpServletRequest
UserController
(from controller)
-httpServletRequest: HttpServletRequest
+getHttpRequest(): HttpServletRequest
ProjectController
(from controller)
-httpServletRequest: HttpServletRequest
-taskService: TaskService
+findAllTasksByProjectId(id: Long): ResponseEntity
+findAllTasksByProjectTitle(title: String): ResponseEntity
+getHttpRequest(): HttpServletRequest
M: BaseModel
D: BaseDto
Serv: CrudService<M
D
?>
CrudController
(from controller)
+findById(id: Long): ResponseEntity
+findAll(): ResponseEntity
+save(obj: D, bindingResult: BindingResult): ResponseEntity
+update(id: Long, obj: D, bindingResult: BindingResult): ResponseEntity
+delete(id: Long): ResponseEntity
+getHttpRequest(): HttpServletRequest

Figure 5.22: Controllers

For example the `findAll()` method of the `CrudController` is a GET API method, meaning that an HTTP body doesn't need to be sent. The a Request URL of this method is the name of the entity. For example, for cities, it is `\cities`. When a controller needs additional behaviour, it can override its parent's methods.

Also, JSON objects which are received from the frontend are validated according to the rules imposed on the DTO objects:



```
@PostMapping(value = "")
public ResponseEntity<?> save(@Valid @RequestBody D obj,
                              BindingResult bindingResult) {
    if (bindingResult.hasErrors()) {
        return new ResponseEntity<>(new ApiErrors(bindingResult), HttpStatus.BAD_REQUEST);
    }
```

Figure 5.23: DTO validation in a controller

5.4.2 Models

Every model extends *BaseModel* which holds a primary key. Also, every Model has its corresponding DTO class. When mapping models to DTOs, nested Model objects are not being nested inside DTOs, they are flattened instead. This means that if `Request` has `User` as a nested field, in the DTO, we'll have only a String value: `userEmail`.

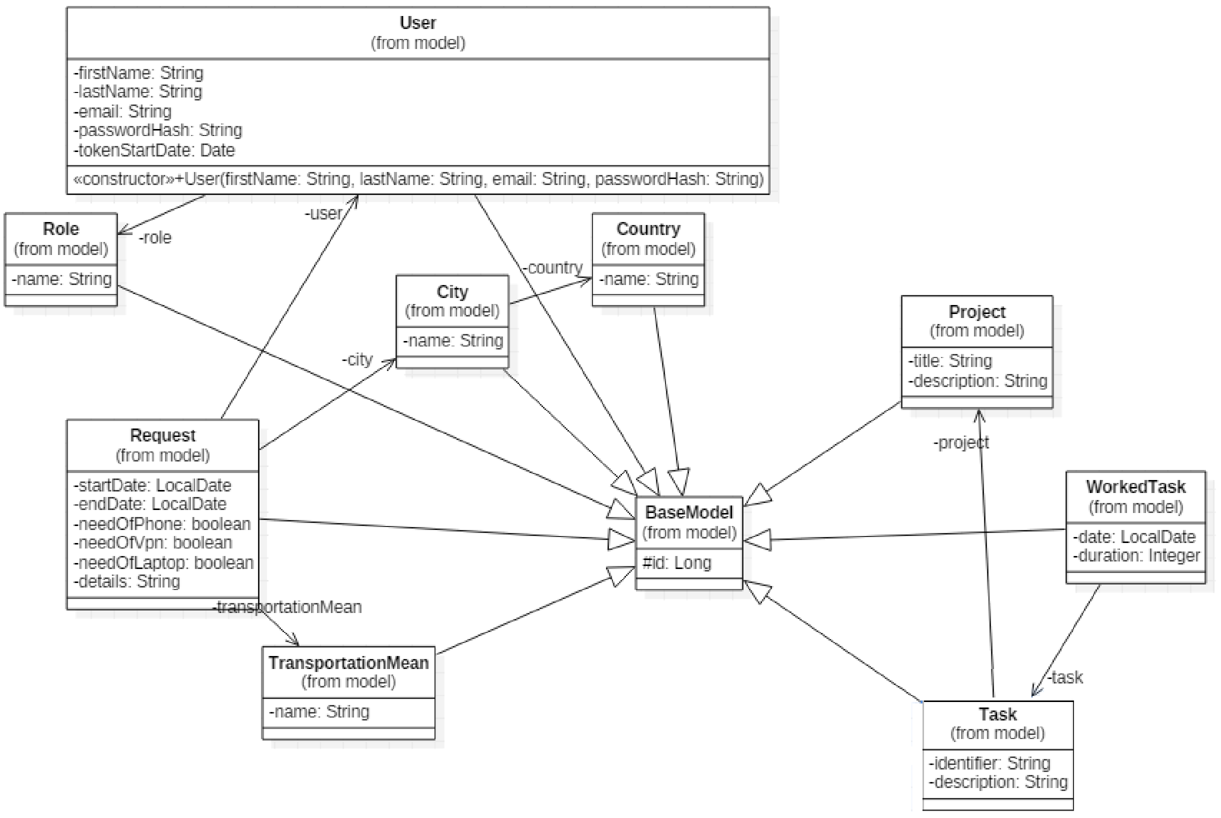


Figure 5.24: Models

5.4.3 Services

The common service class is *CrudService* supporting CRUD operations. Also, every service uses the corresponding repository. A lot of times services need to use not just their corresponding repository but other repositories as well. For example the `UserRepository` is often used by several services because many entities contain user related information such as the email address of the user. For example when a *WorkedTask* is saved, the user who is attached to this worked task needs to be validated.

We can see the *CrudService* being in the center, all other services extend from it.

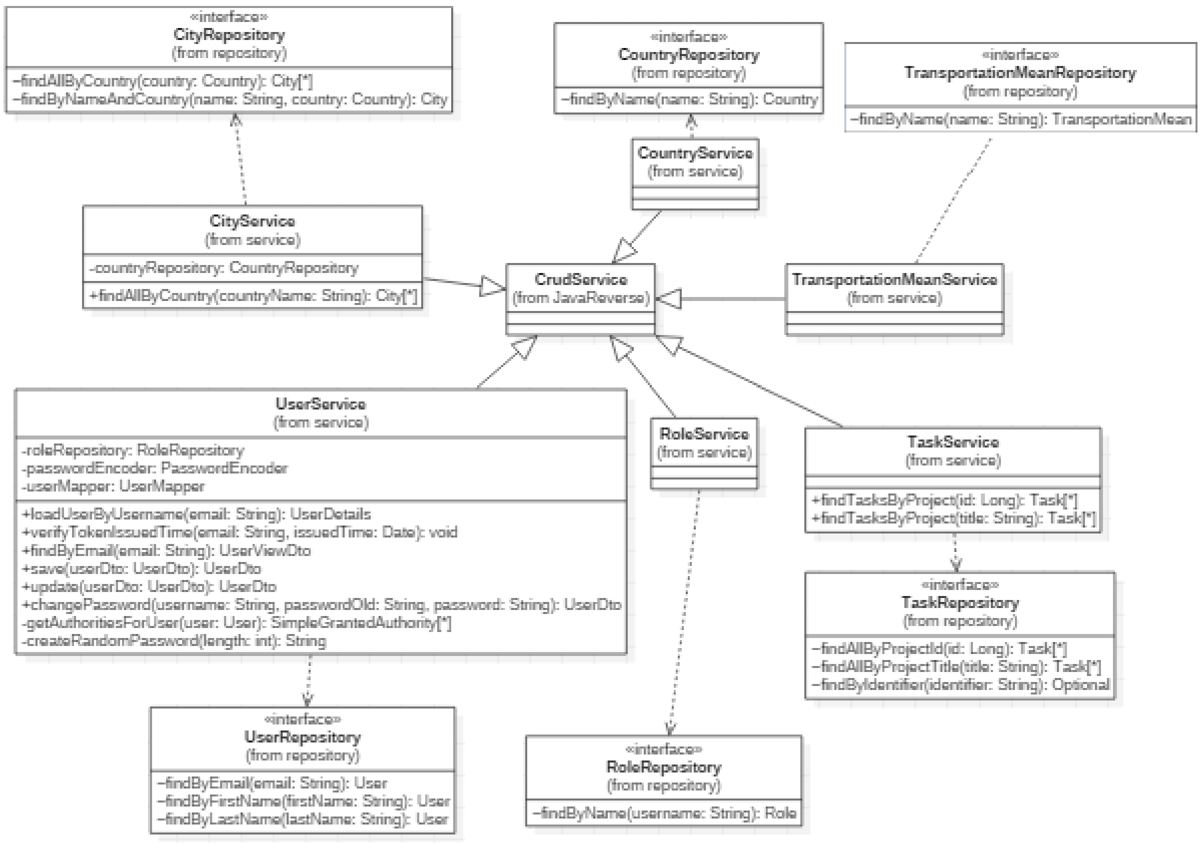

«interface»
CityRepository
(from repository)
~findAllByCountry(country: Country): City[*]
~findByNameAndCountry(name: String, country: Country): City
«interface»
CountryRepository
(from repository)
~findByName(name: String): Country
«interface»
TransportationMeanRepository
(from repository)
~findByName(name: String): TransportationMean
CountryService
(from service)
CityService
(from service)
-countryRepository: CountryRepository
+findAllByCountry(countryName: String): City[*]
CrudService
(from JavaReverse)
TransportationMeanService
(from service)
UserService
(from service)
-roleRepository: RoleRepository
-passwordEncoder: PasswordEncoder
-userMapper: UserMapper
+loadUserByUsername(email: String): UserDetails
+verifyTokenIssuedTime(email: String, issuedTime: Date): void
+findByEmail(email: String): UserViewDto
+save(userDto: UserDto): UserDto
+update(userDto: UserDto): UserDto
+changePassword(username: String, passwordOld: String, password: String): UserDto
-getAuthoritiesForUser(user: User): SimpleGrantedAuthority[*]
-createRandomPassword(length: int): String
RoleService
(from service)
TaskService
(from service)
+findTasksByProject(id: Long): Task[*]
+findTasksByProject(title: String): Task[*]
«interface»
TaskRepository
(from repository)
~findAllByProjectId(id: Long): Task[*]
~findAllByProjectTitle(title: String): Task[*]
~findByIdentifier(identifier: String): Optional
«interface»
UserRepository
(from repository)
~findByEmail(email: String): User
~findByFirstName(firstName: String): User
~findByLastName(lastName: String): User
«interface»
RoleRepository
(from repository)
~findByName(username: String): Role

Figure 5.25: Services

5.4.4 Frontend components

As on the backend, I wanted to reuse as much code as possible on the frontend side, so made three generic components: *ModelForm* which is responsible for opening a pop up which adds new entities, *ModelTable* which shows all the entities in a table and supports delete and edit operations and *ModelTab* which is their parent component.

As previously stated, App is the highest level component, it is like a container, because it can contain any of the lower level components in itself, and it can dynamically change what it contains:

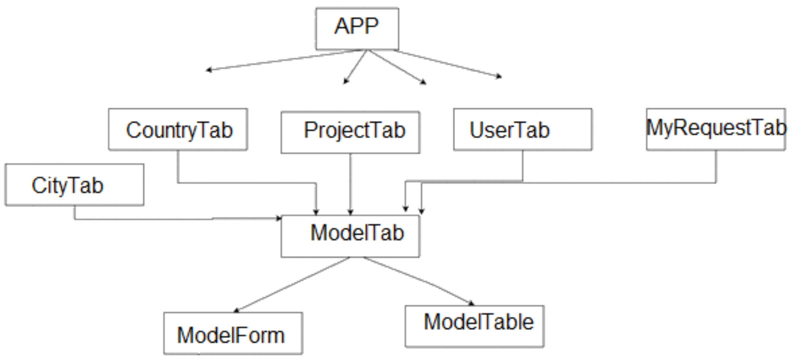


Figure 5.26: Frontend Components

Also, there are some other simple components which don't make use of the generic *ModelTab*, they are standalone components for simple use cases such as changing the password, or viewing profile information:

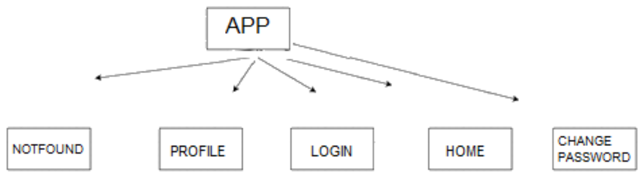


Figure 5.27: Standalone Frontend Components

5.5 Database Design

For users, email, first name, last name are stored. `token_start_date` stores the date from which the user's token is valid. This means that if the user's token is stolen, it cannot be simply deactivated because tokens are not stored in the DB. So, instead, the starting date is lifted, and all the issued tokens become invalid. Also, the UI uses the token to extract the role of the User instead of calling the backend all the time. So, if the role of the user is changed, the user's token needs to be invalidated. In fact, this is automatically done on the backend side. The `users` table is in a one to many relation with the `timecard` and `request` tables because a single user can create many requests and have many timecards but a request or timecard is linked to a single user.

Each user can have a single role. Roles can be: ROLE_ADMIN, ROLE_USER, ROLE_MODERATOR, ROLE_PM for project managers, ROLE_PAYROLL.

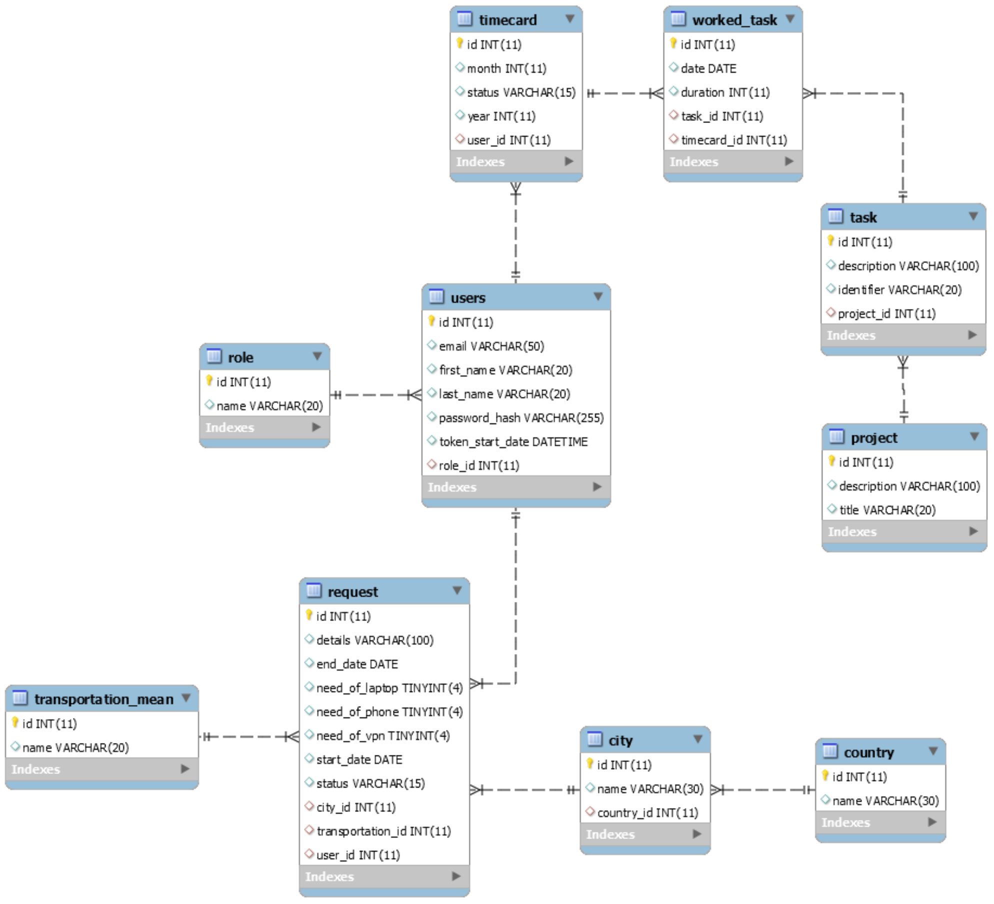


Figure 5.28: MySQL Database Design

The `request` table contains data about a business trip request such as `details`, `start_date`, `end_date`, `status`. `need_of_laptop`, `need_of_vpn` etc. are boolean values and store if the person requesting the business trip request needs these resources. The `request` table is in a many-to-one relation with the `transportation_mean` and `city` tables because a request can have a single city as destination and a single mean of transportation, but cities and transportation means are not limited to a single request. Many requests can go to the same city etc.

Between the `project` and `task` we have a one-to-many relation, because a project has many tasks, a task belongs to a single project.

Between the `timcard` and `task` we have a many-to-many relation, because a a timecard can have many tasks in it and also a task can be present in many timecards. For this, we have a join table named `worked_task`. `worked_task` also stores the duration of the task (how much time the employee spent working on the task) and the date when the employee was working on it.

It is important to note, that tasks are not a one-time thing in the project on which only a single person is working and when it's done, it's done. Instead tasks have a continuum meaning that a task can mean working on a submodule of a big application. In this context the whole application is in a single project but different submodules have different tasks.

Also, it is important to note, that taking vacation is also a task and it has its corresponding project.

The database is in BCNF (Boyce-Codd Normal Form). It respects the first normal form because all the data is atomic. It respects the second normal form because it has no partial dependencies. A partial dependency means that a non-key column is not dependent on the candidate key or not dependent on the whole candidate key if there is a composite one. This is not the case with this design. For example, taking a look at the `project` table, the `description` column is a non-prime attribute and `id` and `title` are both candidate keys. But `description` depends on the whole of both of these keys, so this table is in 2NF. The other tables are as well. The database is also in 3NF because there isn't any transitive dependency. Transitive dependency means that two non-prime columns depend on each other, like if the first name of the user would depend on the last name. This is not the case in our database. And it is in BCNF because it has no functional dependencies of type A -> B where A is not a superkey. Such dependencies would be present if we had some prime-attribute which depends on a non-prime one, like if the title of the project would depend on the description of the project. Of course, this is not the case. So we concluded that our database is in fact a normalized database.

Chapter 6

Testing and Validation

Testing an application is really important. Not just to ensure that it works as expected but also to prevent regression. Regression means that a new feature is added to the application and a previous feature will break. This is where automated tests such as unit testing, integration testing etc. come into play.

6.1 Manual testing

Not everything can be tested with automatic tools. Humans are good at observing bugs which are not detected by tools. So I conducted a throughout manual testing. Some scenarios which were manually tested were:

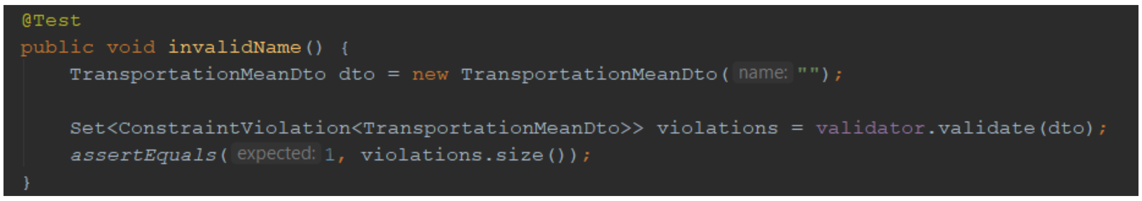
- Refresh the page suddenly in the middle of the interactive tutorial and see if the system still rollbacks all the saved data during the tutorial.
- Try to access a page on the UI which is prohibited for the current role e.g trying to access the Accept/Reject business trip requests page as a regular user.
- Refreshing the page after a successful login to see if the user reconnects to the WebSocket stream and receives the notifications.

6.2 Unit testing

Unit testing means testing a single class or method etc. It is the most often used testing method because of several reasons:

- They are quick to run. Several thousand unit tests can be run in under 10 seconds. This gives a quick feedback to the developer. Every time the user changes something in the code, runs all the unit tests and sees if something is broken.
- They don't need special circumstances to be able to run them. No need for container, no need for database etc.

A sample test which assigns an empty name to the transformation mean and checks if the validation passes:



```
@Test
public void invalidName() {
    TransportationMeanDto dto = new TransportationMeanDto( name: "");

    Set<ConstraintViolation<TransportationMeanDto>> violations = validator.validate(dto);
    assertEquals( expected: 1, violations.size());
}
```

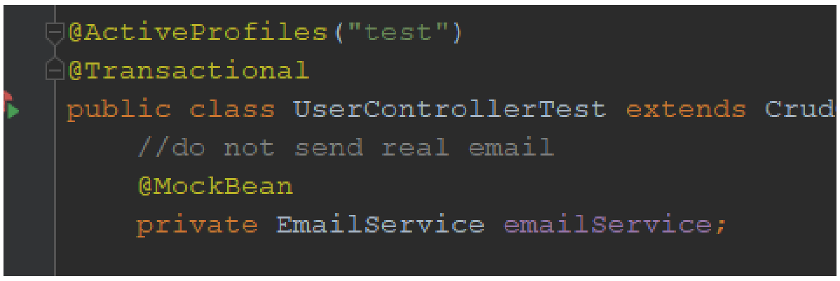
Figure 6.1: Unit testing in Spring

6.3 Mocking

Not everything can be unit tested. Database interactions are hard to test. For these tests, either mocking or integration tests can be used. Mocking means that the methods of a class are not called as they are, instead they are mocked meaning that their functionality is changed. The developer provides a new implementation of the method. For example a database operation which would save an entity can be mocked so that it throws an SQL exception.

Mocking can be also used with integration tests but it is usually used with unit tests. The advantage of mocking is that we can unit test code which would be otherwise impossible.

In this project, the mail service was mocked because it calls the API of a third party. Mocking is the only possibility here:



```
@ActiveProfiles("test")
@Transactional
public class UserControllerTest extends Crud
    //do not send real email
    @MockBean
    private EmailService emailService;
```

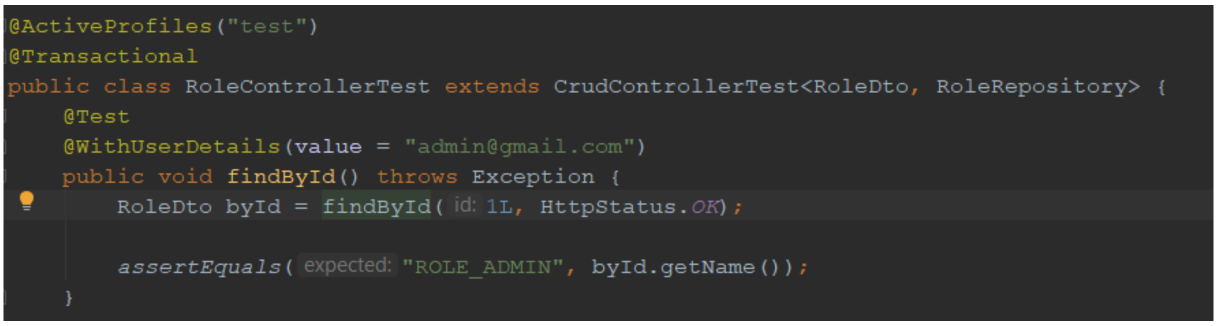
Figure 6.2: Mocking the mail service

6.4 Integration testing

A lot of times, unit testing is not enough. You want to call the database to see if everything behaves correctly. You want to call an API etc.

For this project, I created a base class called `BaseControllerTest` which uses Spring's `MockMvc` class to perform API calls which carry out CRUD operations on entities. `MockMvc` allows us to call real APIs, it just mocks the main servlet of Spring, which means that it doesn't need a container to be able to run. These are still integration tests but with a mocked application servlet.

An integration test which calls an API which returns the role with id 1 and checks if the name of the role is *ROLE_ADMIN*.

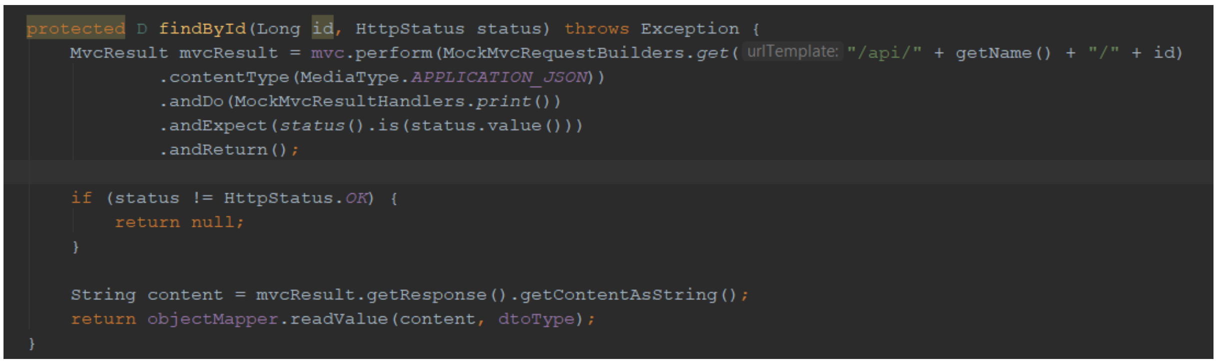


```
@ActiveProfiles("test")
@Transactional
public class RoleControllerTest extends CrudControllerTest<RoleDto, RoleRepository> {
    @Test
    @WithUserDetails(value = "admin@gmail.com")
    public void findById() throws Exception {
        RoleDto byId = findById( id: 1L, HttpStatus.OK);

        assertEquals( expected: "ROLE_ADMIN", byId.getName());
    }
```

Figure 6.3: Spring integration test

The `BaseControllerTest` class contains generic methods like the `findById` method.



```
protected D findById(Long id, HttpStatus status) throws Exception {
    MvcResult mvcResult = mvc.perform(MockMvcRequestBuilders.get( urlTemplate: "/api/" + getName() + "/" + id)
            .contentType(MediaType.APPLICATION_JSON))
            .andDo(MockMvcResultHandlers.print())
            .andExpect(status().is(status.value()))
            .andReturn();

    if (status != HttpStatus.OK) {
        return null;
    }

    String content = mvcResult.getResponse().getContentAsString();
    return objectMapper.readValue(content, dtoType);
}
```

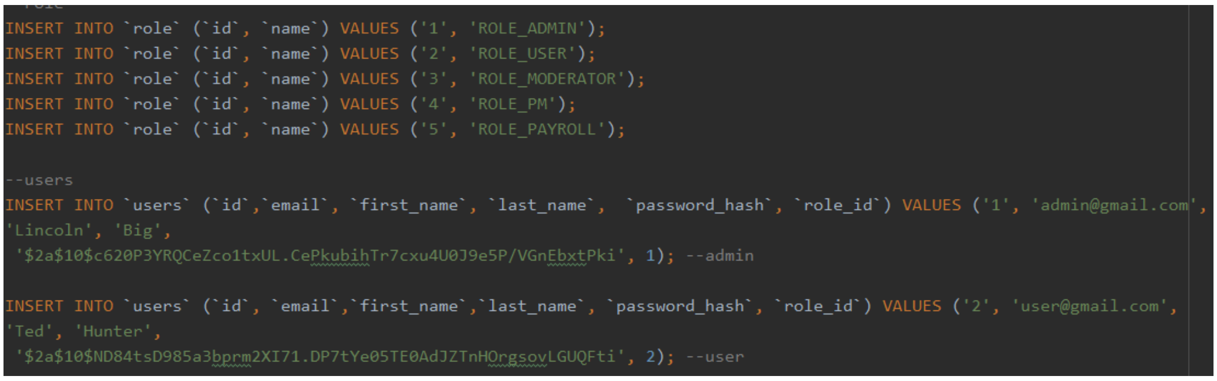
Figure 6.4: Spring's MockMvc class

6.4.1 H2

For integration tests, I have used H2[1] in-memory database because it is faster than a traditional disk based database. Differences are small between MySQL and H2 so what works with H2, should also work with MySQL. Before starting the integration tests, a script named `data.sql` is executed which populates the database with data. No need for manual schema creation because Hibernate takes care of that with the `spring.jpa.hibernate.ddl-auto=create-drop` option.

[1] https://www.h2database.com

H2 was used for all the integration tests. Before starting the integration tests, an in-memory database is created which is identical to the real MySQL database and this database is populated with some dummy data contained in the *data.sql* file:



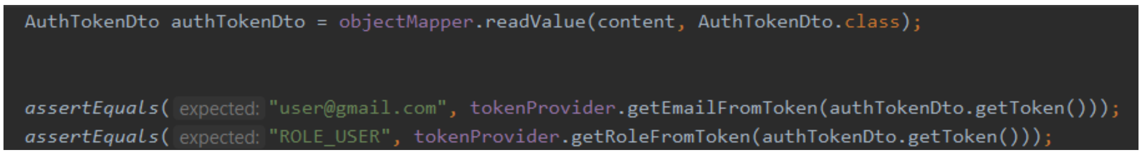
```sql
INSERT INTO `role` (`id`, `name`) VALUES ('1', 'ROLE_ADMIN');
INSERT INTO `role` (`id`, `name`) VALUES ('2', 'ROLE_USER');
INSERT INTO `role` (`id`, `name`) VALUES ('3', 'ROLE_MODERATOR');
INSERT INTO `role` (`id`, `name`) VALUES ('4', 'ROLE_PM');
INSERT INTO `role` (`id`, `name`) VALUES ('5', 'ROLE_PAYROLL');

--users
INSERT INTO `users` (`id`,`email`, `first_name`, `last_name`,  `password_hash`, `role_id`) VALUES ('1', 'admin@gmail.com',
'Lincoln', 'Big',
 '$2a$10$c620P3YRQCeZco1txUL.CePkubihTr7cxu4U0J9e5P/VGnEbxtPki', 1); --admin

INSERT INTO `users` (`id`, `email`,`first_name`,`last_name`, `password_hash`, `role_id`) VALUES ('2', 'user@gmail.com',
'Ted', 'Hunter',
 '$2a$10$ND84tsD985a3bprm2XI71.DP7tYe05TE0AdJZTnHOrgsovLGUQFti', 2); --user
```

Figure 6.5: Populating H2 database with data

Then, the integration tests retrieve data from this dummy data instead of the real database. For example creating a token for a user then checking if the token contains the correct email address and role:



```java
AuthTokenDto authTokenDto = objectMapper.readValue(content, AuthTokenDto.class);


assertEquals( expected: "user@gmail.com", tokenProvider.getEmailFromToken(authTokenDto.getToken()));
assertEquals( expected: "ROLE_USER", tokenProvider.getRoleFromToken(authTokenDto.getToken()));
```

Figure 6.6: Testing Token Generation

Chapter 7

User's manual

7.1 Application installation

Java 11 needs to be installed. Maven version 3+ needs to be installed. Go into the main backend directory (`hr`) and then run `mvn clean install`. This will compile all the .proto files and all the code.

You have to have MySQL version 5, MongoDB 4.0.4 and RabbitMQ 3 installed. If you have Docker[1] installed, you can simply start a MongoDB and RabbitMQ instance with the following commands:

```
mongodb: docker run -d -p 27017-27019:27017-27019 --name mongodb
mongo:4.0.4
rabbitmq: docker run -d -p 5672:5672 -p 15672:15672 --name my-rab rabbitmq:3-
management
```

After this, go to the folder `service` and run
`mvnw spring-boot:run -Drun.arguments="mongodb-update"`.
After that go to the folder `metrics` and run
`mvnw spring-boot:run -Drun.arguments="mongodb-update"`.
After that go to the folder `notification` and run
`mvnw spring-boot:run`.

`-Drun.arguments="mongodb-update"` has to be appended only when the application is first run.

If you are getting an error related to *model* not being found, you may need to manually copy it to Maven's .m2 folder.

The server is started, now let's start the frontend. You need to have Node installed and the npm package manager. Go into the *frontend* directory and run `npm install` first to install all the dependencies needed, then run `npm run serve`. The frontend should be

[1] https://www.docker.com

up and running and calling code from the backend. If the frontend is made available on the employees' machines, then the `API_URL` variable needs to be changed to the server's ip address in the file `src\api\index.js`. The same modification needs to be done in the smartphone project in the `main.js` file.

The files for the smartphone app can either be packaged in an Android/iOS app or directly served from the server with the help of Node (the same way the frontend is served). In this case the application is accessed just as a website and is not installed on the employees' phones. The smartphone app must connect to the company's network if it wants to access the server.

7.2 Application utilization - General information

The user is directed to the home page after a successful login. There is a dropdown called *Action* on the top of the page where available actions can be selected by the user.

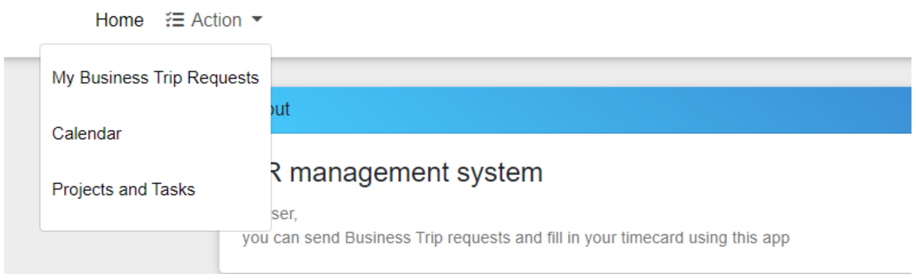

Home
Action
My Business Trip Requests
Calendar
Projects and Tasks
ut
R management system
ser,
you can send Business Trip requests and fill in your timecard using this app

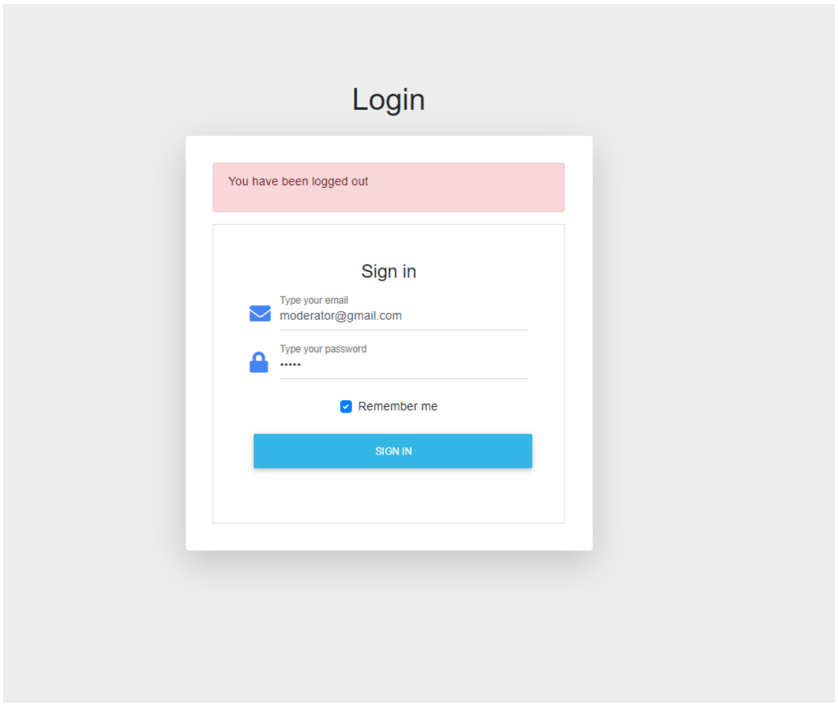
Figure 7.1: Action dropdown

Every table in the application which shows data can be sorted by any column by clicking on one of the headers. Sorting can be done in both ascending or descending order by clicking twice on the header.

7.3 Application utilization - Regular User level

7.3.1 Authentication

When the user first accesses the application, he has to provide an email and password to be able to log in. The remember me option can be selected if the user wants to be logged in for 30 consecutive days instead of only 5 hours.



Login

You have been logged out

Sign in

Type your email
moderator@gmail.com

Type your password
•••••

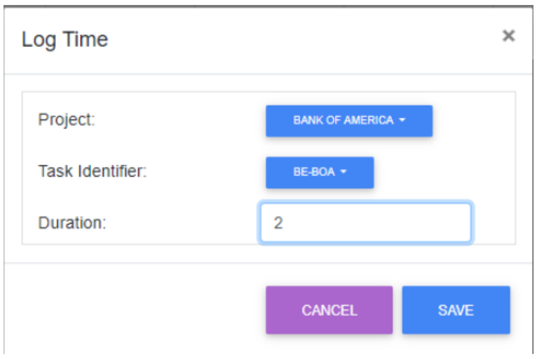
Remember me

SIGN IN

Figure 7.2: Log in page

7.3.2 Filling timecards

Calendar action needs to be selected from the Action dropdown. A timecard appears. To log hours, click on one of the days or select an interval by selecting multiple days and a pop up will appear.



Log Time

Project: BANK OF AMERICA

Task Identifier: BE-BOA

Duration: 2

CANCEL SAVE

Figure 7.3: Timecard - log hours

After selecting the project and task, the new logged hours will be placed in the calendar:

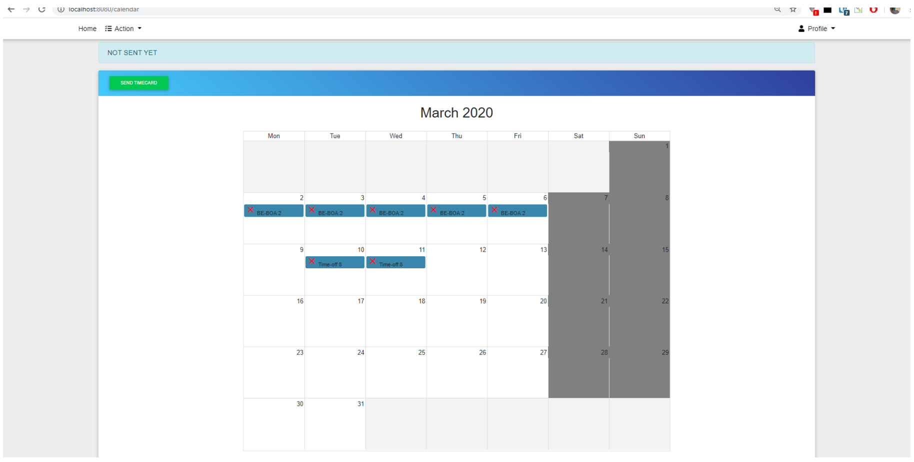
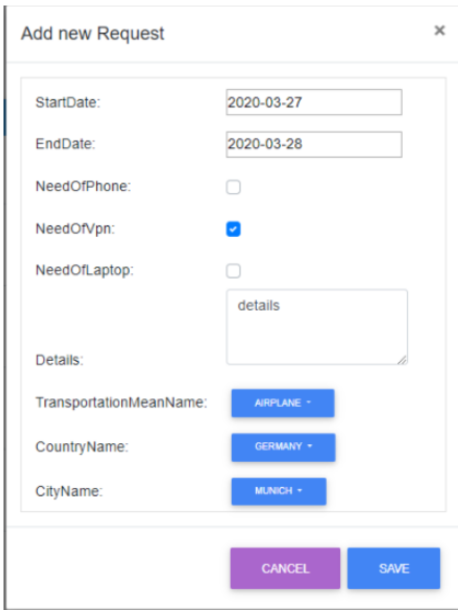


Figure 7.4: Timecard - after logging

7.3.3 Submitting business trip requests

My business Trip Requests action needs to be selected from the Action drop-down. After clicking on the **Add new Request button**, a pop up appears which needs to be completed with valid data and hit **SAVE**.



Add new Request

StartDate:	2020-03-27
EndDate:	2020-03-28
NeedOfPhone:	☐
NeedOfVpn:	✓
NeedOfLaptop:	☐
Details:	details
TransportationMeanName:	AIRPLANE
CountryName:	GERMANY
CityName:	MUNICH

CANCEL SAVE

Figure 7.5: Sending a business trip request

Also, on this page, past requests can be deleted.

7.3.4 Going through the Onboarding process

After opening the smartphone application, you have to log in. After that, you will be presented with the following screen and you can press one of the two buttons corresponding to the tutorial you'd like to view.

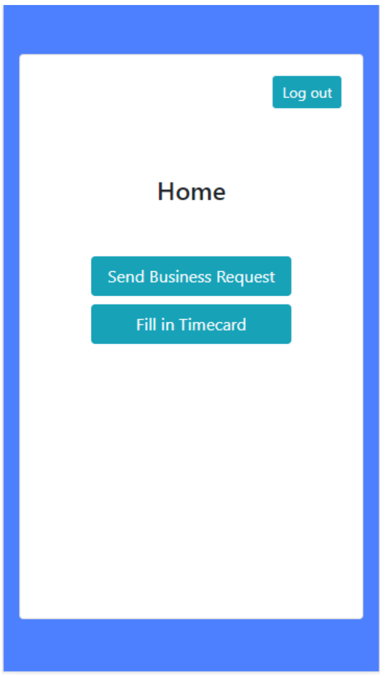


Figure 7.6: Home screen - smartphone app

But you need to have the Desktop app open the whole time, otherwise it won't work. When pressing one of these buttons, the Desktop App becomes alive and starts showing the interactive tutorial.

7.3.5 Change Password

Change Password action needs to be selected from the Profile dropdown on the right hand side.

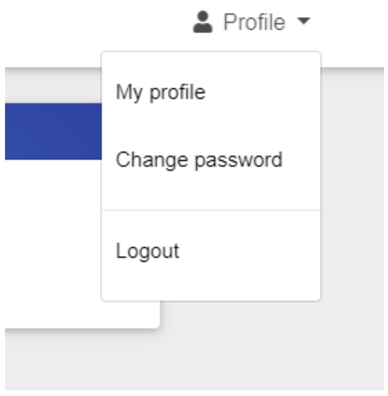
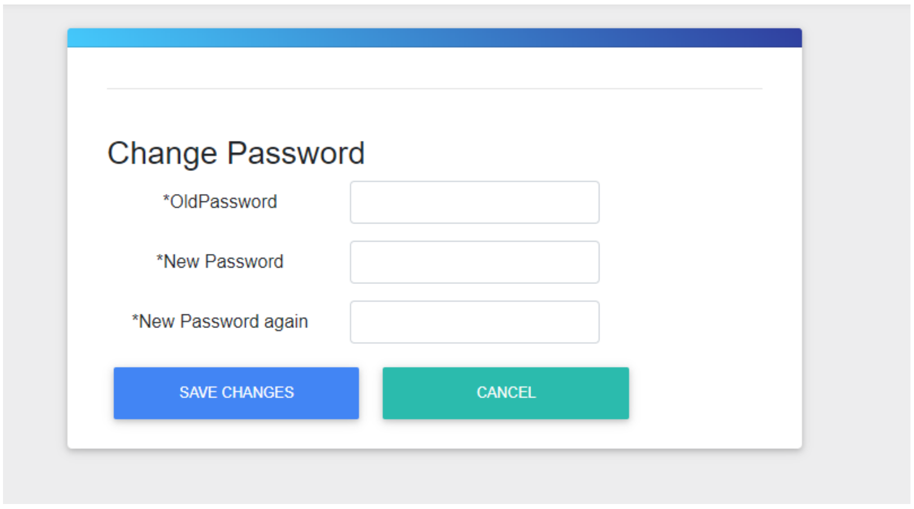


Figure 7.7: Profile dropdown

You will see a page where the current password needs to be introduced, and the new one twice.

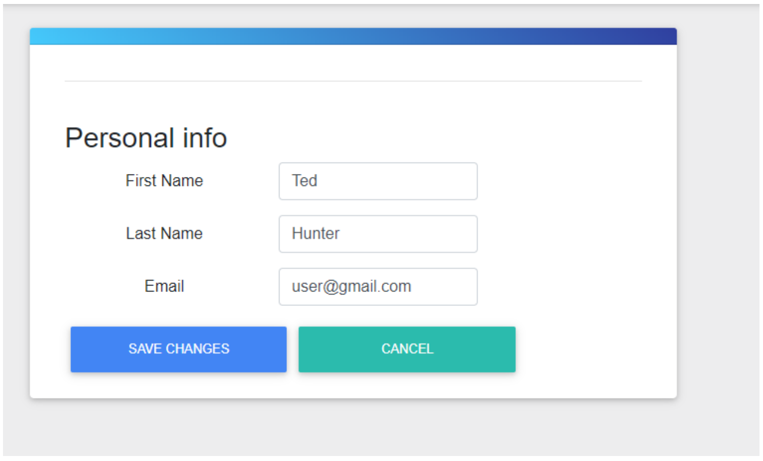

Change Password

*OldPassword	
*New Password	
*New Password again	

Figure 7.8: Change Password page

7.3.6 Changing Public Profile

My Profile action needs to be selected from the Profile dropdown.


Personal info

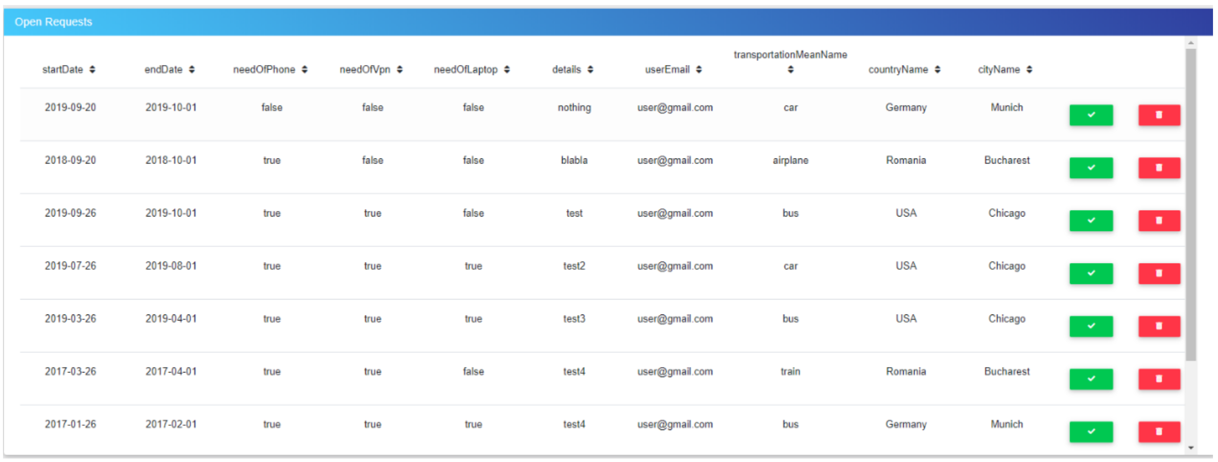
First Name	Ted
Last Name	Hunter
Email	user@gmail.com

Figure 7.9: My Profile page

7.4 Application utilization - Moderator level

7.4.1 Approving requests

Accept/Deny Requests action needs to be selected from the Action dropdown. Two action buttons are present next to each row for request approval/deletion.


Open Requests

startDate	endDate	needOfPhone	needOfVpn	needOfLaptop	details	userEmail	transportationMeanName	countryName	cityName		
2019-09-20	2019-10-01	false	false	false	nothing	user@gmail.com	car	Germany	Munich	✓	🟥
2018-09-20	2018-10-01	true	false	false	blabla	user@gmail.com	airplane	Romania	Bucharest	✓	🟥
2019-09-26	2019-10-01	true	true	false	test	user@gmail.com	bus	USA	Chicago	✓	🟥
2019-07-26	2019-08-01	true	true	true	test2	user@gmail.com	car	USA	Chicago	✓	🟥
2019-03-26	2019-04-01	true	true	true	test3	user@gmail.com	bus	USA	Chicago	✓	🟥
2017-03-26	2017-04-01	true	true	false	test4	user@gmail.com	train	Romania	Bucharest	✓	🟥
2017-01-26	2017-02-01	true	true	true	test4	user@gmail.com	bus	Germany	Munich	✓	🟥

Figure 7.10: Accept/Deny Requests

7.4.2 View Metrics

They appear on the Moderator's Home Page. In total there are seven charts with the following metrics: number of business trip requests based on cities, countries, duration, transportation mean, status, laptop and year/month.

The company can infer valuable information from these charts. For example, the next chart which is based on the number of business trip requests for a given month and year can show spikes in some months when too many people go to a business trip request like in September 2019. These spikes can be analyzed and deduced what went wrong. When a spike happens, too many people leave the company at once (even if just for a sort duration) which means that less people remain to solve internal activities which can be a huge burden on the company. This shouldn't happen.

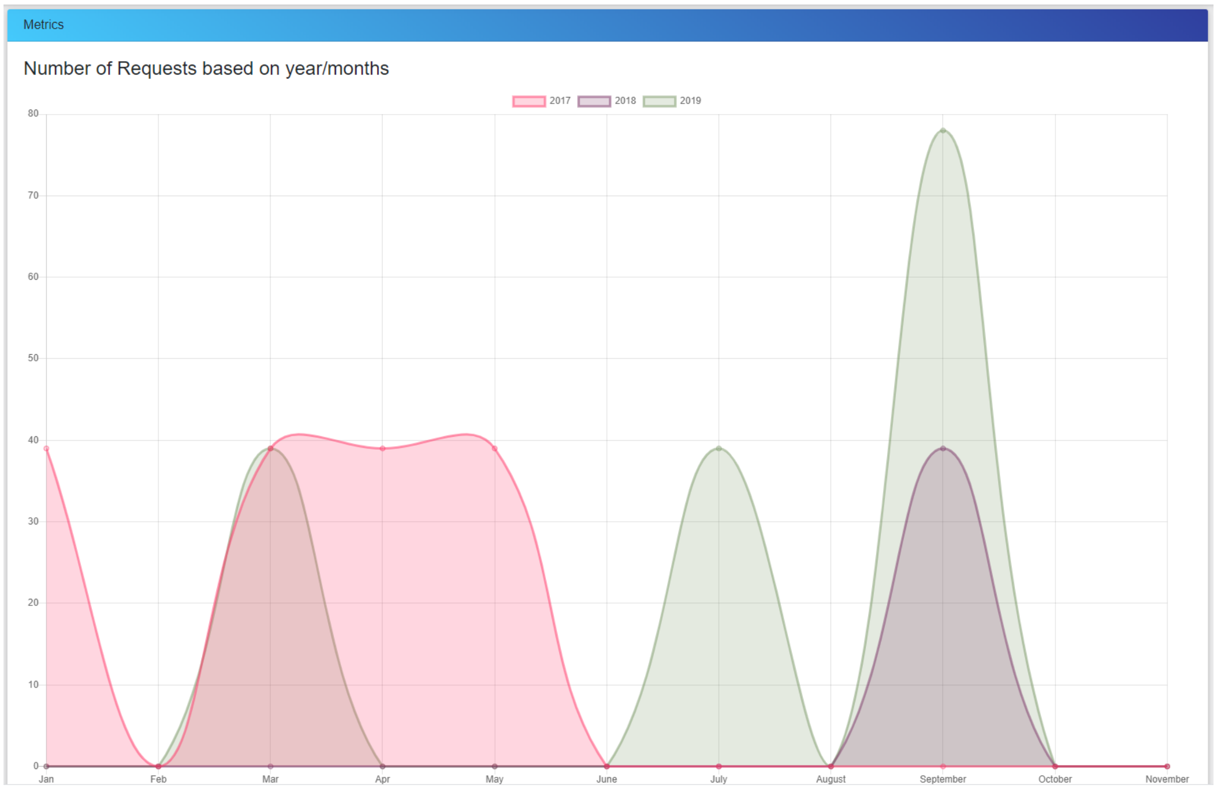
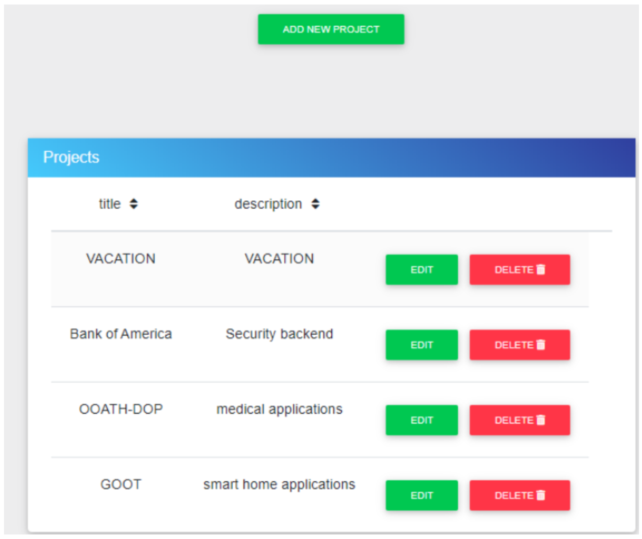


Figure 7.11: Metrics based on year and month

7.5 Application utilization - Administrator level

7.5.1 CRUD operations on basic entities

Modify XYZ action needs to be selected from the Action dropdown where XYZ can be **cities**, **countries**, **transportation**, **projects**. There is an **add new XYZ** button on the top of the page. After opening it, a pop up appears and after completing it, a new XYZ will be saved and added. Also, there are two buttons for editing and deletion.



ADD NEW PROJECT

Projects

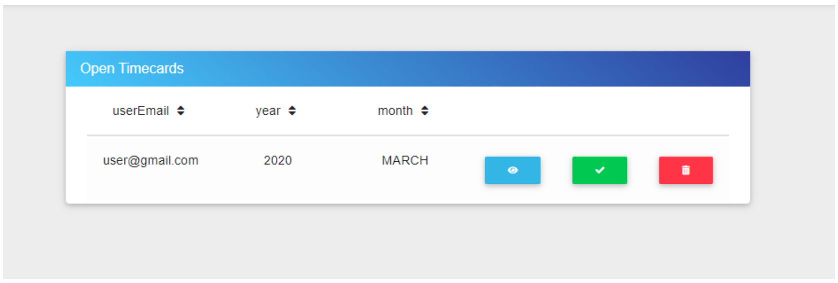
title	description		
VACATION	VACATION	EDIT	DELETE
Bank of America	Security backend	EDIT	DELETE
OOATH-DOP	medical applications	EDIT	DELETE
GOOT	smart home applications	EDIT	DELETE

Figure 7.12: Creating and updating projects

7.6 Application utilization - Project Manager level

7.6.1 Approving Timecards

Timecards action needs to be selected from the Action dropdown.



Open Timecards

userEmail	year	month
user@gmail.com	2020	MARCH

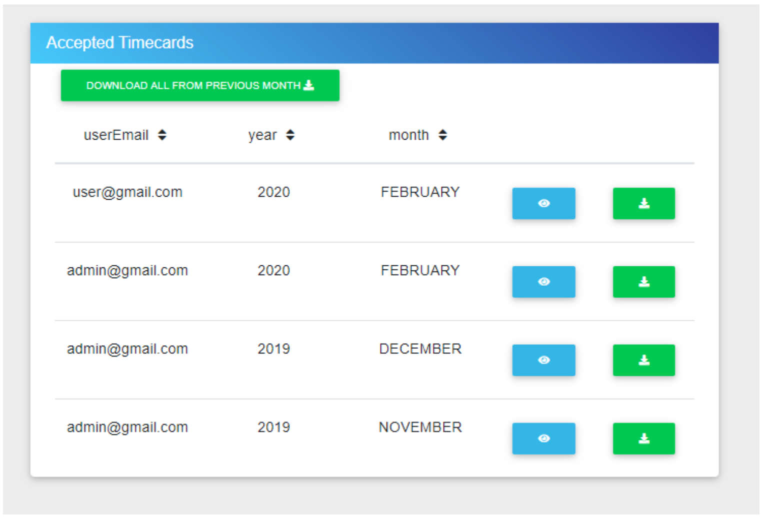
Figure 7.13: Approve/Reject Timecard

Three buttons are available: View/Approve/Reject Timecard. When **View Timecard** is clicked, a pop up appears which shows all the information related to the corresponding timecard.

7.7 Application utilization - Payroll level

7.7.1 Exporting Timecards as PDF

Accepted Timecards action needs to be selected from the Action dropdown. Two buttons are available next to each timecard:



userEmail	year	month		
user@gmail.com	2020	FEBRUARY		
admin@gmail.com	2020	FEBRUARY		
admin@gmail.com	2019	DECEMBER		
admin@gmail.com	2019	NOVEMBER		

Figure 7.14: Exporting Timecard as PDF

The first button shows a single timecard in a calendar, the second button downloads the PDF file with the all the information related to that timecard. Also there is a button with the text **Download all from previous month** which downloads all the timecards from the previous month in different PDF files.

Chapter 8

Conclusions

8.1 Results

We managed to build an easy to use HR system capable to keep track of employee's work, let employees send business trip requests which are stored in the system and later on a person responsible for this can check metrics related to all these requests and make some conclusions like too many people ask for a laptop when they go on a business trip or too many people want to go with a car instead of train. Also, this application is an improvement over other similar HR systems because it doesn't have a confusing UI, it has an interactive onboarding tutorial which helps new employees become familiar with the system and also the UI is reactive which means the pages load quickly and everything is fast and responsive.

8.2 Evaluation

It was hard to find an objective way to evaluate the project. What I did was to ask two persons (who haven't used such systems in their life) to fill their timecards without telling them how to do that. I measured the time taken to finish their task. Every material on the internet and also the documentation from the previous chapter was available to them. The learning time was also calculated into the final time.

Table 8.1: Measured time to complete task

System	**User1**	**User2**
Own system	200s	140s
Sentrifugo [6]	600s	500s
Orange HRM [7]	350s	420s

It can be seen that this system is more efficient when it comes to learning how to use it. Sentrifugo [6] didn't have a very intuitive UI, that's why it performed the worst. Both of these two open source systems had some documentation on the internet, but it wasn't as clear and intuitive as the documentation and interactive tutorial from this system.

8.3 Future developments

One area to improve the Interactive tutorial is in **audio content**. Currently the interactive tutorial only provides a visual part but not an audio one. The component could provide a voice which explains every step of the tutorial. In this way, not just one, but two sensory organs are stimulated, namely the eyes and ears which provides better learning capabilities for the end user.

Some other features which could be added:

- **Stop and Try** feature. The user would have the possibility to stop the tutorial at any point, go back with some steps, or go forward, or simply try to continue alone without the help of the tutorial.
- **Disable Audio** feature. The user has the possibility to disable the sound of the tutorial.
- **Enlarge/Minimize cursor effect** feature. The user can make the cursor which shows the interactive tutorial bigger or smaller depending on his preferences.
- **Notification** feature. The user gets a notification if the tutorial was updated. This usually means that the system itself was updated and the user won't be lost when using the new system.

Further improvements:

- Provide metrics not just for business trip requests but for timecards as well
- Provide more customizable metrics such as being able to select some fields, select some possible values for them or select a range of possible values and see all the requests complying to those values: e.g. selecting all the business trip requests which take between 4 and 7 days and the employee requested a laptop and the request is open
- The UI is not tested currently with automated tests, only manual testing is done. This can be improved by testing it with some automated tools such as Selenium.

Bibliography

[1] A. S. S. Navaz, S. F. A S, C. Prabhadevi, V. Sangeetha, and S. Gopalakrishnan, "Human Resource Management System," *International Organization of Scientific Research - Journal of Computer Engineering*, vol. 8, pp. 62–71, (Impact Factor : 1.686), 09 2013.

[2] G. Dessler, N. D. Cole, and J. Bulmash, *Human resources management in Canada*, 10th ed. Toronto : Pearson Prentice Hall, 2008, ch. 3.

[3] Zubaidah Abdulhakeem Majeed and Sibel Tariyan Özyern, "Implementation of the human resources information systems and comparative study of various platforms," *Universal Journal of Engineering Science*, pp. 66 – 78, 2016.

[4] R. Wayne Mondy and Robert M. Noe, *Human Resource Management.* Pearson, 2005.

[5] Tannenbaum and Scott I., "Human resource information systems: User group implications," *Journal of Systems management*, vol. 41, no. 1, p. 27, 1990.

[6] Sentrifugo, "Free and powerful human resource management system," URL: http://www.sentrifugo.com.

[7] Orange HRM, "An open source human resource management system that covers personnel information management, employee self service, leave, time tracking etc." URL: https://www.orangehrm.com.

[8] HRManagersoftware, "Human resource management software, hr system designed for small and medium sized businesses," http://hrmanagersoftware.com/.

[9] Oracle, "Oracle human resources management system," https://www.oracle.com/assets/018835.pdf.

[10] M. Silva and C. Lima, *The Role of Information Systems in Human Resource Management.* IntechOpen, 10 2018, ch. 7.

[11] Masum, Abdul Kadar Muhammad and Beh, Loo-See and Azad, Md Abul Kalam and Hoque, Kazi, "Intelligent human resource information system (i-hris): a holistic decision support framework for hr excellence." *Int. Arab J. Inf. Technol.*, vol. 15, no. 1, pp. 121–130, 2018.

[12] UNC CHARLOTTE, HR department, “New employee onboarding process,” https://hr.uncc.edu/sites/hr.uncc.edu/files/media/documents/onboarding_ppt.pdf.

[13] Kronos, “New hire momentum: Driving the onboarding experience,” https://interactive.blr.com/Global/FileLib/HRDA_Campaigns/Kronos-New-Hire-Momentum-Onboarding_FINAL.PDF.

[14] Cake HR, “Award-winning hr management software,” URL: https://cake.hr.

[15] BambooHR, “Cloud-based, intuitive, affordable HR system,” URL: https://www.bamboohr.com.

[16] Zoho, “Suite of software systems to run an entire business on,” URL: https://www.zoho.com.

[17] Vaadata, “JWT tokens and security – working principles and use cases,” https://www.vaadata.com/blog/jwt-tokens-and-security-working-principles-and-use-cases.

[18] A. Alinone, “From data push to websockets,” https://lightstreamer.com/share/docs/Lightstreamer_presentation.pdf.

Appendix A

Glossary

Table A.1: Glossary

Term	**Definition**
Ajax	Asynchronous JavaScript and XML - technology which is used to send HTTP requests asynchronously
TCP	Transmission Control protocol, layer 4 protocol, used for reliable packet transfer
PHP	Widely used scripting language of the web
DOM	Document Object Model - tree-like representation of a web UI interface
Maven	Java package management and build automation tool
POJO	Plain Old Java Object - denotes a Java object which contains only fields and basic getters and setters
SPA	Single Page Application - type of Web User Interface where there is a single page with many components, and these components are only rendered when needed
Hashing	Generating a value from text using a formula, used for encryption
Local storage	Area of a modern browser capable of storing key-value pairs
IDL	Interface description language, language used to describe a software interface
ORM	Object Relational Mapper - tool used to map object oriented classes onto database tables
DO	Data Object - object oriented object which maps to a database table
DTO	Data Transfer object - object which is an external representation of a DO object, it is returned to the client

Term	**Definition**
CRUD	Create, Read, Update, Delete operations on entities
Backend	Part of the application running on the server, not visible to the end user
Frontend	Part of the application running on the client's machine
HTTP	Hypertext Transfer Protocol - Application Layer Protocol used for high level data communication between applications
API	Application Programming Interface - it is usually loosely used, means something which has an interface and specification

Appendix B

List of Figures

Appendix C

List of Tables