Content

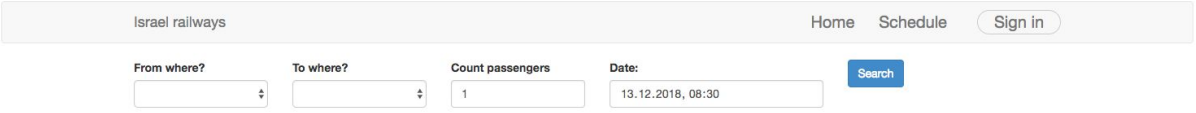
Abstract

This application simulates the work of the information system of the transport railway company. The system provides various opportunities for customers and administrators (company employees). The client can search for travel options and purchase tickets for himself and friends. The administrator's functionality includes working with a database of routes, namely the creation / modification / deletion of stations and routes, including coordinates, distances and prices.

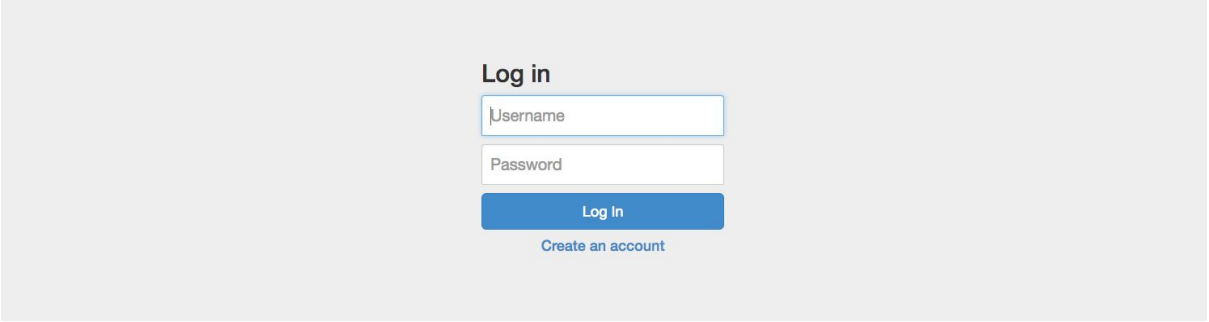
Userstory

Registration

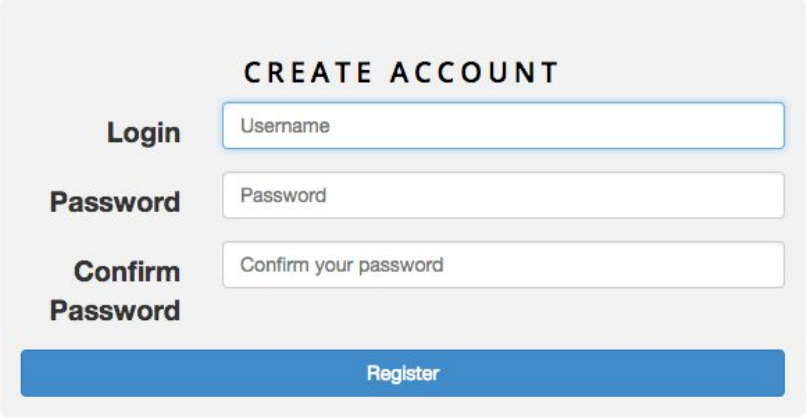
First user gets into main page



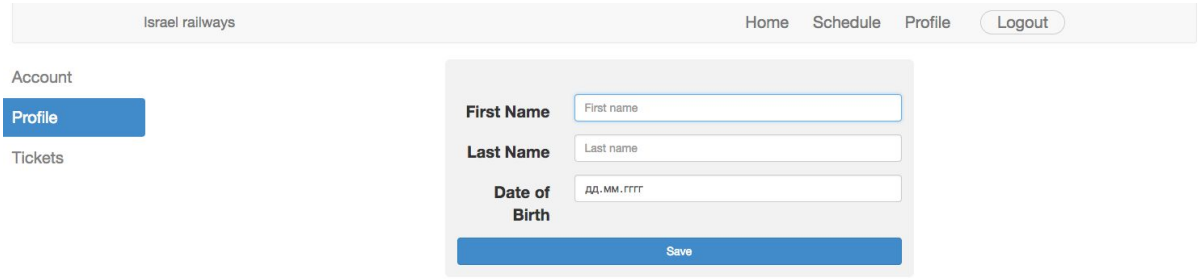
Then user click “Sign in” and gets into login page



Then clicking “Create an account” user gets into registration form

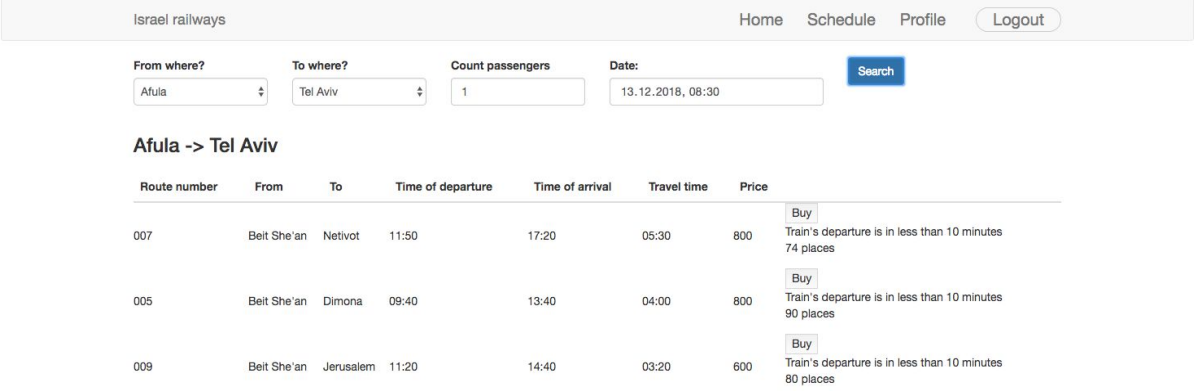


Then user fills fields (there are some filter on fields). Finished users gets into Profile page where he can filled his personal data such as surname, firstname and birthday. This fields are required for buying ticket. If user wasn’t fill these field he must fill it each time then he want to buy ticket.

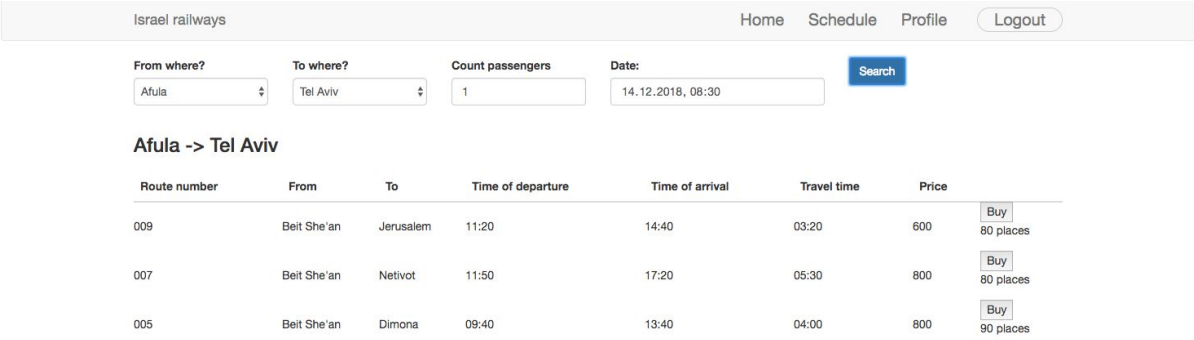


Searching route and buying ticket

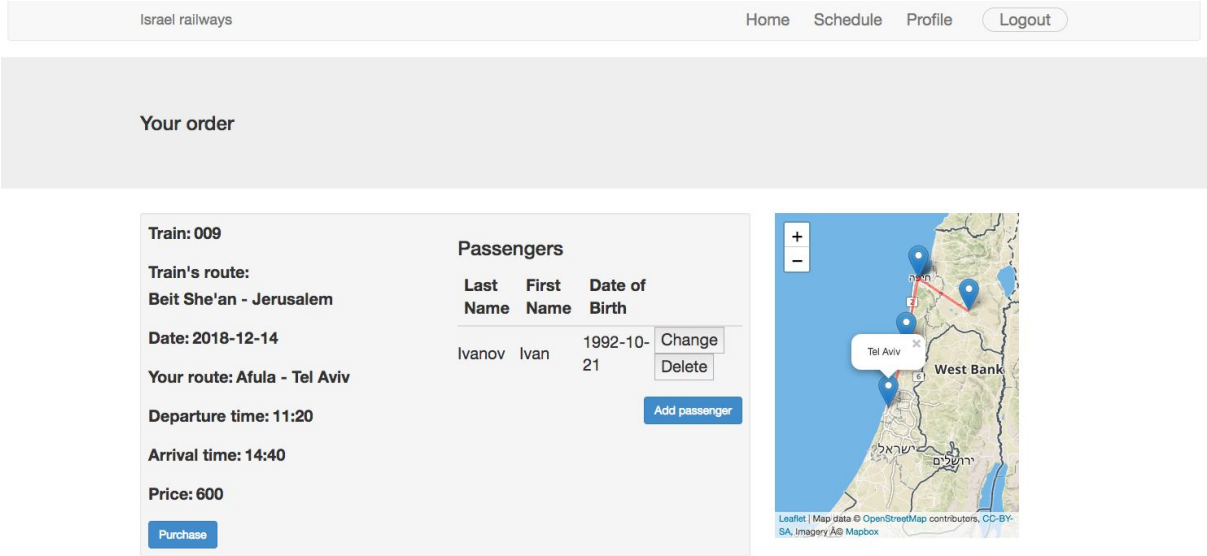
Then user can searching routes. User can click “Home” and gets into home page back. There user can choose station from which he wants to go, station to which he wants to go and date of trip. Also user can choose minimal time of departure.



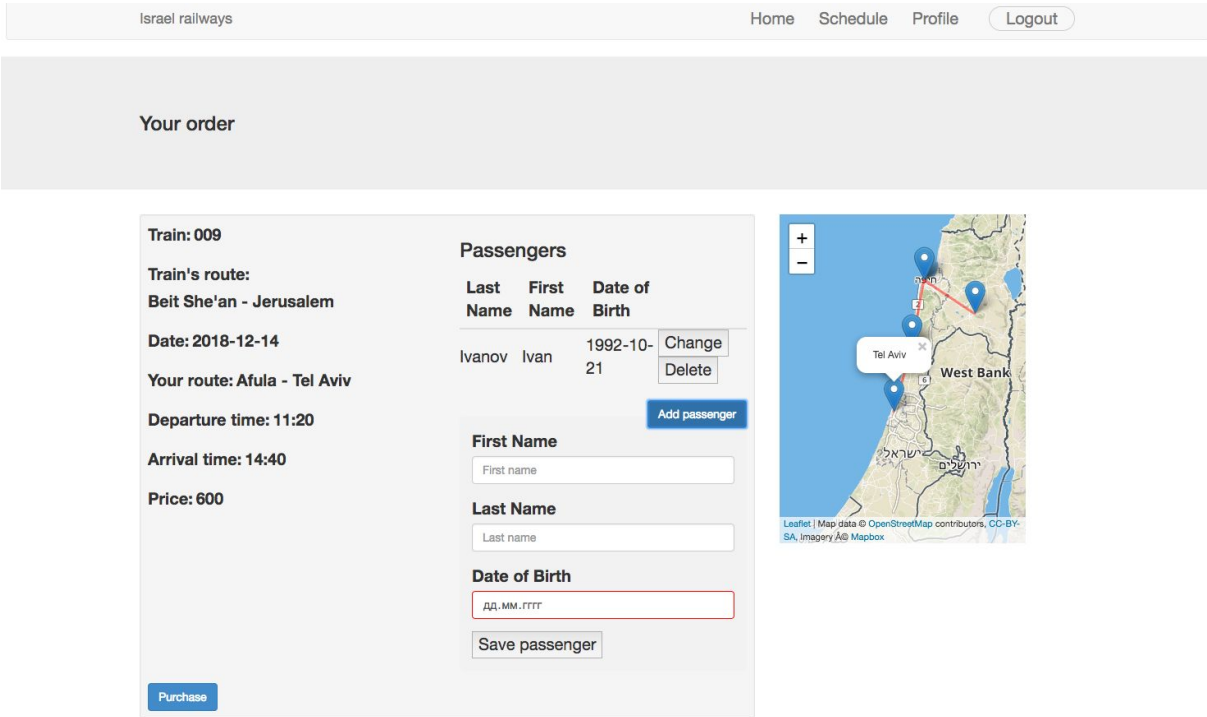
For this date all train are already gone and user can’t buy ticket. Let’s select next day.



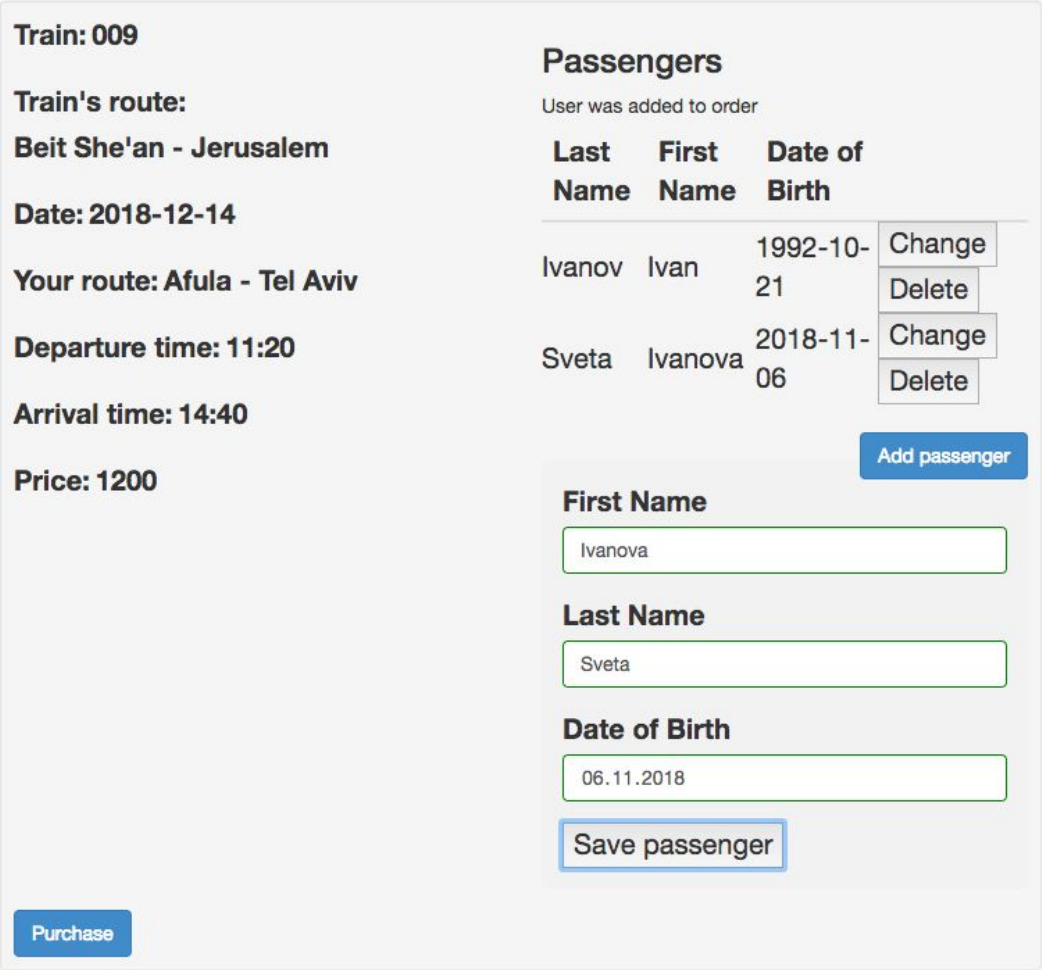
Now user can click on “Buy” in suitable route.Then user goes into ticket’s page.



There he can see his route on map. Also he can add other passengers clicking on “Add passenger”.

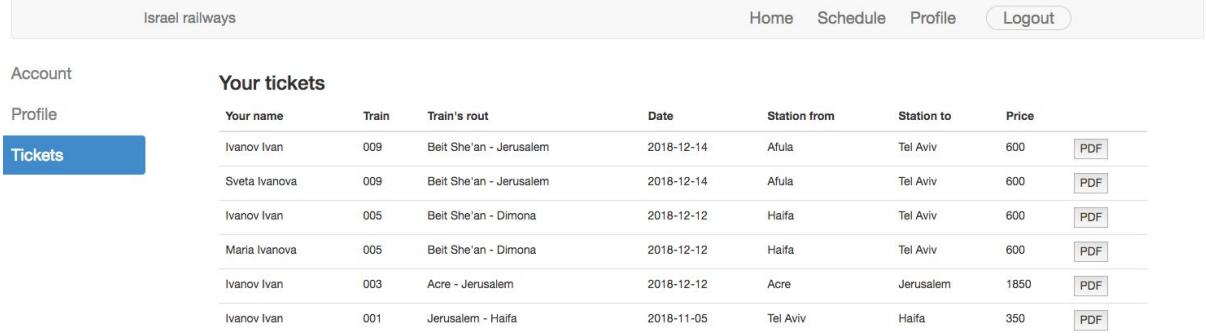


Then he must enter data of person and click “Save passenger”. When he added all his additional passengers. User can click “Purchase” and purchase tickets.



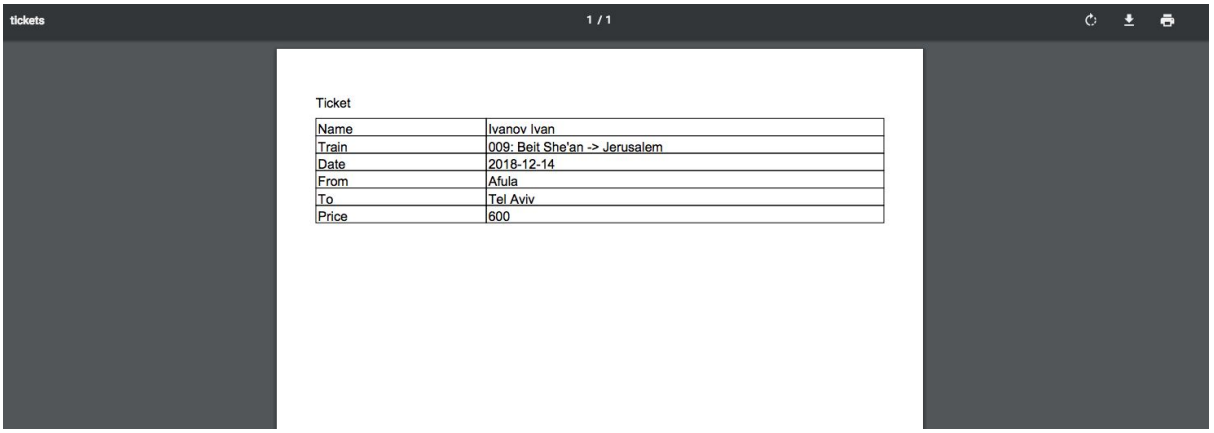
Showing tickets

Then user can show his tickets in profile by clicking “Profile” and then “Tickets”.



Your name	Train	Train's rout	Date	Station from	Station to	Price	
Ivanov Ivan	009	Beit She'an - Jerusalem	2018-12-14	Afula	Tel Aviv	600	PDF
Sveta Ivanova	009	Beit She'an - Jerusalem	2018-12-14	Afula	Tel Aviv	600	PDF
Ivanov Ivan	005	Beit She'an - Dimona	2018-12-12	Haifa	Tel Aviv	600	PDF
Maria Ivanova	005	Beit She'an - Dimona	2018-12-12	Haifa	Tel Aviv	600	PDF
Ivanov Ivan	003	Acre - Jerusalem	2018-12-12	Acre	Jerusalem	1850	PDF
Ivanov Ivan	001	Jerusalem - Haifa	2018-11-05	Tel Aviv	Haifa	350	PDF

All his tickets user can print in pdf format.



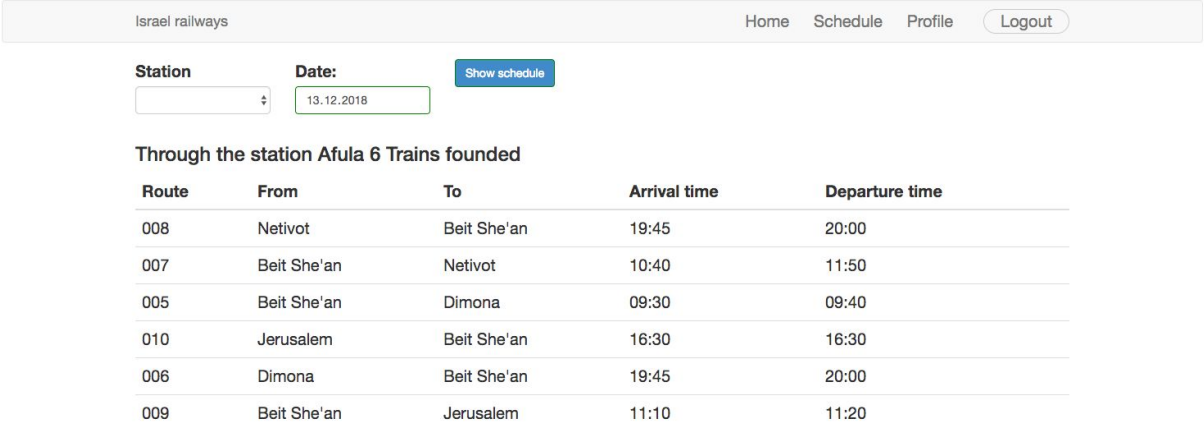
tickets 1 / 1

Ticket

Name	Ivanov Ivan
Train	009: Beit She'an -> Jerusalem
Date	2018-12-14
From	Afula
To	Tel Aviv
Price	600

View timetable

User can watch timetable by clicking “Schedule” in top menu. Then choose station and date.



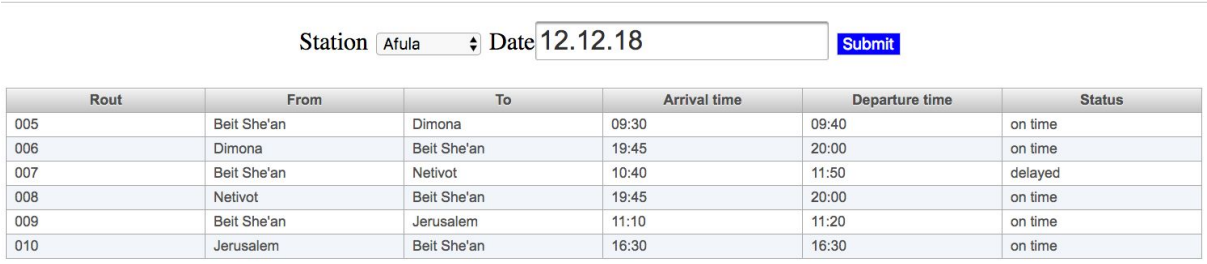
Israel railways Home Schedule Profile Logout

Station Date: 13.12.2018 Show schedule

Through the station Afula 6 Trains founded

Route	From	To	Arrival time	Departure time
008	Netivot	Beit She'an	19:45	20:00
007	Beit She'an	Netivot	10:40	11:50
005	Beit She'an	Dimona	09:30	09:40
010	Jerusalem	Beit She'an	16:30	16:30
006	Dimona	Beit She'an	19:45	20:00
009	Beit She'an	Jerusalem	11:10	11:20

Or user can watch actual information in timetable on Tablo.



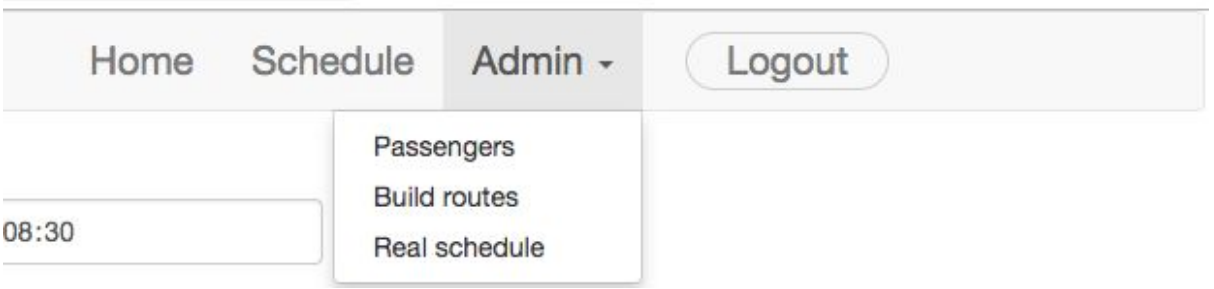
Station Afula Date 12.12.18 Submit

Rout	From	To	Arrival time	Departure time	Status
005	Beit She'an	Dimona	09:30	09:40	on time
006	Dimona	Beit She'an	19:45	20:00	on time
007	Beit She'an	Netivot	10:40	11:50	delayed
008	Netivot	Beit She'an	19:45	20:00	on time
009	Beit She'an	Jerusalem	11:10	11:20	on time
010	Jerusalem	Beit She'an	16:30	16:30	on time

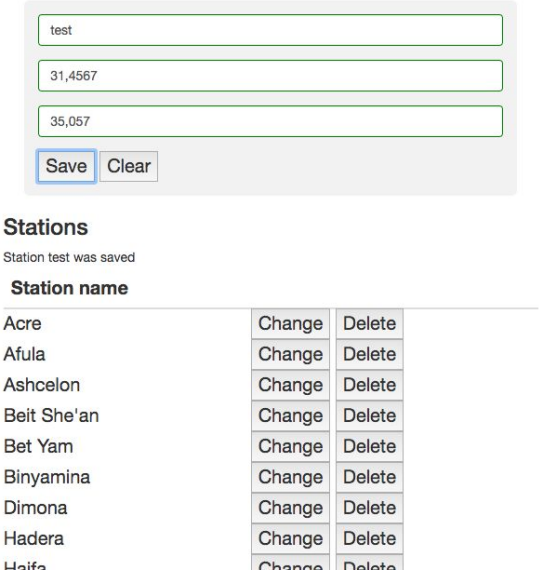
Adminstory

Creating route

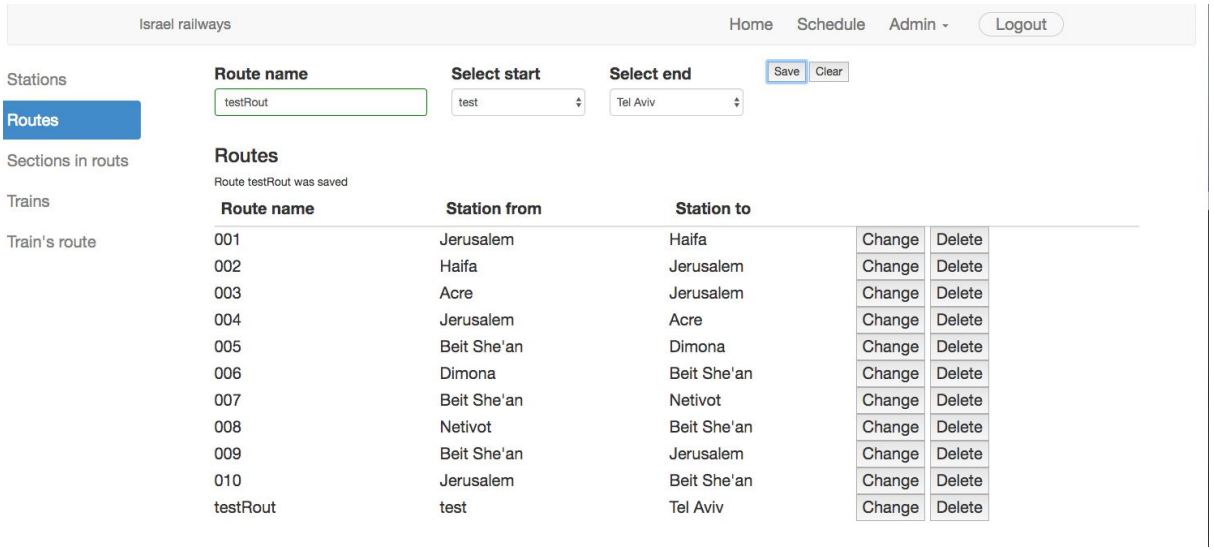
Admin can log in with “Username” “admin” and password “adminadmin”. When he log in, in top menu admin menu appears.



Selecting “Build routes” admin gets into managing stations page. There admin can create new station with manual geographical coordinates or can click on map choosing coordinates. Also it possible to changing and deleting existing stations.

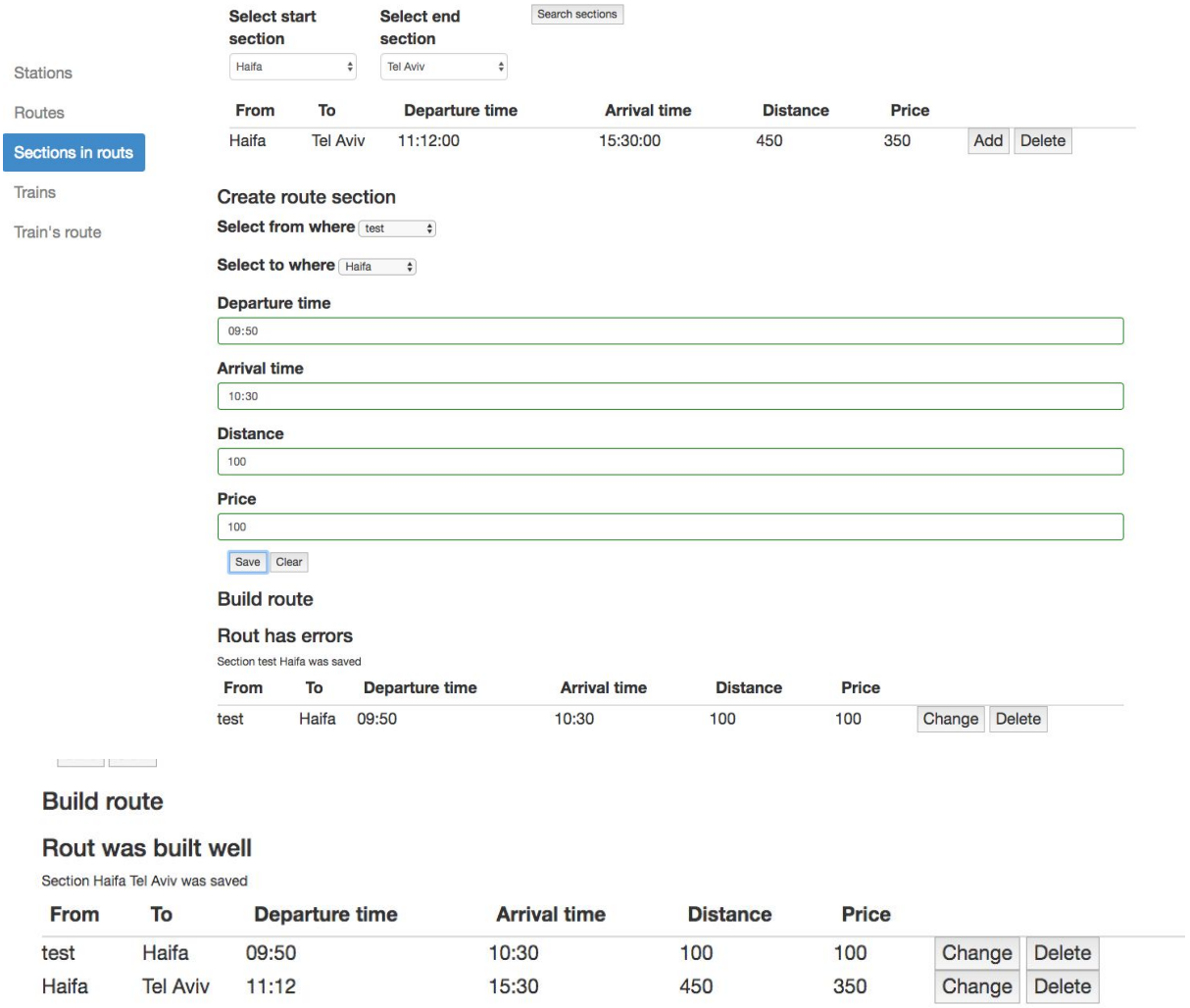


On tab “Routes” admin can managing general information about routes, like name, start and station. Also it possible to changing and deleting existing routes.

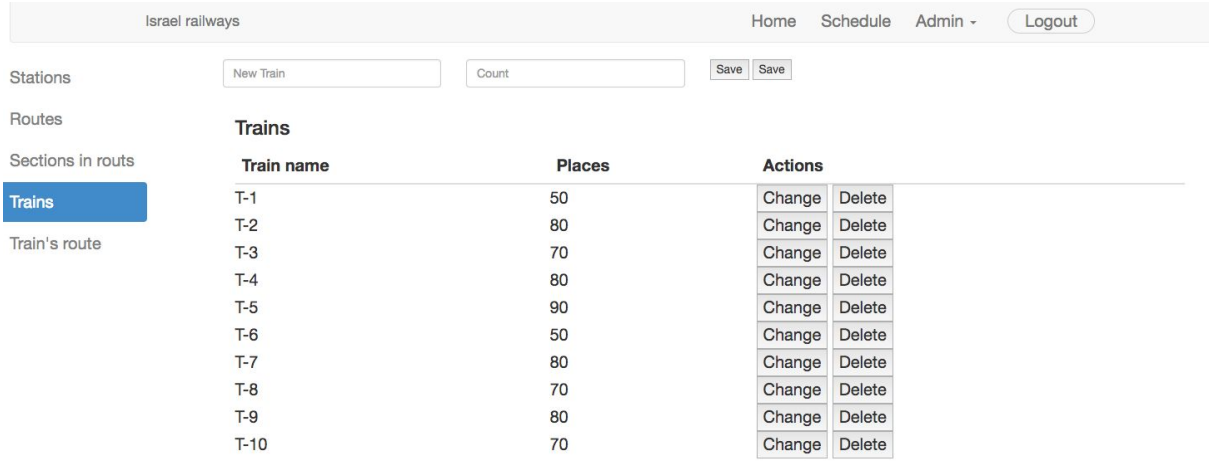


On tab “Sections in routes” admin can managing information about routes legs. It is possible to add existing route leg or create new by clicking on the corresponding buttons. There is validation stations

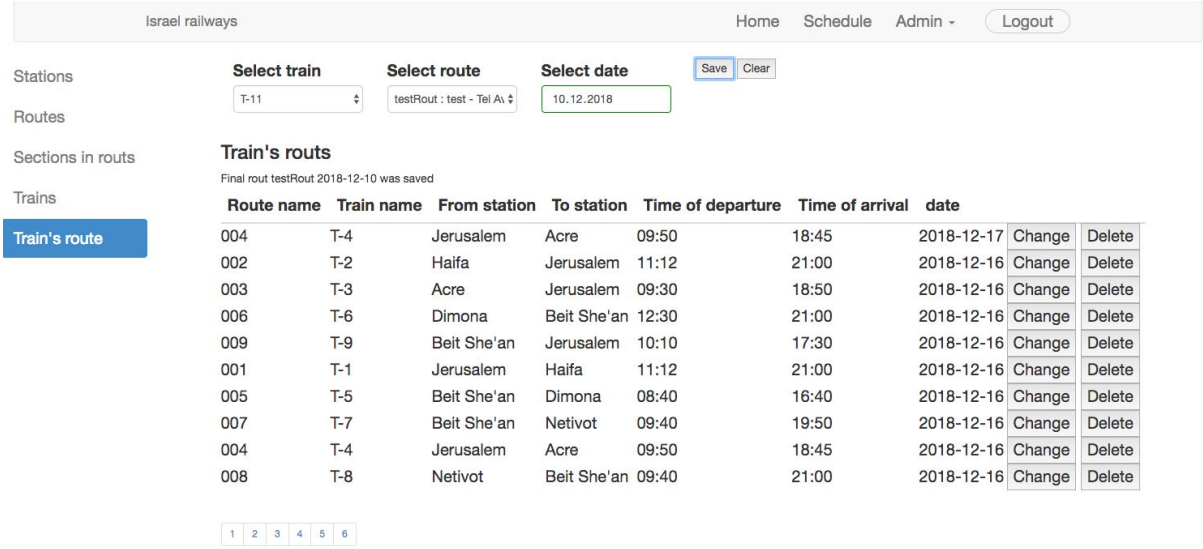
in routes. For right route stations must be from start station of route to end station of route consistently.



On tab “Trains” admin can managing general information about trains, like name and count places in train.. Also it possible to changing and deleting existing trains.

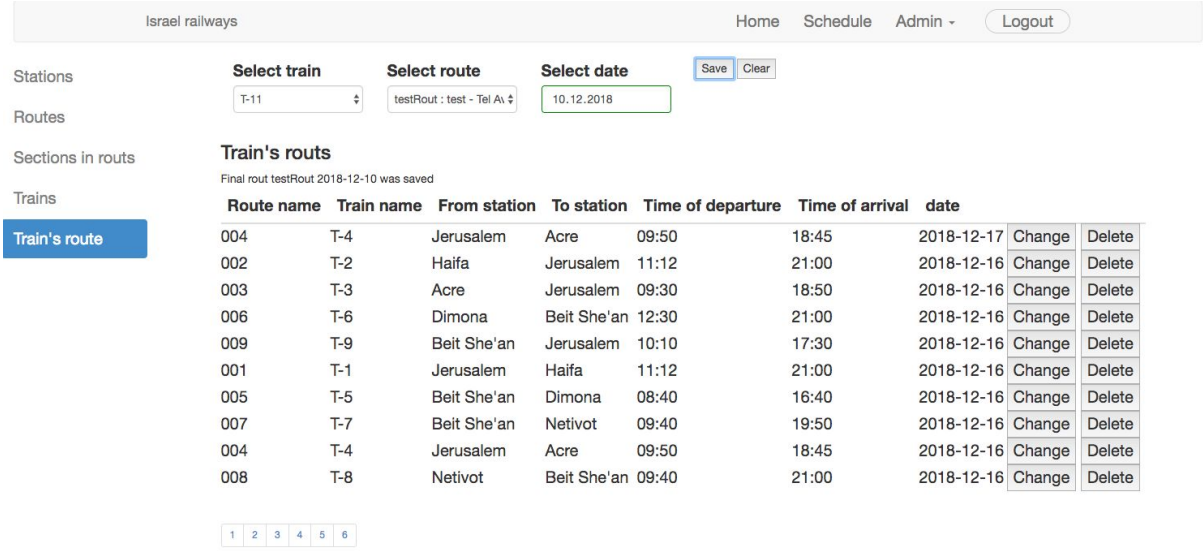


And finally admin map route and train on date on tab “Train’s route”.



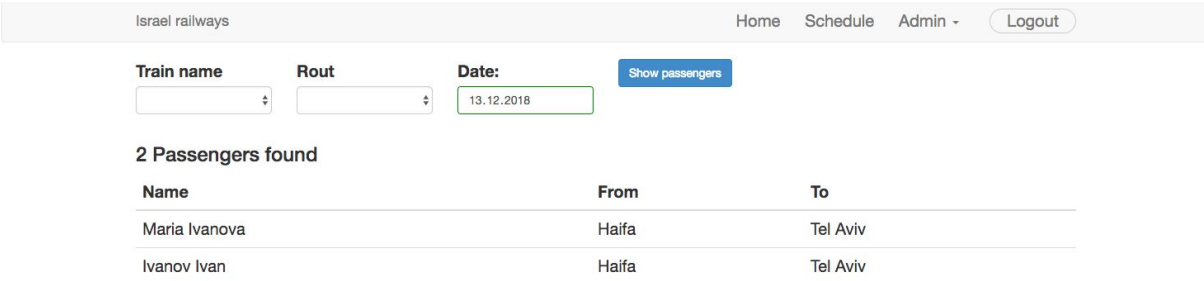
Managing real schedule

Selecting “Real schedule” admin gets into managing timetable page. Admin can choose station and date and then send “on time”, “delayed” and “canceled” for route.



View passengers

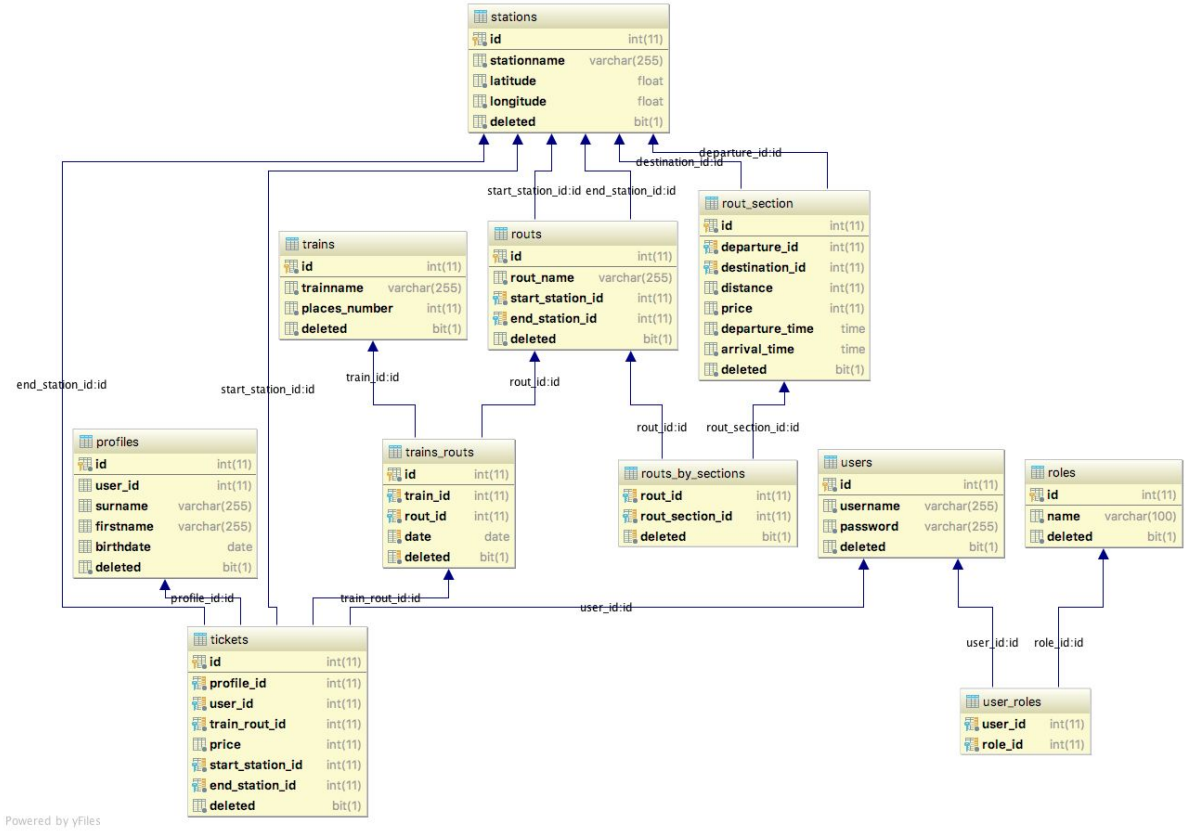
Selecting “Passengers” admin gets into page where he can view passengers list in train- route on date.



Architecture

Data model

According to scheme route represents set of route sections which starting from start rout station and end to end route station. Route is not attached to a specific date. It allows create one route for several days. Date for route is set farther. It's possible create different route for one start and end stations: it needs only create different route sections for this routes.



General

All tables have field "deleted". This field provides soft deleting of record: instead of real deleting it just marked as deleted. Almost all table have filed id, accept tables which only map two entity.

Table users

The table represents data of account user. It contains username and password. Field username must be unique.

Table roles

The table represents data of role. It contains name role which must be unique. Each user from table Users has one or many roles, and each role has one or many users. It's provided by table user_roles.

Table profile

The table represents data of user's profile. It contains user_id, surname, firstname and birthday. Field user_id represents user account for this profile, but profile can belong nothing.

Table tickets

The table represents data of ticket. It contains id, user_id, profile_id, train_rout_id, price, start_station_id and end_station_id. Field user_id represents user account which buy ticket. Field profile_id represents data of user's profile. Field price shows price which paid user for ticket.

Table trains

The table represents data of train. It contains name train and count places in train. Name train must be unique, it is internal name (not visible for passengers).

Table stations

The table represents data of station. It contains station name and geography coordinates: latitude and longitude.

Table rout_sections

The table represents route leg. It contains two stations (without "stop" between them): start and end, price for section, distance between two stations, departure and arrival time. All fields are required. If station in section was deleted, this rout section also will be deleted.

Table routes

The table represents data of rout. It contains route name, start and end stations. All fields are required. Routes are mapped with its sections by table routs_by_sections. If station in route was deleted, this route also will be deleted.

Table trains_routs

The table represents data of scheduled train. It contains train_id, rout_id and date. All fields are required. If train or route was deleted, this scheduled train also will be deleted.

Entities

BaseEntity includes fields "id" and "deleted" - it general for all other entities which extends BaseEntity.

Roles represents table Roles. It links with UserData by @ManyToMany.

UserDate represents table Users and links with Roles, UserProfile (@OneToMany because one account can represent one person), and Ticket (@OneToMany because one user account can have many tickets, but one ticket can have only one buyer).

UserProfile represents table Profiles and links with UserData and Ticket.

Station represents table Station and links with RoutSections (@OneToMany because one section can be in different route sections as start and end of sections) and Route (@OneToMany because one section can be in different route as start and end of route).

RoutSection represents table RoutSections and links with Station and Rout (@ManyToMany because in route can be many sections and one section can be in different routes).

Rout represents table Rout and links with Station, RoutSection and FinalRout (@OneToMany because one route can be in different final route, but one final route have only one route).

Train represents table Trains and links with FinalRout (@OneToMany because one train can be in different final route, but one final route have only one train).

FinalRout represents table FinalRout and links with Train and Rout.

Business logic

Application has ten services, which proves transactionality if it is necessary:

UserService - provides create, change and delete user account and user profile. It allows find user by username, profile's data, id.

UserDetailsService - provides safe load granted authority set by username.

StationService - provides create, change and delete stations. It allows find station by name, id and all stations.

SheduleSenderService - provides sending messages for Tablo.

SecurityService - provide auto logging user after registration and safe log in user.

TrainService - provides create, change and delete trains. It allows find train by name, id and all trains.

TicketService - provides create one ticket and forming several tickets on group passengers. Also it allows delete ticket, find by account, find by profile, find by rout and date, validate user with repeating profile's data in train and get free places in route on date.

RoutService - provides create, change and delete route. It also allows to find by Id, name, find valid routes (with right routes legs), find by start station, find by route leg, get price for route from start to end stations, and so on.

RoutSectionService - provides create, change and delete route section. It also allows find by id, departure and destination stations.

FinalRoutService - provides create, change and delete final route (route on date with train). It also allows find all final routes, by page of 10 records, by date, by station and date, get map time departures, get map time arrivals, get prices in custom route, validate departure train is in less than 10 minutes.

Tests

JUnit tests created for all services in application. Checking all methods in services is in that tests with Mocks. Also it was created Selenium tests for checking web login user and admin.

Technologies and frameworks

- JDK 9
- IDE IDEA
- WildFly 14.01
- DB MySql 8.0.11
- Maven
- Spring Framework 5.1.1
- JSP
- EJB
- JSF
- ActiveMQ
- WebServices

Features:

- JS + Ajax - to make asynchronous HTTP calls to the server and reload parts of page
- itextpdf - to print ticket to pdf
- Sonar - to analyze code
- Leaflet - for interactive maps on pages
- Bootstrap - for styles on pages
- Selenium - to automating tests web

UI

For page styles bootstrap 3.3.0 styles used mainly. It provides fast load page if this styles already on user's computer. But there some custom style proves larger font, padding and colors. There are custom styles for admin navbar, navbar, registration and login.

For validation on page library jquery/validation and html-validation used. jquery/validation example:

```
$().ready(function () {
  $('#userForm').validate({
    rules: {
      ...
      password: {
        required: true,
        minlength: 8
      },
      confirmPassword: {
        required: true,
        minlength: 8
        equalTo: "password"
      }
    },
    messages: {
        ...
      password: {
        required: "Please enter password",
        minlength: "Your password must be at least 8 characters long"
      },
      confirmPassword: {
```

```
            required: " Please enter password",
            minlength: "Your password must be at least 8 characters long"
            // equalTo: "Please enter the same password as above"
        },
        ...
    }
  });
});
```

Request from users send by js and ajax. Map shows by library open-source library Leaflet for open-street-map. Example:

```
function stationEdit(index) {
  event.preventDefault();
  var station = $("#idStation-" + index).val();
  var object = {stationId: station};
  $.post(contextPath + "/admin/stations?change", object).done(function (result) {
    $('#stationMessage').text('');
    $('form[name=stationForm]').val(result);
    $('#idForm').val(result.id);
    $('#stationName').val(result.stationName);
    $('#latitude').val(result.latitude);
    $('#longitude').val(result.longitude);
    if (theMarker != undefined) {
      mymap.removeLayer(theMarker);
    };
    var latitude = $('#latitude').val();
    var longitude = $('#longitude').val();
    theMarker = L.marker([latitude, longitude]).addTo(mymap)
      .bindPopup($('#stationName').val()).openPopup();
  }).fail(function () {
    $('#stationMessage').text('Edit station failed');
  });
}
```

Build and deploy

First, it is important to create database scheme railway_site_db via command in MySql-console : CREATE DATABASE railway_site_db.

Second, run script database.sql in resources folder. It create all necessary tables and links between them.

Third, load and install WildFly server.

Fourth, in main folder of project run command: mvn clean install wildfly:wildly.

After this, in the folder where wildfly is stored via command cd /standalone/configuration gets into configuration folder, where it is necessary rename standalone-full.xml to standalone.xml, then edit this fail. In description of jms-queue it is necessary add <jms-topic name="rwTopic" entries="java:/jms/topic/rw"/>, and in deployments-section:

```
<deployment name="mysql-connector-java-8.0.11.jar" runtime-name="mysql-connector-java-8.0.11.jar">
        <content sha1="2c3d25fe1dfdd6496e0bbe47d67809f67487cfba"/>
    </deployment>
```

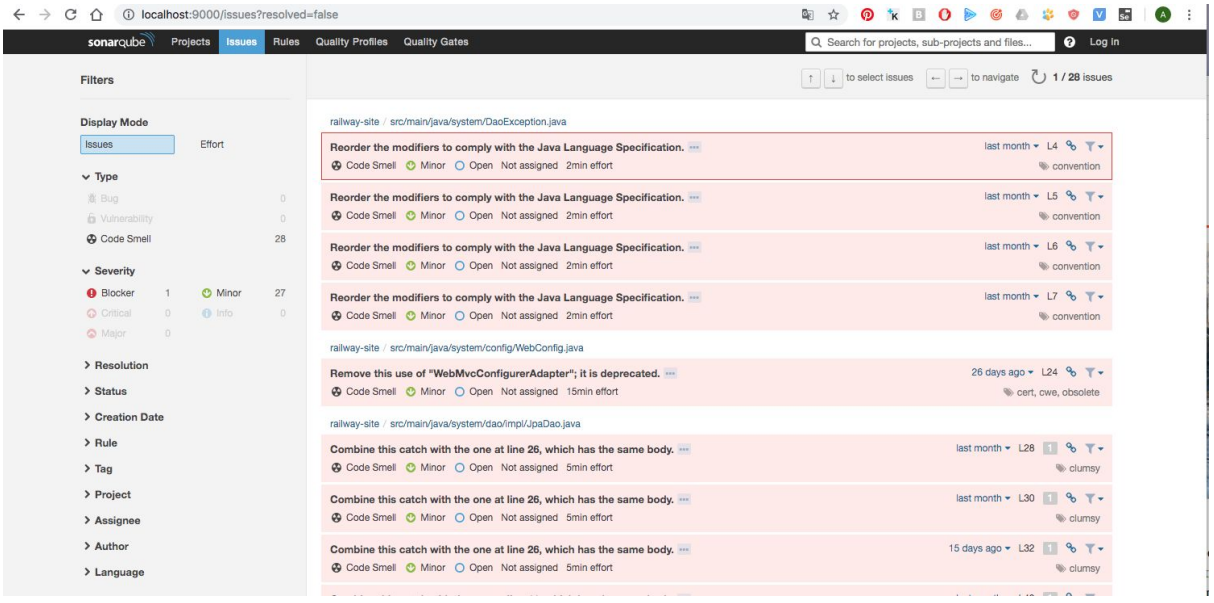
Then via command in console: cd ../../bin gets into run-folder and run standalone.sh

Logging

All log represents in log/mylog.log. Logging provides by Logback. In application errors in dao and services, info about logging/logout, user actions are logged. For example:

10:15:22.548 [main] INFO s.s.i.FinalRoutServiceImpl#73 Deleted Final Rout from station1 to station4
10:15:22.591 [main] INFO s.s.i.FinalRoutServiceImpl#58 Created Final Rout from station1 to station4
10:15:22.722 [main] INFO s.s.i.RoutSectionServiceImpl#50 Created Rout Section from station1 to station2
10:15:22.732 [main] INFO s.s.i.RoutSectionServiceImpl#65 Deleted Rout Section from station1 to station2
10:15:22.867 [main] INFO s.s.i.RoutServiceImpl#52 Created Rout from station1 to station4
10:15:22.886 [main] INFO s.s.i.RoutServiceImpl#67 Deleted Rout from station1 to station4
10:15:22.951 [main] INFO s.s.i.StationServiceImpl#43 Created Station testStation
10:15:22.959 [main] INFO s.s.i.StationServiceImpl#56 Deleted Station testStation

Sonar results



Known issues

The main limitation is the fact that the route is given only one day. For more day on route it is necessary remark some methods dependent from concrete date. It would take a lot of time to develop and debug therefore this issue is resolved to be missed.

The next issue is the fact that error page for PageNotFound is provided by WildFly but not custom. This issue is not significant and could not be sorted out in a short time.

The next issue is the fact that when count of final route on page Train's route became more than a multiple of ten new page doesn't appear dynamic, it is necessary to refresh page for view new page. This issue is not significant and could not be sorted out in a short time.

And finally, tests not covers dao level, and Selenium tests covers only some web-possibility, not all.

Follow steps

It will be nice to add searching route with transfers for the convenience of users. But there was not enough time for developing this issue.

Moreover, for convenience of admin it will be possible to add building route through map by clicking between stations, it is reduce possible mistakes during filling field "distance" in route sections. But there was not enough time for developing this issue.

Also it will be possible to implement loading tickets for profile when new user registered and for this profile information has already exist tickets. But there was not enough time for developing this issue.

It will be great to implement label for searching route "weekly", "monthly" and so on. But there was not enough time for developing this issue.