University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Internal Payroll Support

KIV/ASWI

June 11, 2019

Lukáš Černý
Hung Hoang
Václav Jirák
Dominik Poch

Contents

1 Introduction

The target of this project was to create an application which maintains employees' time off. The application should help make a situation in a company more transparent and provide a digital recordings of employees' overtimes and sick days in a comfort and accessible way.

2 User documentation

2.1 Main screen

Main screen is different for employer and employee. The employer's main screen is shown in Figure 1. You can see five various panels which give the employer all important information in one place. The first panel is called User approval. It shows a list of all users that need the approval. When the user is approved, he is allowed to log into the application. Under the first panel there is the Vacation approval. It is a table of every vacation users want to take. Employers' time off is not shown in the table because it is accepted automatically. The employer can accept or disallow the vacation. All described panels were employer specific but rest of the main screen is shown to the employee too. The rest consists of three panels, My oncoming vacation, Vacation remaining and a calendar. Remaining vacations contains vacations (overtime) and sick days. Those provide information about the selected time off a user can take. If he does the taken sick day or vacation is displayed in the My oncoming vacation box. There is basic information about the selected time off with a possibility to cancel the vacation or sick day. The main component of the screen is the mentioned calendar that allows the user to select the desired vacation or sick day. A pop up menu with settings of time off shows up when a day is selected. The sick day can be taken just for the whole day but the menu provides some input fields for starting and ending time when the vacation is selected.

There are more parts of the screen which has not been described in the previous paragraph because they are used through the whole application. A menu on the left side of the screen and a picture and a name of the logged user in the upper right corner. The menu has two different states again, one for the employer and employee. The employee does not see the Employees item whose content is described closer in the Section 2.2. Localisation supports the Czech and English language. They can be swapped with the little flag next to the picture of the logged user. The image and name serve its purpose because they are used for a change of a notification of an incoming reset of remaining vacations and sick days.

2.2 Employees

The Employees page contains just one component - a list of users. Nevertheless it provides multiple functions. Three buttons, that allows the employer change default settings, import users' data from .xls file format and export users to pdf, are placed next to a title of a table. Default settings consist of the notification and number of sick days. Import and export has not been fully implemented yet so it is working only provisionally.

For each row of the table (user) exists a possibility to show its full profile, example in Figure 3, or edit a role and the number of sick days and vacations. That functionality is provided by small buttons on the each row of the table.

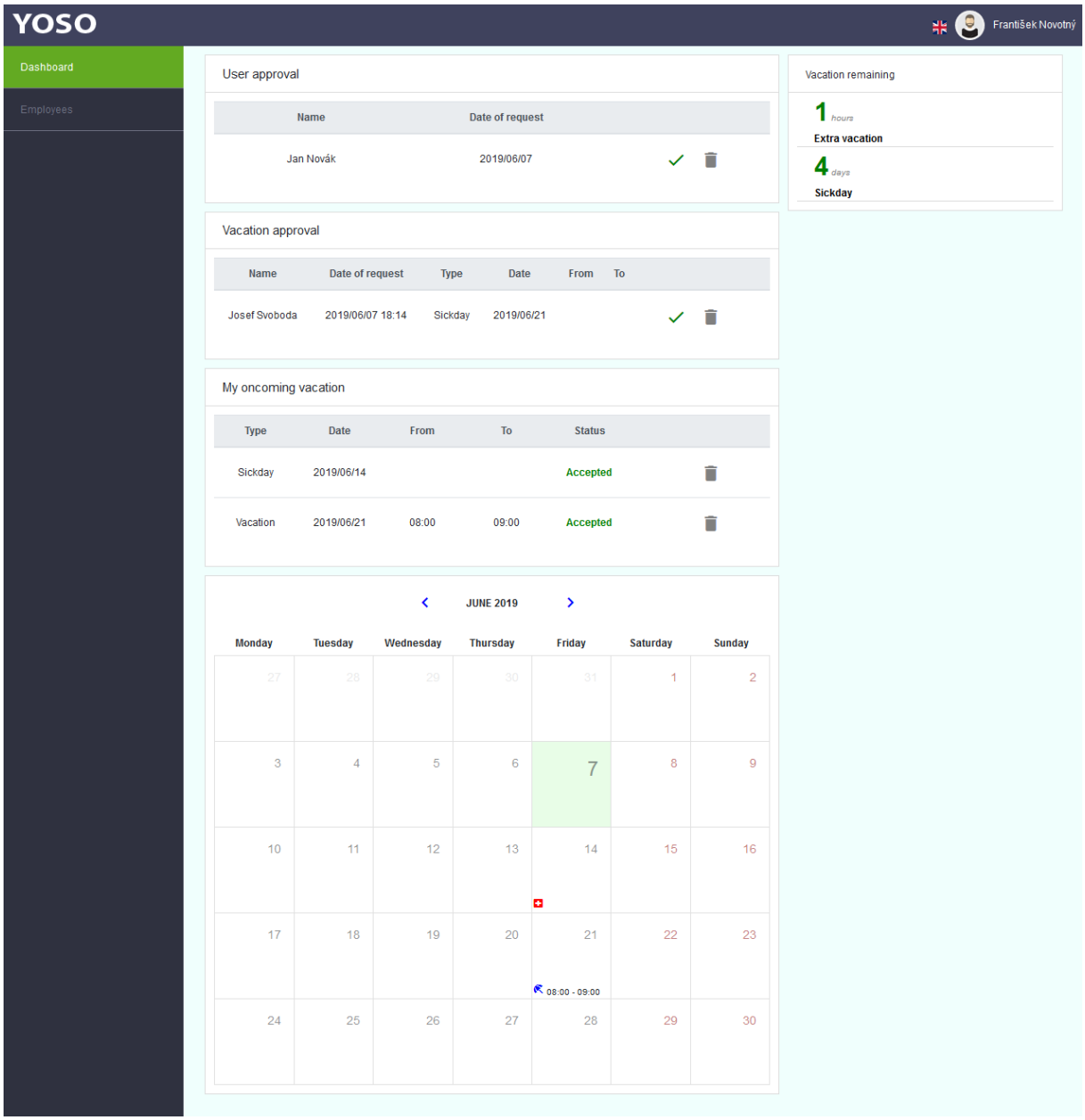

Figure 1: The employer's main screen

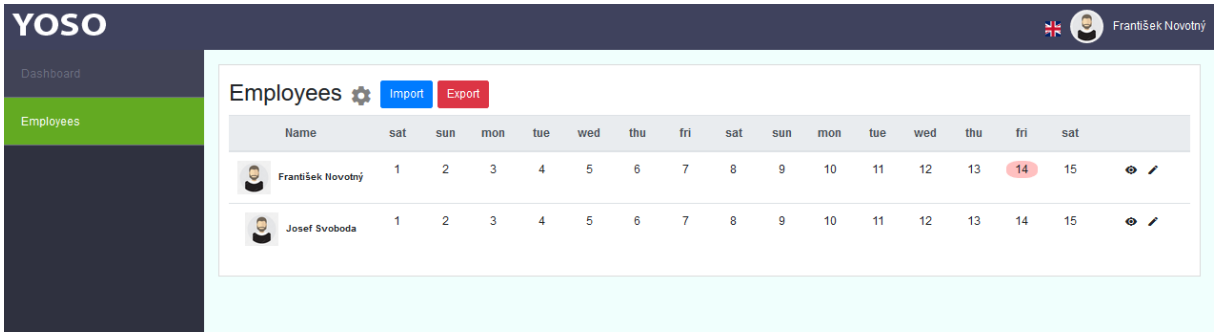

Figure 2: The list of employees

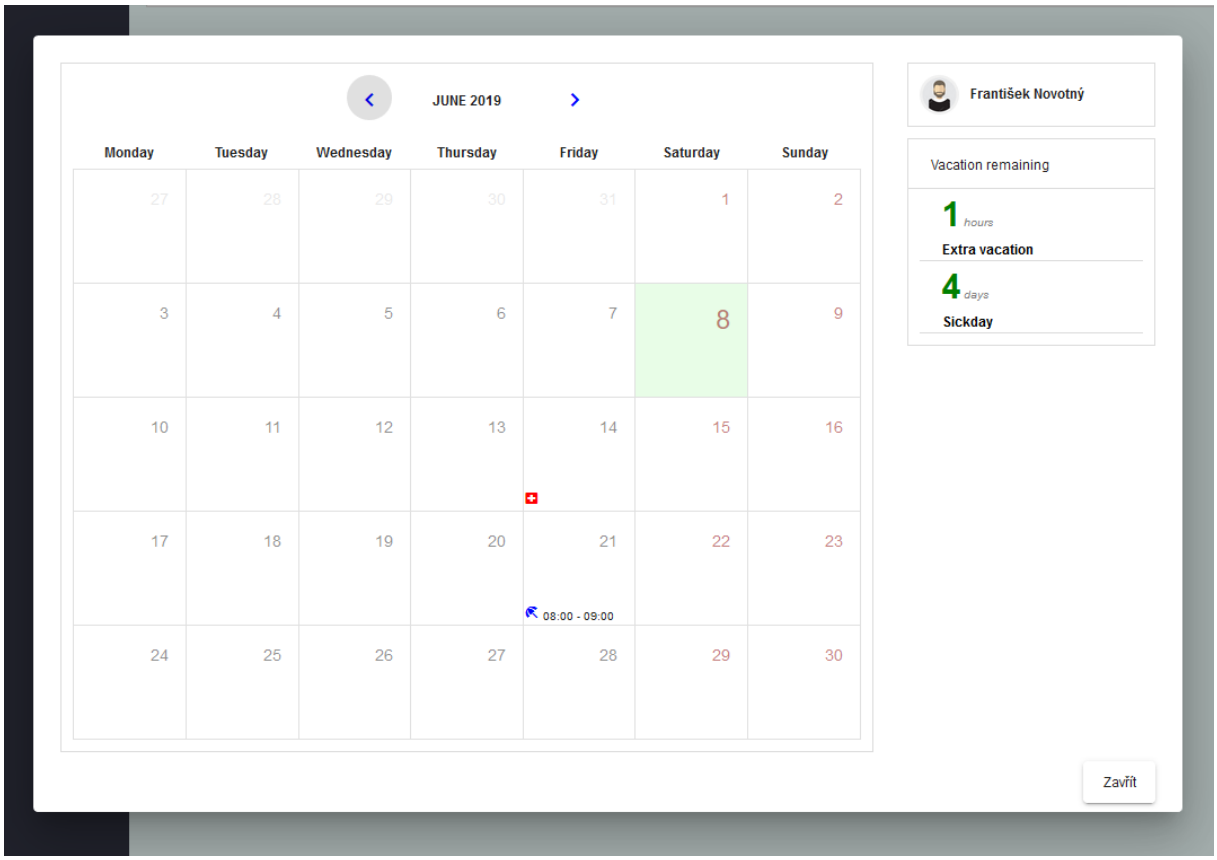

Figure 3: The full user profile

3 Technical documentation

3.1 Architecture

3.1.1 High level

The Figure 4 shows a default design of an architecture of the application. There is visible separation of a frontend and backend of the application and a method of their communication.

The frontend of the application runs in the Node.js[1] a JavaScript runtime environment and is developed in the TypeScript platform Angular[2]. The server, backend, is implemented in Java with Spring Boot[3] framework and is connected to the MariaDB[4] database. The application is deployed on the Docker[5] which makes the deployment easier. The communication between the fronend and backend is controlled by the REST API and database queries are handled by the JDBC. The development has been going on GitHub[6]. A summary of used technologies is located in Figure 5.

[1]https://nodejs.org/en/
[2]https://angular.io/
[3]https://spring.io/projects/spring-boot
[4]https://mariadb.org/
[5]https://www.docker.com/
[6]https://github.com/

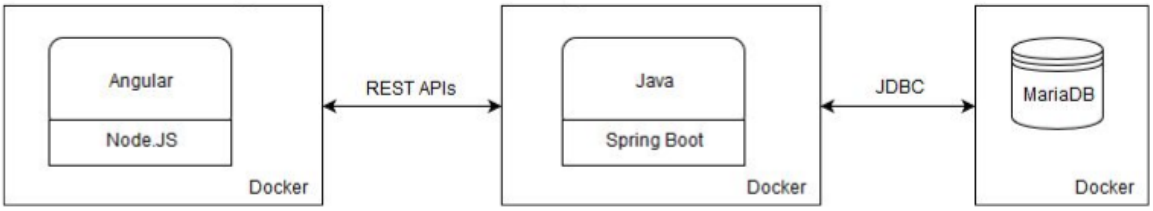

Figure 4: The high level architecture

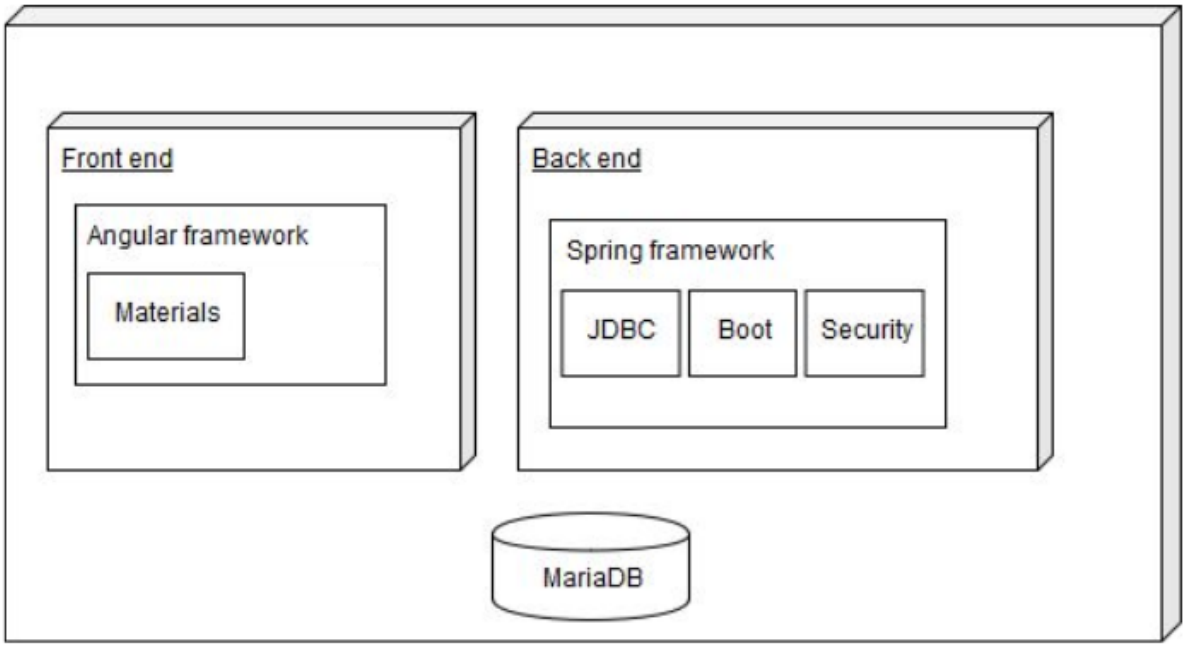

Figure 5: The used technologies

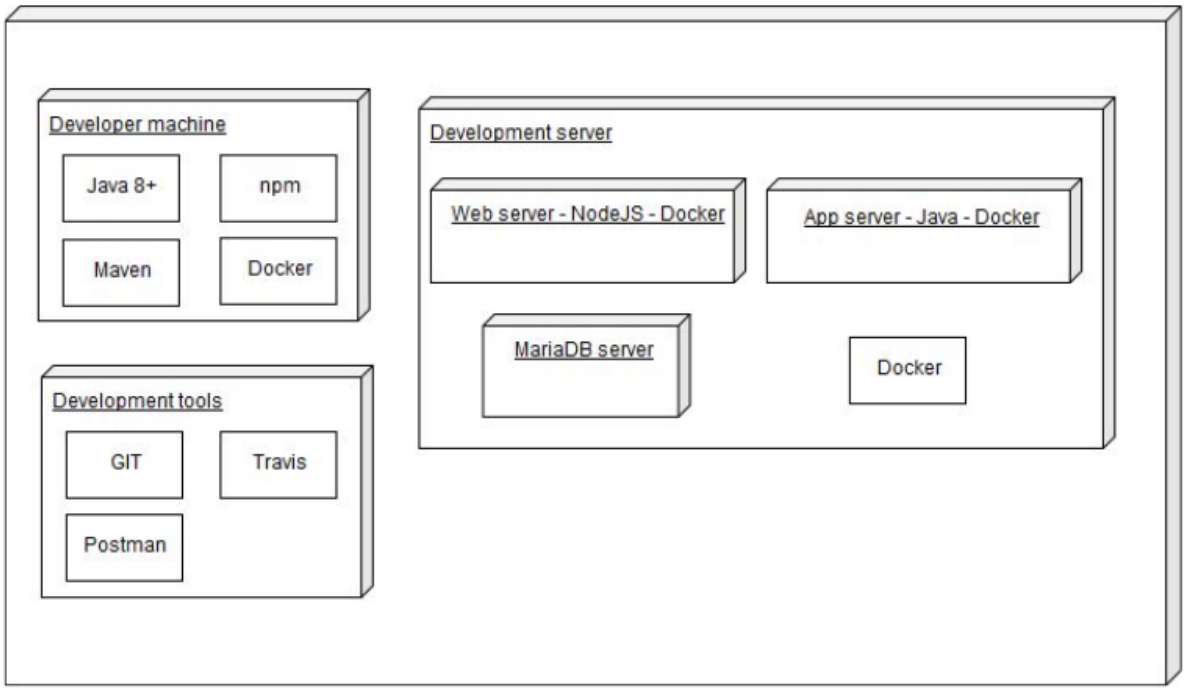

Figure 6: The development environment

3.1.2 Development environment

Figure 6 shows a development environment. The Developer machine box describes technologies which are important for successful compilation and running of the application. The Development tools box helps manage the project and the Development server shows docker containers in which the application is deployed.

3.2 REST API

`GET /users?[lang=<CZ,EN>]&[status=<ACCEPTED, PENDING, REJECTED>]`
Description: It gets all users who can be filtered by their authorisation status.
Content:

- id: long - the user's id
- firstName: string - the user's first name
- lastName: string - the user's last name
- photo: string - the URL of a photo
- calendar: list<vacation_day> - the list of selected vacation in +-7 days
  - id: long - the id of the vacation (sick day)
  - date: yyyy/mm/dd - the date of the vacation (sick day)
  - from: hh:mm - the start of the vacation (sick day = null)
  - to: hh:mm - the end of the vacation (sick day = null)
  - type: <VACATION, SICK_DAY> - the vacation or sick day

`GET /users/requests/vacation?[lang=<CZ,EN>]&[status=<ACCEPTED, PENDING, REJECTED>]`
Description: It gets all requests to select the every user's vacation or sick day. They can be filtered by their approval.
Content:

- id: long - the id of the vacation (sick day)
- firstName: string - the user's first name
- lastName: string - the user's last name
- date: yyyy/mm/dd - the date of the vacation (sick day)
- from: hh:mm - the start of the vacation (sick day = null)
- to: hh:mm - the end of the vacation (sick day = null)
- type: <VACATION, SICK_DAY> - the vacation or sick day
- status: <ACCEPTED, PENDING, REJECTED> - the approval status of the request
- timestamp: yyyy/mm/dd hh:mm:ss - the creation of the request

`GET /users/requests/authorization?[lang=<CZ,EN>]&[status=<ACCEPTED, PENDING, REJECTED>]`
Description: It gets all requests to authorise the user. They can be filtered by their authorisation status.
Content:

- id: long - the user's id
- firstName: string - the user's first name
- lastName: string - the user's last name
- status: <ACCEPTED, PENDING, REJECTED> - the status of the authorisation
- timestamp: yyyy/mm/dd hh:mm:ss - the creation of the request

`GET /user/<id || me>/profile?[lang=<CZ,EN>]`
Description: It gets the full profile of the logged user or the user with the given id.
Content:

- id: long - the user's id
- firstName: string - the user's first name
- lastName: string - the user's last name
- photo: string - the URL of a photo
- vacationCount: float - the number of available vacations
- sickDayCount: int - the number of assigned sick days

- takenSickDayCount: int - the number of taken sick days
- status: <ACCEPTED, PENDING, REJECTED> - the authorisation status
- role: <EMPLOYEE, EMPLOYER> - the user role
- notification: yyyy/mm/dd hh:mm:ss - the notification of an incoming reset of remaining vacations and sick days
- email: string - the user's email address

`GET /user/<id || me>/calendar?[lang=<CZ,EN>]&from=yyyy/mm/dd, [to=yyyy/mm/dd], [status=<ACCEPTED, PENDING, REJECTED>]`
Description: It gets all vacations and sick days in the given range. If "to" is not specified it returns all vacations and sickdays from "from". They can be filtered by their approval.
Content:

- id: long - the id of the vacation (sick day)
- date: yyyy/mm/dd - the date of the vacation (sick day)
- from: hh:mm - the start of the vacation (sick day = null)
- to: hh:mm - the end of the vacation (sick day = null)
- type: <VACATION, SICK_DAY> - the vacation or sick day
- status: <ACCEPTED, PENDING, REJECTED> - the approval status

`GET /user/id/settings?[lang=<CZ,EN>]`
Description: It gets settings of the user with the given id.
Content:

- vacationCount: float - the remaining vacations
- sickDayCount: int - the number of assigned sick days
- role: <EMPLOYEE, EMPLOYER> - the user role

`GET /setttings?[lang=<CZ,EN>]`
Description: It gets the latest default settings.
Content:

- sickDayCount: int - the number of assigned sick days
- notification: yyyy/mm/dd hh:mm:ss - the notification of an incoming reset of remaining vacations and sick days

`POST /settings?[lang=<CZ,EN>]`
Description: It creates new default settings.
Content:

- sickDayCount: int - the number of assigned sick days
- notification: yyyy/mm/dd hh:mm:ss - the notification of an incoming reset of remaining vacations and sick days

`POST /user/calendar/create?[lang=<CZ,EN>]`
Description: It creates new vacation or sick day.
Content:

- date: yyyy/mm/dd - the date of the vacation (sick day)
- from: hh:mm - the start of the vacation (sick day = null)
- to: hh:mm - the end of the vacation (sick day = null)
- type: <VACATION, SICK_DAY> - the vacation or sick day

`PUT /user/settings?[lang=<CZ,EN>]`
Description: It changes the settings of the user with the given id.
Content:

- id: long - user's id
- vacationCount: float - the remaining vacations
- sickDayCount: int - the number of assigned sick days
- role: <EMPLOYEE, EMPLOYER> - the user role

`PUT /calendar/edit?[lang=<CZ,EN>]`
Description: It edits the vacation or sick day with the given id.
Content:

- id: long - the id of the vacation (sick day)
- date: yyyy/mm/dd - the date of the vacation (sick day)
- from: hh:mm - the start of the vacation (sick day = null)
- to: hh:mm - the end of the vacation (sick day = null)

`DELETE /calendar/delete?[lang=<CZ,EN>]`
Description: It deletes the vacation or sick day with the given id.
Content:

- id: long - the id of the vacation (sick day)

`PUT /user/requests?[lang=<CZ,EN>]&type=<VACATION, AUTHORIZATION>`
Description: It approves or disapproves request.
Content:

- id: long - the user's id for the type=AUTHORIZATION or the id of the vacation for the type=VACATION
- status: <ACCEPTED, REJECTED> - the approval status

`GET /export/pdf?[lang=<CZ,EN>]`
Description: It exports .pdf file.
Content:

- binary data (pdf)

`POST /import/xls?[lang=<CZ,EN>]&file=<binary_file>`
Description: It imports .xls file.
Content:

- name: string - the name of the file
- size: long - the size of the file

3.3 Frontend

3.3.1 Technologies

- **Angular** - a `TypeScript-based` opensource framework for building web applications. The Angular 7 was used in this project
- **Bootstrap** - a library for building responsive applications
- **Angular Material** - an Angular library which contains UI elements
- **ngx-translate** - a library for Angular that provides internationalisation

3.3.2 Main structure

Foundation stones of the frontend are three Angular components - `App`, `Dashboard` a `Employees`. The `App` component is a backbone of the application and with a help of `app-routing` module, renders the `Dashboard` or `Employees` component. The `Dashboard` component is visible to the employer and employee, but its content changes depending on a role of the user. Both roles share few subcomponents:

- The component that provides information about remaining hours of vacation and sick days.
- The component with ongoing time off which permits to delete selected vacations and sick days.
- The calendar that shows all accepted vacations and sick days and enables to select the desired time off.

The employer has two more components:

- The component for approval of new users.
- The component for approval of new vacations and sick days.

The `Employees` component is available just for the employer. The component provides multiple functions. Above all it is used to manage users. Other functions are management of default settings, import of xls files and export of pdf.

3.3.3 API

Communication between frontend and backend or rather frontend and api is implemented through `api services`. These `services` are used by components which gets or sends information to backend. For example:

- `basic` - a basic `service` which is extended by other more specific `services`
- `file` - import/export of files
- `settings` - management of default settings
- `user` - operations with the user
- `users` - operations with all users

3.3.4 Localisation

Localisation has been built on the `ngx-translate` library. Language can change without reloading the screen. Files with translations for Czech and English languages are saved in `webapp/src/app/ assets/i18n/[jayzk].json`

3.4 Backend

3.4.1 Structure

The server part of the application is divided into six primary packages:

- `business` - main business logic which consists of file management and handling of API messages
- `domain` - domain classes which corresponds to tables in the database and contains their logic
- `dto` - objects send through the API
- `repository` - database queries
- `localization` - localisation of error messages
- `rest` - REST API services

3.4.2 File management

Multiple messengers and enumerations are located in file management but main business code is focused inside `FileServiceImpl` whose meaning is to generate a pdf file and calls methods which parse xls files. The parsing is implemented in `business/file/excel`, more precisely classes `SheetAttendanceParser` and `ExcelParses` hanles that functionality. The first one takes care of parsing of a single sheet inside the xls file and the second class uses the previous one and parses the whole document.

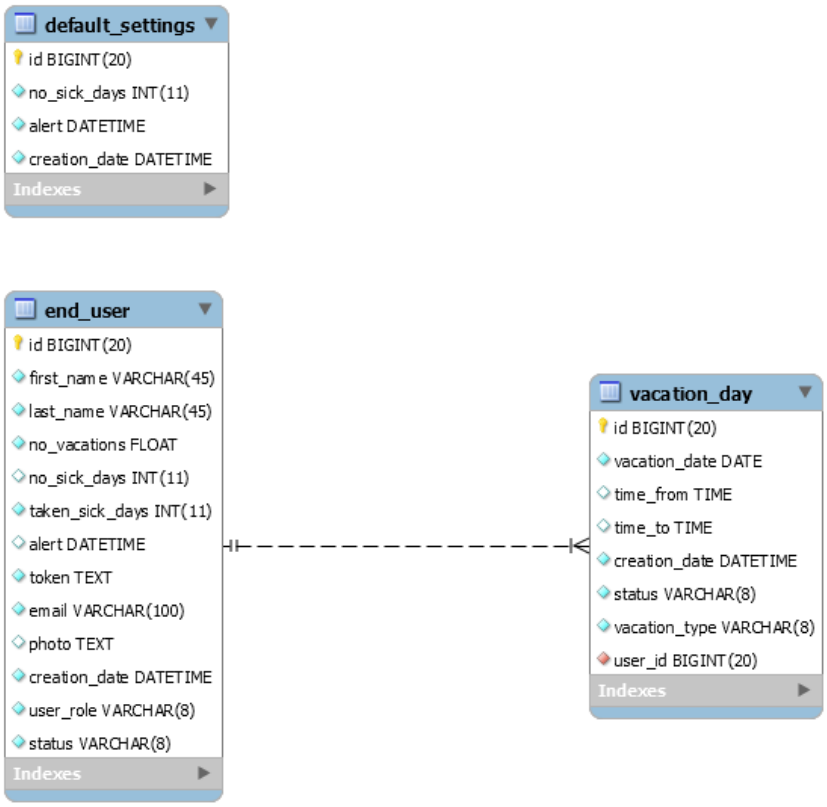

Figure 7: ERA model

3.4.3 Database

Domain classes `DefaultSettings`, `User` and `Vacation` are equivalent to tables *default_settings*, *end_user* and *vacation_day* in the database whose ERA model is in Figure 7. Domain classes primarily contains getters and setters which includes input restrictions and controls data that are sent to the database.

The communication itself is handled by `RequestRepository`, that takes care of getting users and vacations in a form of requests for authorisation and approval of vacations, `UserRepository`, that controls communication with *end_user* and *default_settings* tables, and at last but not least `VacationRepository` that gets, inserts and updates records in the *vacation_day* table.

3.4.4 REST API

The first of classes, which deal with communication over REST API, are DTOs. They are messengers that are sent via REST API. The most vital components are classes inside the `rest` package and the `Manager` interface and the `ApiManager`, which implements it in the `business` package. If we take a look inside the `rest` package there are multiple classes and interfaces and the most important are the `RESTConfiguration`, which transforms dates and times so they can be sent to the frontend, and `ApiController` that consists of REST API endpoints. Requests from the endpoints are susequently processed in the `ApiManager` class. It interconnects all modules of the application and calls everything necessary to process all requests.