System Specification

General Information

Responsibilities
The team is split up in two responsibilities:

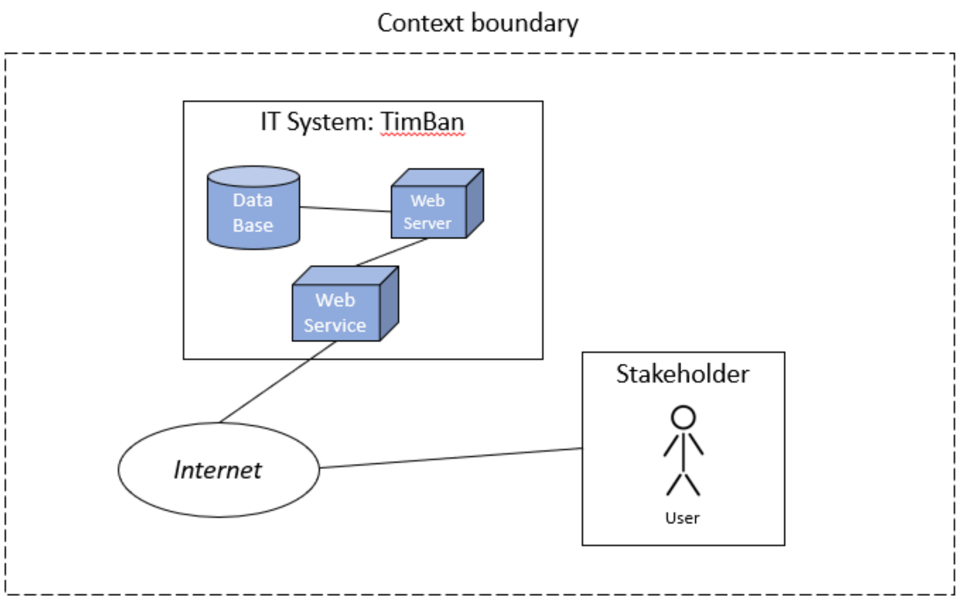
- **Frontend:**
    - Antonio, Lars
- **Backend:**
    - Mathis, Sven

Implementation
The project basis is based on the Java Framework Spring Boot. Thus, a lot of conditions are defined. Such conditions are the naming convention and the structure of the endpoints.

To further define a clear structure, REST is applied to the structure.

Components in the System

The illustration below shows the context diagram of our application "TimBan". The application itself is in the system boundary and everything in the system context boundary has an influence or is influenced by the system.



Client-Server Communication

How is the communication?
As soon as a user accesses TimBan with a web browser (the client) for recording the work time, the client automatically initiates a request to TimBan's server. The credentials of the user are being checked and may be stored in a database, and the web server accesses the database server as a client.

An application server interprets the returned data by applying the business logic, which has been established by Timban, and provides the output to the web server. Finally, the web server returns the result to the client web browser and the result will be displayed.

Based on REST

The following table shows the definition of the endpoints that are accessed in regards to the REST standards.
If more models (e.g. Project) are added, they follow the same principle as the structure shown below. The main idea is that the simple CRUD operations are supported. If needed more endpoints are defined and documented.

Endpoint	Method	Description
/users	GET	Access users
/users/create	POST	Create a user
/users/{user}	GET	Display specific users
/users/update/{user}	PUT	Update a specific user
/users/delete/{user}	DELETE	Delete a specific user
/user/login	POST	Login of the user
/user/logout	POST	Logout of the user
/records	GET	Display time records (server returns all entries from the current year)

/records?orderby=day	GET	Display time records for a specific day (rendered on client side)
/records?orderby=week	GET	Display time records for a specific week (rendered on client side)
/records?orderby=month	GET	Display time records for a specific month (rendered on client side)
/records?orderby=year	GET	Display time records for a specific year (rendered on client side)

Define if JSON, Strings, *Objects*…

The answers of the server are sent with JSON or Strings, depending on what is requested. If an Object is requested the answers of the server are sent with JSON and if single information is requested the answers of the server are sent with Strings.

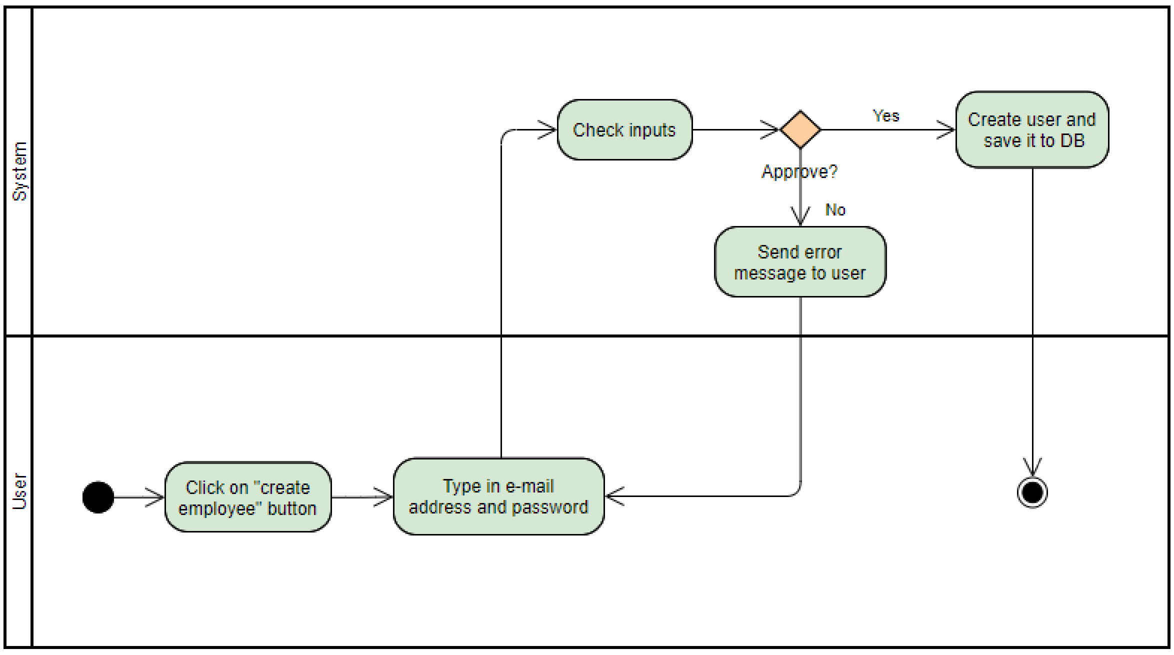
What messages are exchanged at what time (sequence of messages)

User-Action	**Response**
“Create User” Submit Button	Notification, if the user has been created successfully or not. Response as a String. User list updated and responded as JSON.
Show time records	Time recording of specific user are responded as JSON.
Change filter of time records to specific year.	Time recordings of specific user and year are responded as JSON.

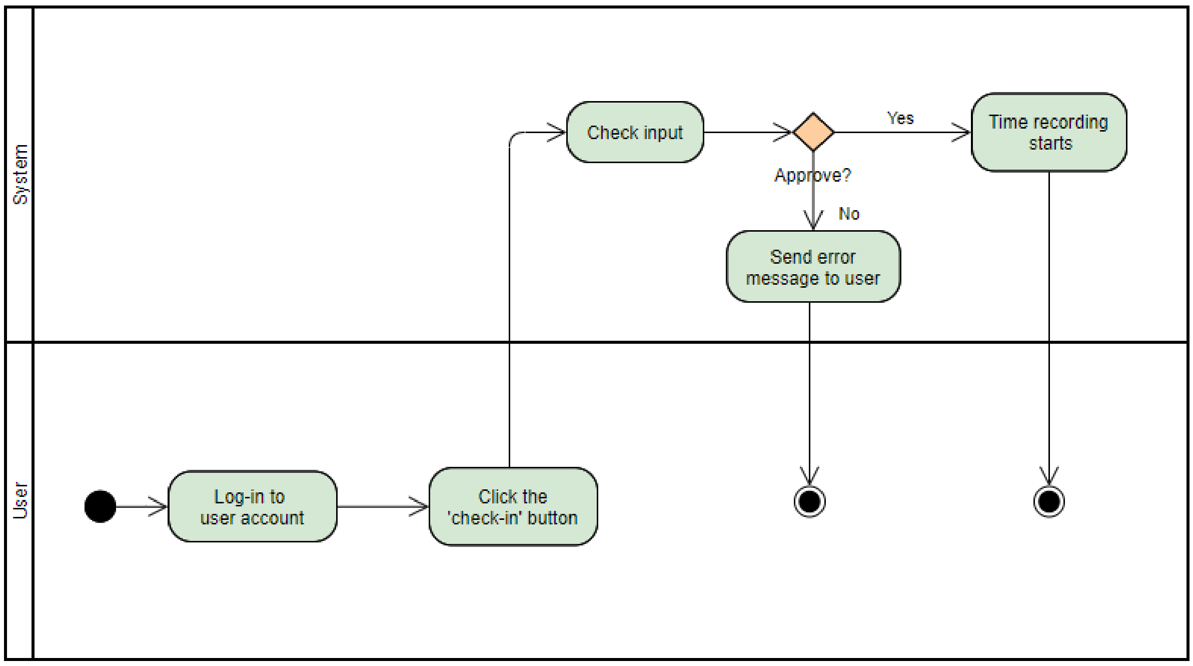
Preconditions for the client

The table and the activity diagrams below show the preconditions for different events of the TimBan application. Each precondition must be met for an event to be executed successfully.

Event	**Preconditions**
Login (to existing account)	1. Account for the used email address exists. 2. Password for account is correct.
Check-In (start recording)	1. Logged in to the user account. 2. Not checked-in yet.
Check-out (stop recording)	1. Logged in to the user account. 2. Account has to be checked-in before.
Edit “time recording”	1. Logged in to the user account. 2. Time recording is not older than 7 days.
Create employee	1. Logged in to the admin account. 2. E-mail address for the new employee is correct and not used yet. 3. Password has min. 5 characters.
Change email address	1. Logged in to the user account. 2. New email address is correct and not used yet.
Change password	1. Logged in to the user account. 2. New email address is correct and not used yet.


System
User
Check inputs
Yes
Create user and save it to DB
Approve?
No
Send error message to user
Click on "create employee" button
Type in e-mail address and password

Create employee



Check-in

Main classes (Models)

In the following table, the minimal required classes are collected.

Entity / Model name	**Attributes**	**Description**
User (Model)	- id: Long - UserName : String - Email: String - Password: String - IsAdmin: boolean - WeeklyHours: Long - createdOn: Timestamp - deletedOn: Timestamp	IsAdmin defines, if the user is an admin user who can create other users and correct/adjust TimeRecord. WeeklyHours define hours to work - is set by admin; default: the companyHours are taken.
TimeRecord (Model)	- id: Long - UserId: Long (foreignKey) - StartRecording: boolean - StopRecording: boolean - TimeStamp: Timestamp	Either StartRecording or StopRecording can be true.
CompanyConfig	- CompanyName: String - CompanyHours: int	In this class, the company configuration is loaded and saved if modified.

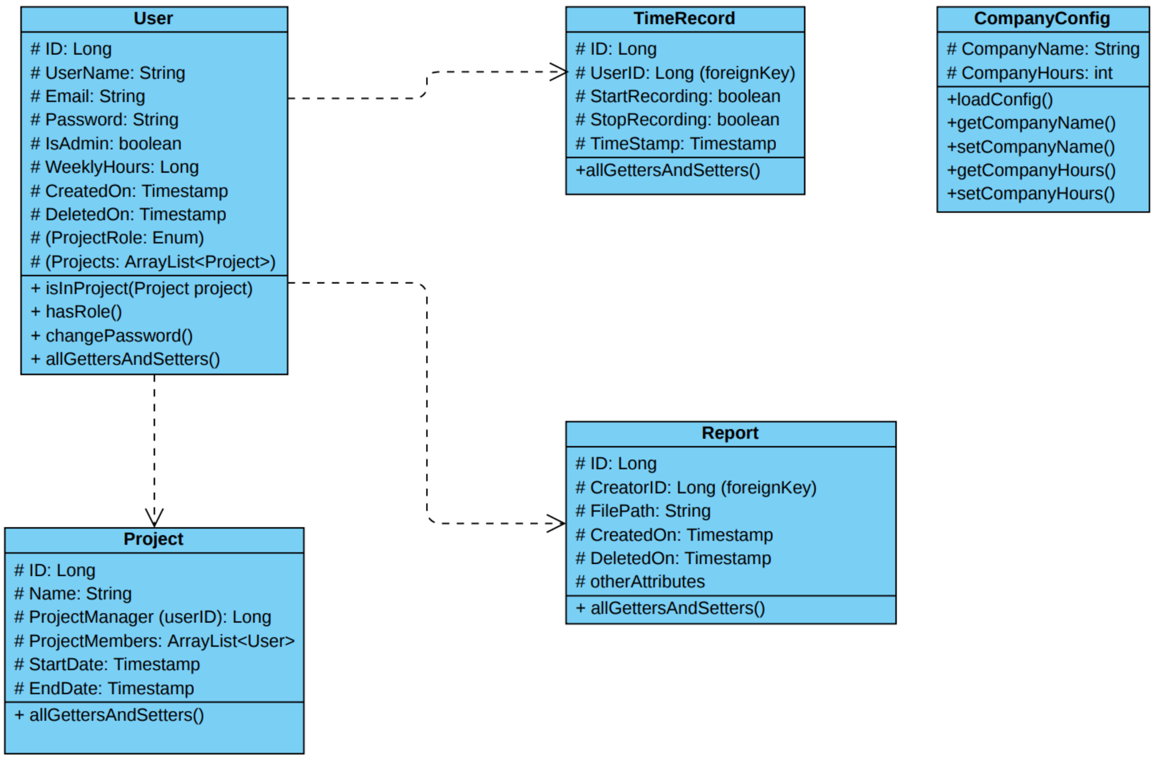
The following table summarizes roughly the Classes, respectively the adjustments that need to be done to existing classes, that are relevant for a further enhancement of the system.
Projects are implemented in the system.

Entity / Model name	**Attributes**	**Description**
User (Model)	- ProjectRole: Enum - Projects: ArrayList<Project>	Existing model gets adjusted. Projects hold all projects where the user is involved in
Project (Model)	- id: Long - Name: String - ProjectManager (userID): Long (foreignKey) - ProjectMembers: ArrayList<User> - StartDate: Timestamp - EndDate: Timestamp (nullable)	

Report (Model)	- id: Long - creatorId: Long (foreignKey) - filePath: String - createdOn: Timestamp - deletedOn: Timestamp	

Class Diagram

The class diagram shows the relations of the different classes. Attributes in parentheses show the adjustments if the project is added to the system.

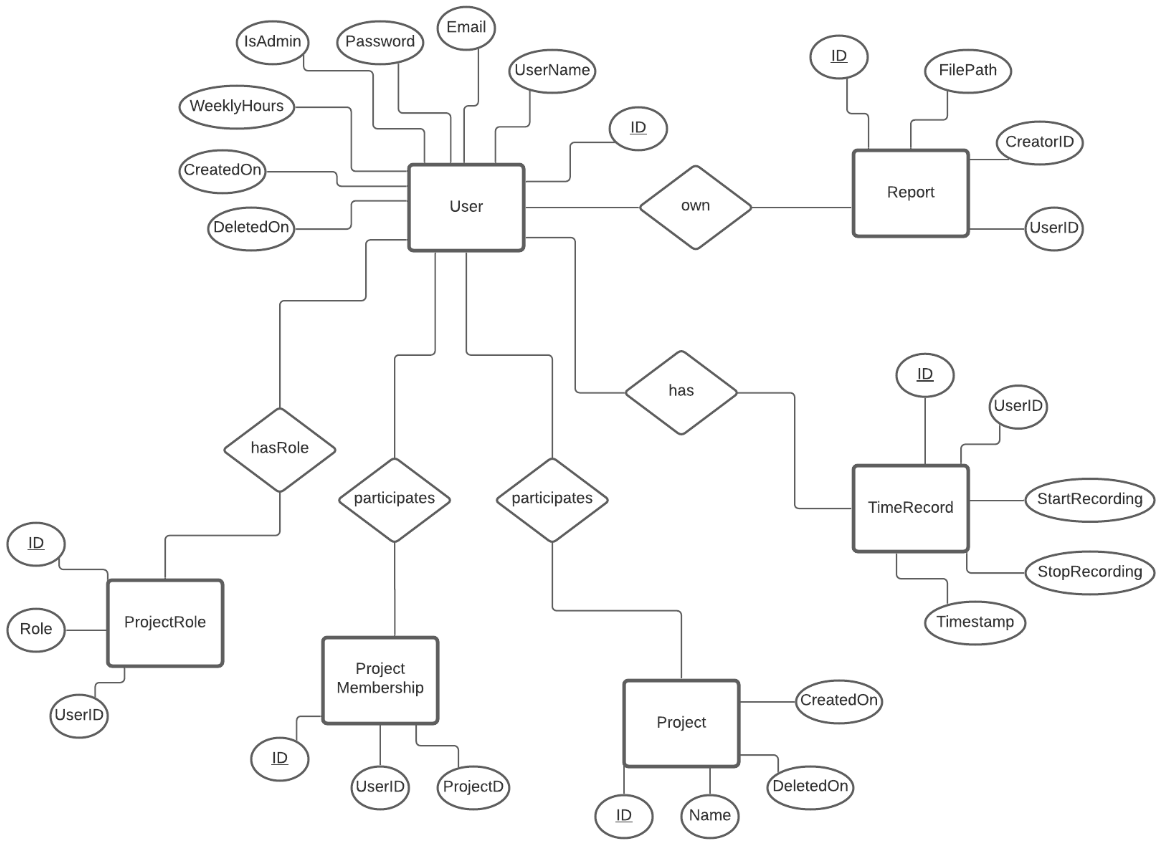


Data Structures, Database and ERD

Database
Due to the application of a PostgreSQL database in a recent project, we decided to use such a database.

Thanks to the SpringBoot framework, the data access can be done using the Spring Data JPA. Therefore no difficult SQL queries have to be declared. If needed, query statements can be defined manually.

ERD
According to the class diagram, the entity relationship diagram (ERD) can be designed. In the diagram the table Company Config is not displayed as this is not that relevant in regards of relationships to other relations. The most important entity is the 'User'.
The entities 'ProjectRole', 'ProjectMembership', and 'Project' are only relevant in the further development step when projects are added to the system.


IsAdmin
Password
Email
UserName
WeeklyHours
ID
CreatedOn
DeletedOn
User
own
ID
FilePath
CreatorID
Report
UserID
has
hasRole
participates
participates
ID
UserID
TimeRecord
StartRecording
StopRecording
Timestamp
ID
Role
ProjectRole
UserID
Project Membership
ID
UserID
ProjectD
Project
CreatedOn
DeletedOn
ID
Name