**Technical Solution Description**

CONTENT

[DESCRIPTION 1](#_Toc454324456)

[USED TECHNOLOGIES AND FRAMEWORKS 1](#_Toc454324457)

[SCHEME OF DATABASE 2](#_Toc454324458)

[MODULES OF THE APPLICATION 3](#_Toc454324459)

[FUNCTIONALITY 3](#_Toc454324460)

[IMPROVEMENT IN THE NEXT RELEASES 9](#_Toc454324461)

DESCRIPTION

Application “UberBahn” represents an information system of a company, which provides passenger rail transportation. It allows adding stations, routes and getting information about trains and registered passengers for its employees. Application provides clients with information about trains, station timetables and gives opportunity to buy tickets.

Application “UberBahn Reports” allows getting information concerning purchased tickets per selected period.

USED TECHNOLOGIES AND FRAMEWORKS

IDE – IntelliJ Idea 16.1

JDK 1.8

WildFly 10 Final

JSP/JSTL

MySQL Server 5.7/HSQLDB

Spring 4

Apache Maven 3.3.9

JPA 2.1 / Hibernate 5.1

Log4j 2

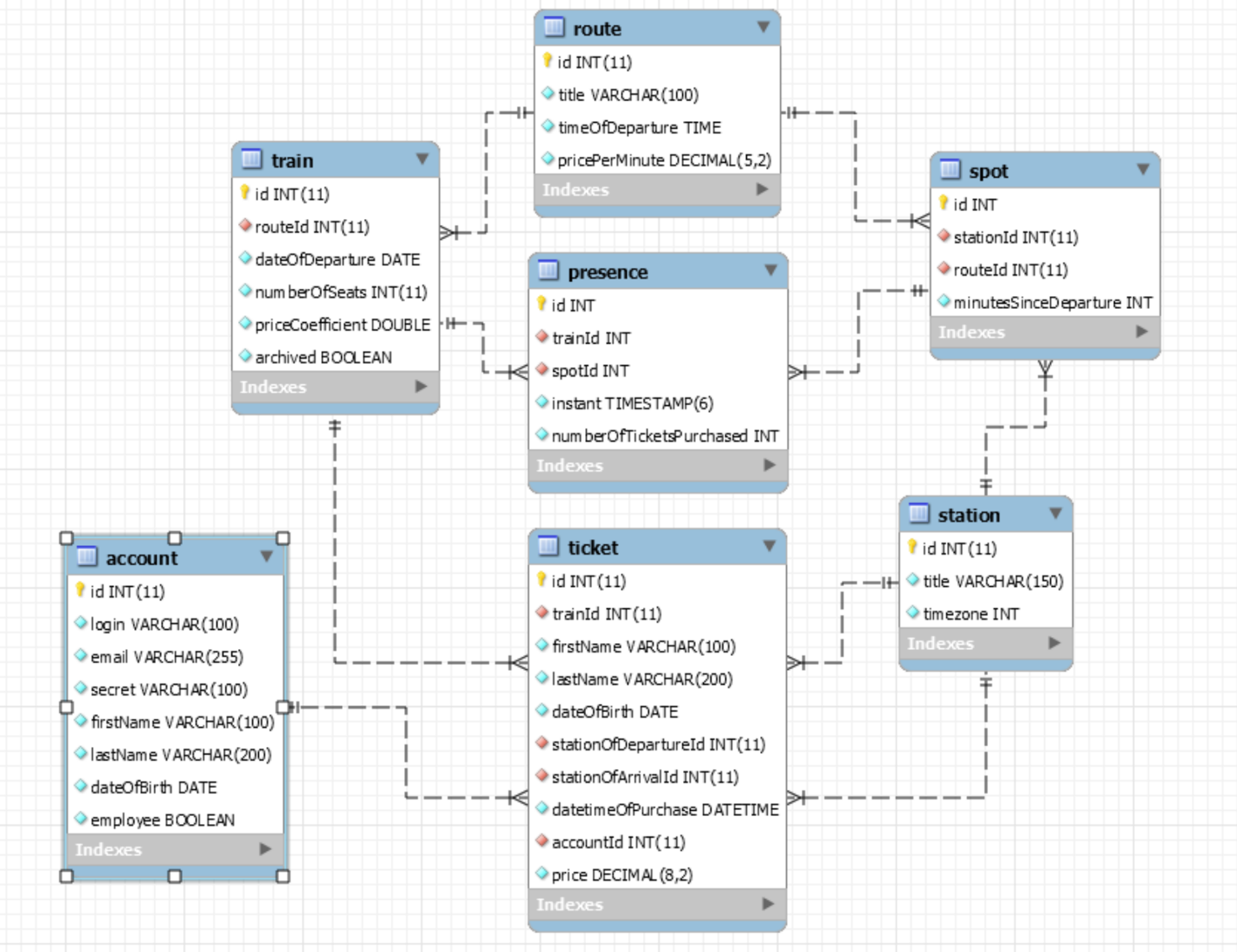
JUnit/Selenium

HTML, CSS

JavaScript, jQuery

EJB 3/JSF 2

SCHEME OF DATABASE



The scheme of database consists of 7 tables: train, route, spot, presence, station, ticket and account.

Each table includes primary key – id.

Table station contains information about station titles (unique), table route contains information about route title (unique) and time of departure, table account includes login (unique), email (unique), secret (or password), first name, last name, date of birth and information whether a user is an employee or not.

Table train contains information about route (relation between table train and table route Many-To-One), date of departure and number of seats. Relation between tables route and station Many-To-Many, to resolve this problem table spot is added, which includes routeId and stationId (relation between tables spot and route or spot and station Many-To-One), minutes since departure. Unique constrains are routeId and stationId, routeId and time since departure. Table ticket provides information about train (relation between tables ticket and train Many-To-One), passenger first name, last name, date of birth (unique constraint), stations of departure and arrival (relation between tables ticket and station Many-To-One), date and time of purchase and account (relation between tables ticket and account Many-To-One). Table presence includes information about train (relation between tables presence and train Many-To-One), spot (relation between tables presence and spot Many-To-One), instant, number of purchased tickets. Unique constraints are trainId and spotId.

MODULES OF THE APPLICATION

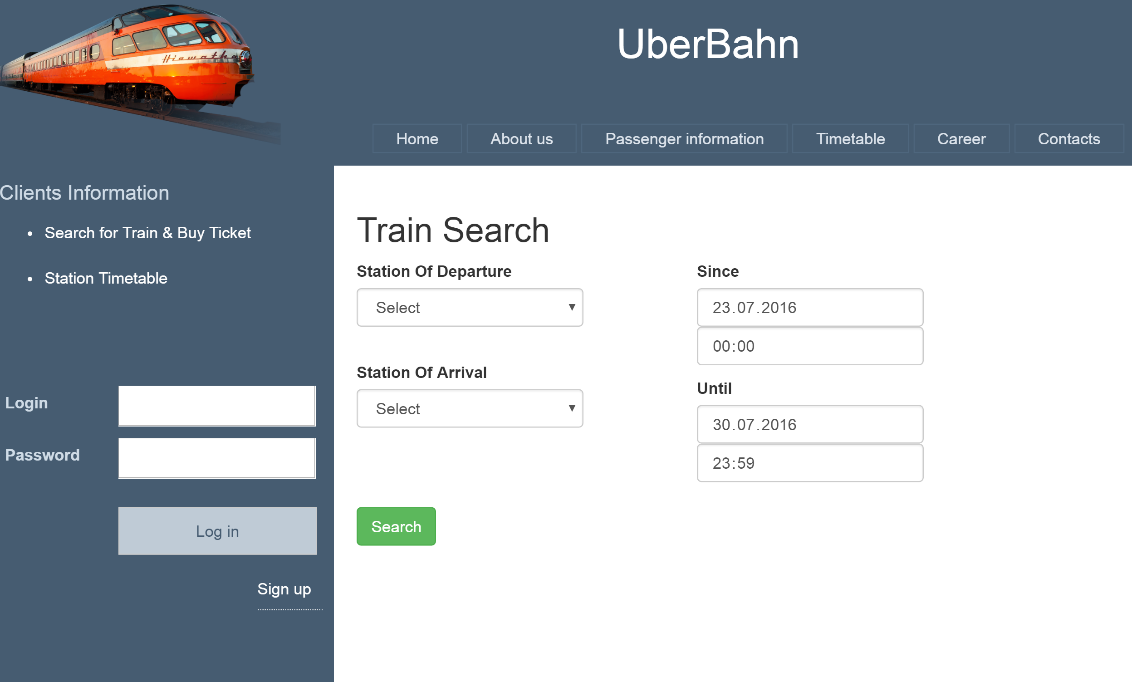
Application implements 3-tier architecture. View-tier represented by controllers, scripts; Business-Logic-tier by services, transports; Data-tier by repositories, entities.

Project consists of the following modules:

* Entities
* Repositories
* Services-API
* Services-Beans
* Transports
* Web-Main

Due to such architecture Dependency injection pattern implemented, which means that a class should not configure its dependencies statically but should be configured from the outside. Another pattern – Data Transfer Objects.

USER INTERFACE



Common parts – header, navigation, footer.

BUSINESS LOGIC

Services:

1. AccountService methods:

* create (creates new application user account, saves it in database and returns account details);
* getByLogin (gets account from database by application user login and returns account details);
* getById (gets account from database by its id and returns account details);
* existsLogin (checks whether account with such login already exists);
* existsEmail (checks whether account with such email already exists).

1. PresenceService methods:

* archive (marks trains which arrived at final station as archived, update them in database and drops their presences).

1. RouteService methods:

* getById (gets route from database by its id and returns route details);
* create (creates new route, saves it in database and returns route details);
* existsRoute (checks whether route with such title already exists);
* getAll (returns collection of route details for existing routes).

1. StationService methods:

* create (creates new station, saves it in database and returns station details);
* getTimetable (returns station timetable for selected period);
* getAll (returns collection of station details for existing stations);
* existsStation (checks whether station with such title already exists).

1. TicketService methods:

* create (creates new ticket, saves it in database and returns ticket details);
* getTicketInfo (gets ticket from database by its id and returns ticket details);
* countTicketsAvailable (counts number of tickets available for purchase);
* getTicketInfos (returns collection of purchased ticket details by account per period);
* getTicketInfos (returns collection of purchased ticket details per period).

1. TrainService methods:

* getAll (returns collection of train details for trains passing stations in specified period of time);
* create (creates new train, saves it in database and returns train details);
* getTrainInfos (returns collection of train details by route id);
* getPassengerInfos (returns collection of passenger details for train);
* existsTrain (checks whether train with such route and date of departure already exists);
* getByDepartureArrivalAndTrainId (returns train details by departure, arrival stations and train id).

Transport objects encapsulate data. They are used to transfer data between services and View-layer, contain information, displayed to users.

ENTITIES

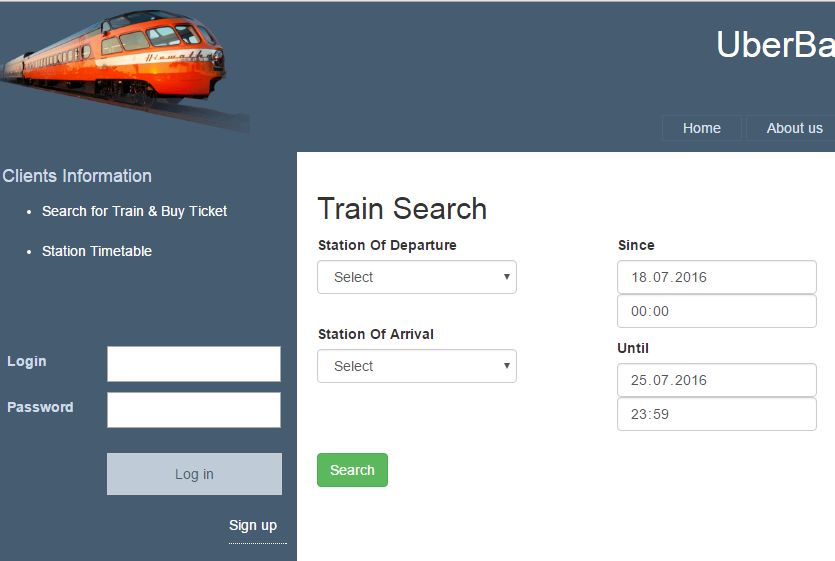
Repositories:

Entities display the structure of database diagram and relations between tables.

FUNCTIONALITY

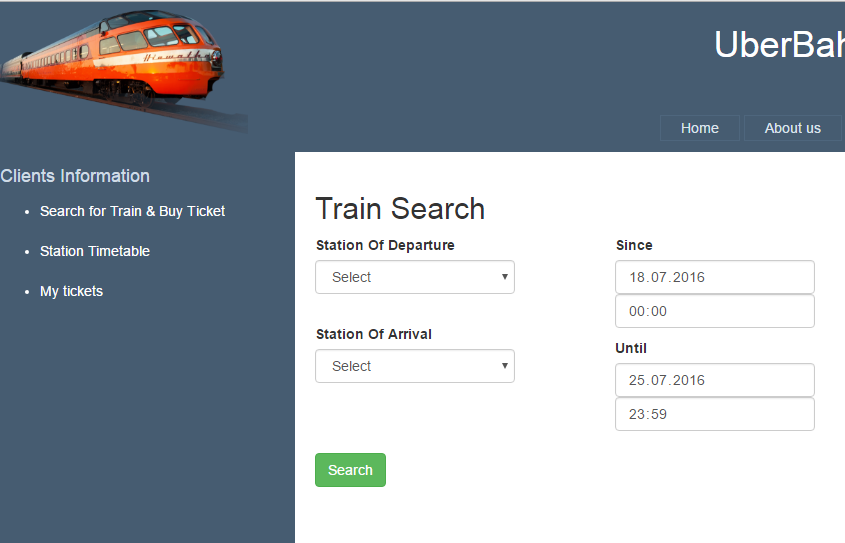
For unregistered users:

* find train from station A to station B;
* get timetable per each station;
* sign up.



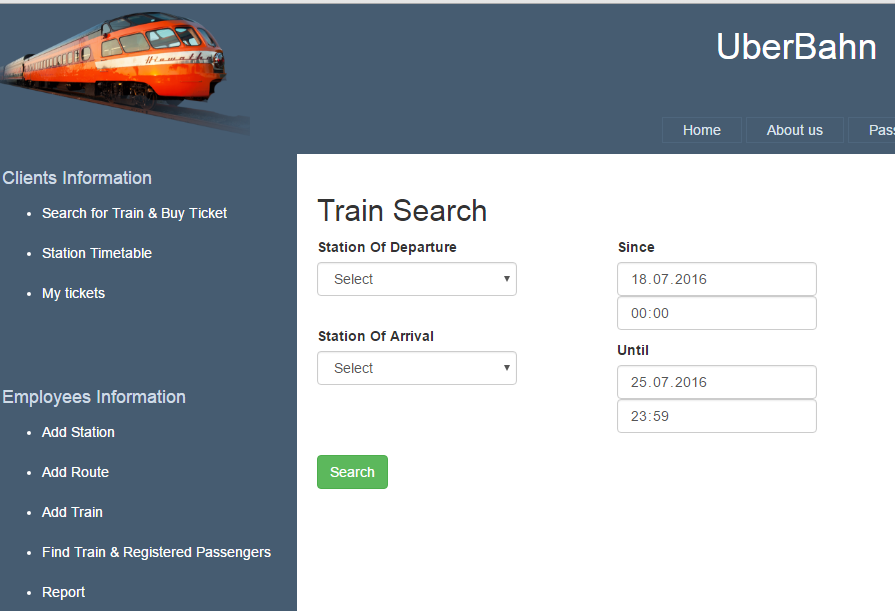
For clients:

* buy ticket (include train search);
* get information about purchased tickets.



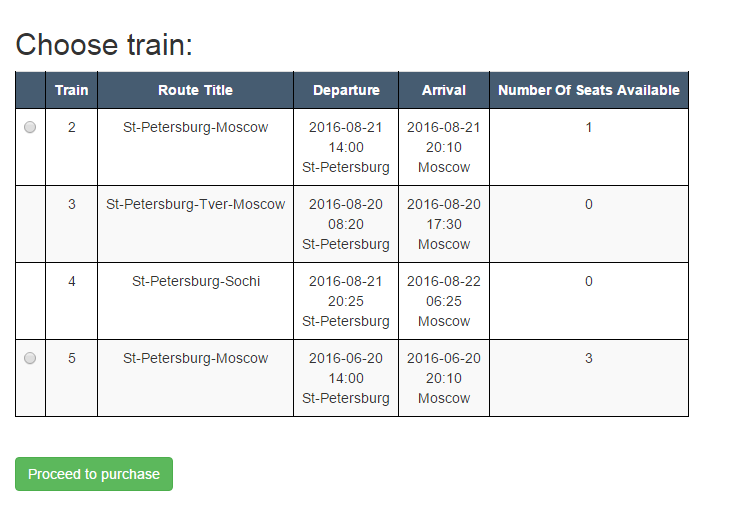
For employees:

* add new stations, routes and trains;
* find trains by route;
* view registered passengers (include train search);
* get reports concerning purchased tickets.



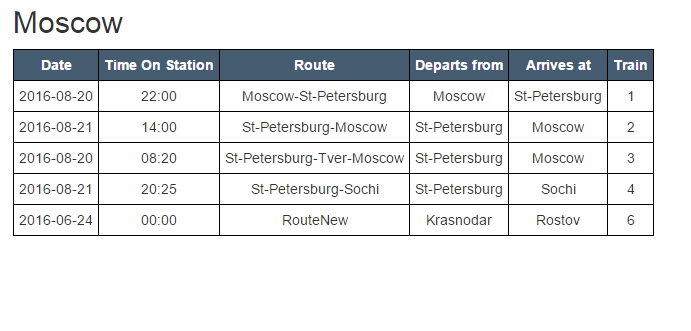
The majority of operations with information system are associated with train search. The main page of the application gives opportunity to find trains departing from station A and arriving at station B in specified period of time.

Second screen shows trains, number of available seats and allows to choose train and follow the link in order to buy ticket.

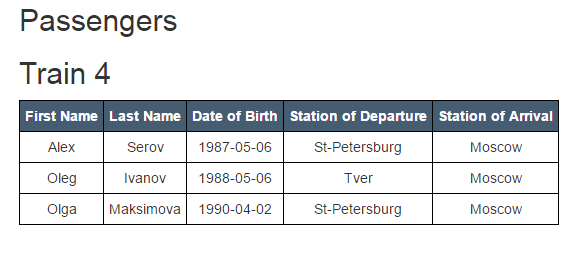


Purchasing ticket includes entering passenger information (first name, last name and date of birth).

Client may also choose station, enter period of time and find trains passing the station.



Employees may add new stations, routes and trains. Application also provides them with information about trains, following each route. After choosing train employees have an opportunity to go to the next screen and view registered passengers.



IMPROVEMENT IN THE NEXT RELEASES

- Changing scheme of database in order to optimize queries

- Adding time-zones

- Exception handling and logging

- Unit-tests

- UI for different types of users

- Dependency Injection pattern