1. Content

1) introduction

2) List of technologies and frameworks

3) My features

4) The scheme of database

5) Explication and implementation of the model from the task.

6) Modules in the application

7) UI

8) Business logic

9) Entity, DAO, Transactions

10) Screenshots of applications

11) Unit Tests

1. introduction

An application was written that simulates the operation of the information system of a certain company that carries out passenger rail transportation. A multi-user client-server application with a network connection was created. All data is stored on the server side. Each client can download some data, after each data modification operation, the data must be synchronized with the server.

2. List of technologies and frameworks

1. Spring (first application)
2. IDE IDEA
3. Tomcat
4. DB – MySQL
5. Maven
6. JPA
7. AS - WildFly
8. EJB (second application)
9. JSF
10. MQ (for notifications from the server)
11. Rest (for data exchange between the client and the server)
12. Lombok
13. Sonar
14. Log4j
15. Hibernate

3. My features

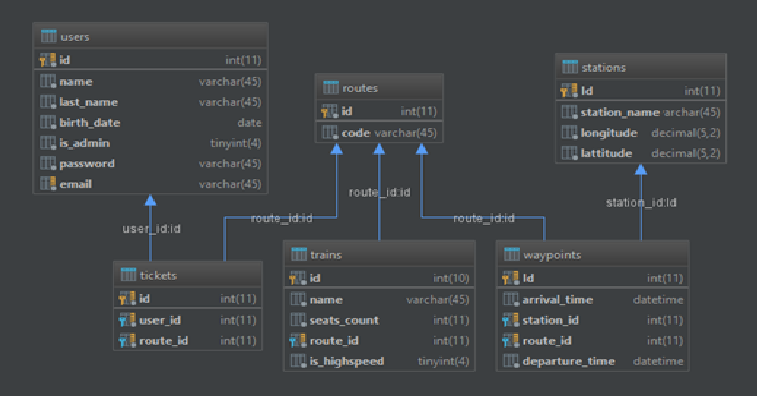
A component is available that counts the distance between stations using their coordinates.

Component calculating the cost of the ticket.

Added localization, English and Russian languages available

Sonar is connected.

4. The scheme of database.



In addition to the required entities we have waypoints, routes and tickets.

Tickets connects to users with “many to one” connection.

Trains connects to routes with “one to one” connection.

Waypoint connect to routes with “many to one” connection.

Waypoints connects to stations with “one to many” connection.

I introduced these entities because it seemed to me logical.

Each train has a route. The route has a waypoint which contains the name of the station and the time when the train will arrive there. At any time, any train can change the route.

5. Explication and implementation of the model from the task.

In my application, according to the task, there is a search for a train going from point A to point B. The table is filled using DTO objects of TrainsStationsDTO. It`s created on jsp pages.

The train schedule for the station is carried out using the data transfer objects StationScheduleDTO. It`s created on jsp pages.

Admin menu carried out using jQuery.

Login menu created with Spring Security

Second application with Station schedule used DTO`s too. Created on JSF, used EJB.

6. Modules in the application

I have 3 modules: Webmodule responsible for frontend (Spring MVC), Backend module responsible for backend, and e-schedule module that separated from other modules. E-schedule module responsible for second application.

Webmodule and backend closely connected, so controllers from webmodule can call services from backend.

WebModule have Rest controller that connects with rest client in e-schedule module.

7. UI

The bootstrap was used for the graphical interface. Header and footer was created with bootstrap to.

Here is my bootstrap imports

<link rel="stylesheet" type="text/css"  
 href="**${**pageContext.request.contextPath**}**/resources/bootstrap/css/bootstrap.css"/>  
<link rel="stylesheet" type="text/css"  
 href="**${**pageContext.request.contextPath**}**/resources/bootstrap/css/bootstrap-grid.css"/>  
<link rel="stylesheet" type="text/css"  
 href="**${**pageContext.request.contextPath**}**/resources/bootstrap/css/bootstrap-reboot.css"/>  
<link rel="stylesheet" type="text/css" href="**${**pageContext.request.contextPath**}**/resources/css/sbb.css"/>  
<script src="**${**pageContext.request.contextPath**}**/resources/bootstrap/js/bootstrap.js"></script>  
<script src="**${**pageContext.request.contextPath**}**/resources/scripts/sbb.js"></script>

Header.css have header visual settings.

Footer.css have footer visual settings.

Admin.css have settings for admin menu.

Sbb.css have settings for all other.

Sbb.js have scripts for:

butTicket() – redirects to buy ticket page and give parametrs for it

goToAdminPage() – redirects to admin menu

goToMyTicketsPage() – redirects to your tickets

goToSchedule() – redirects to Station schedule page

goToRegisterPage() – redirects to registration page

8. Business logic.

My services:

**RouteService**

**StationService**

**TicketService**

**TrainService**

**UserService**

**WaypointService.**

In addition to CRUD requests my services have another method

**RouteService:**

findRouteByCode – find route in database with code entered in string “routeCode”

findAllRoutes – return all routes existing in database

**StationService:**

getAllStations – get all stations existing in database

findStationByName – finds station with required name in database

**TicketService:**

findTicketByUserAndRoute – finds tickets with required user and route

findTicketsByUser – finds all tickets belonging to required user

**TrainService:**

getTrainsByStationsAndDate – return all trains going between required stations in required date

findTrainByRoute – return train with required route

getAllTrains – return all trains

**UserService:**

findUserByEmailAndPassword – return user with required email and password

findUserByNameAndLastNameAndDate – return user with required Name, LastName and date of birth,

findById – return user with required id

WaypointService:

findWaypointByRouteAndStation – return waypoint with required route and station

9.Entity, DAO, Transactions

**Base entity:**

Have only id

**Route:**

Have code, train, waypoints, and tickets

**Station:**

Have name, latitude, longitude and waypoints

**Ticket:**

Have route and user

**Train:**

Name, Boolean isHighspeed, seats\_count and route

**User:**

Name, lastname, date of birth, email, password, Boolean isAdmin and tickets

**Waypoint**:

Arrival time, departure time, station and route

All entities extends BaseEntity, so they can inherit id.

**AbstractDao:**

Creates entity manager

**GenericDao:**

Creates CRUD requests

**RouteDao:**

CRUD requests

findAllRoutes() – returns all routes in database

findRouteByCode – returns route with required code

**StationDao:**

CRUD requests

getStationSchedule – gets station schedule with required date

findStationByName – find station by name

**TicketDao:**

CRUD requests

findTicketByUserAndRoute – finds ticket with required user and route

findTicketsByUser – finds all tickets belongs to the user

**TrainDao:**

CRUD requests

getTrainsByStationsAndDate – get trains between a and b stations in required date

findTrainByRoute – find train by route

**UserDao:**

CRUD requests

findUserByEmailAndPassword – find user with required email and password

findUserByNameAndLastNameAndDate – find user with required name, lastname and date of birth

findById – find user by id

**WayPointDao:**

CRUD requests

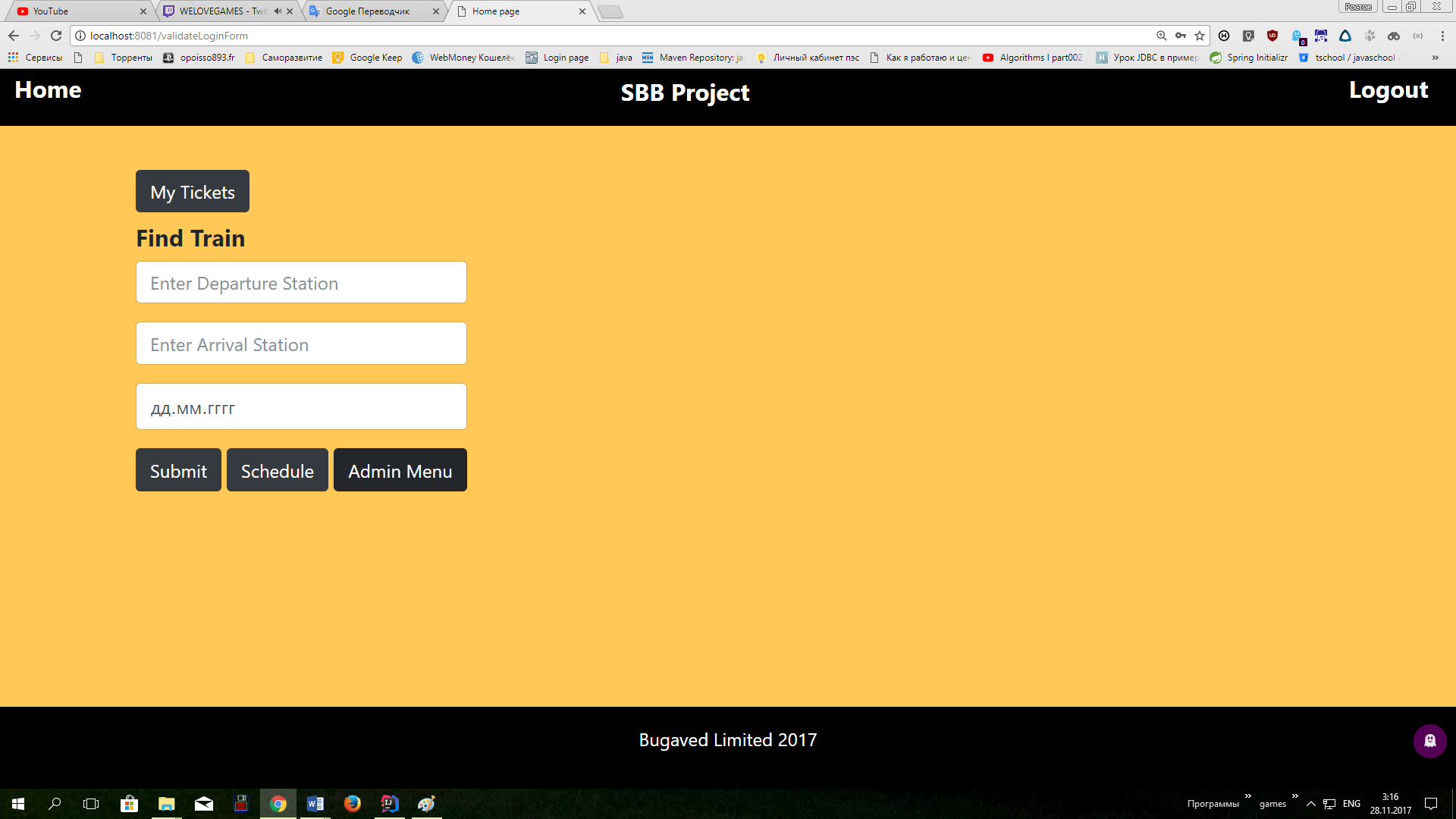
findWaypointByRouteAndStation – finds waypoint by route and station

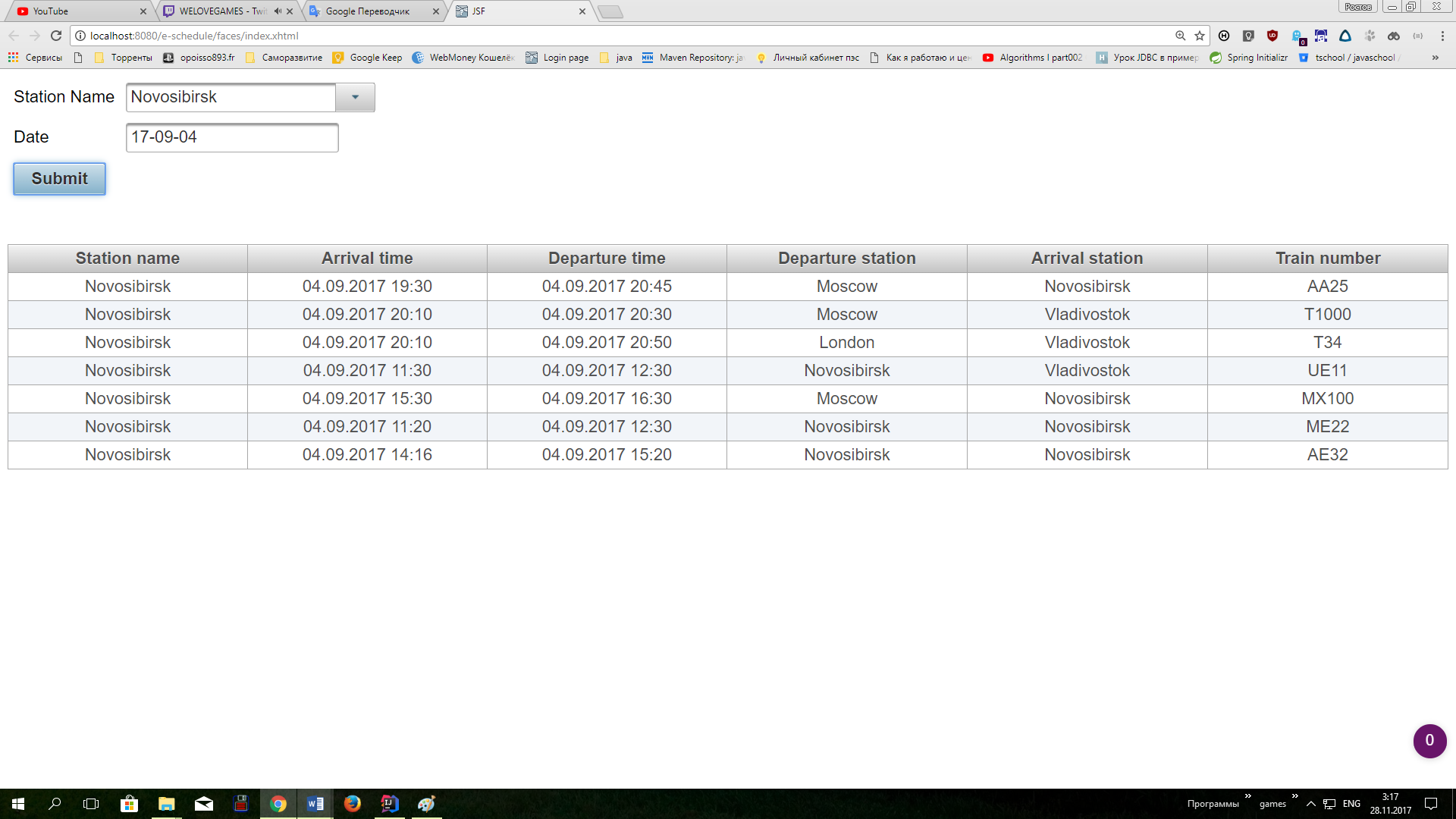
GenericDao is interface. AbstractDao is abstract and implements genericDao.

Every Dao extends AbstractDao, so they can inherit EntityManager

Transactions works with hibernate managed by spring.

10. Screenshots of applications





11. Unit Tests

**RouteServiceTest – tests every method of routeService:**

findExistingRouteTest() – find existing route

findNonExistingRouteTest() – find not existing route

createRouteTest() – create route

findAllRoutesTest() – find all routes in database

**StationServiceTest – tests every method of stationService**

findExistingStationTest() – find existing station

findNonExistingStationTest() – find non existing station

findAllStationsTest() - finds all stations in database

getStationScheduleTest() – finds schedule of station in required date

createStationTest() – create station

**TicketServiceTest – tests every method of ticketService:**

findExistingTicketTest() – find existing ticket

findNotExistingTicketTest() – find not existing ticket

findTicketsByUserTest() – find tickets with required user

createTicketTest() – create ticket

**TrainServiceTest – tests every method of trainService**

getExistTrainsByStationsAndDateTest() – find existing train between a and b station in required date

getNonExistTrainsByStationsAndDateTest() – find non existing train with required stations and date

findExistTrainByRouteTest() – find existing train with required route

findNonExistTrainByRouteTest() – find not existing train with required route

getAllTrainsTest() – get all trains in database

**UserServiceTest – tests every method of userService**

findExistUserByNameAndLastNameAndDateTest() – find existing user with required name, lastname and date

findNonExistUserByNameAndLastNameAndDateTest() – find non existing user with required name,lastname and date of birth

findExistUserByIdTest() – find existing user by id

findNonExistUserByIdTest() – find non existing user by id

persistUserTest() – create user

**WaypointServiceTest – tests every method of waypointService**

findAllWaypointsTest() – find all waypoints in database

createWaypointTest() – create waypoint