Content

[1. Project goals 1](#_Toc18922811)

[2. Technologies and frameworks 1](#_Toc18922812)

[3. Additional features 1](#_Toc18922813)

[4. Database scheme 2](#_Toc18922814)

[5. Model implementation 3](#_Toc18922815)

[6. Modules and their interaction 3](#_Toc18922816)

[7. UI 3](#_Toc18922817)

[8. Services 4](#_Toc18922818)

[9. DAO 4](#_Toc18922819)

[10. Screenshots 5](#_Toc18922820)

[11. Junit tests 6](#_Toc18922821)

[12. Logging 7](#_Toc18922822)

# Project goals

Main goal of first part was creating a railway information system, in which rail road employees could add information about trains and passengers could retrieve this information and purchase ticket.

Main goal of second part was creating app which represents a timetable. It should retrieve data from rest-api of first app and update timetable whenever changes happens.

# Technologies and frameworks

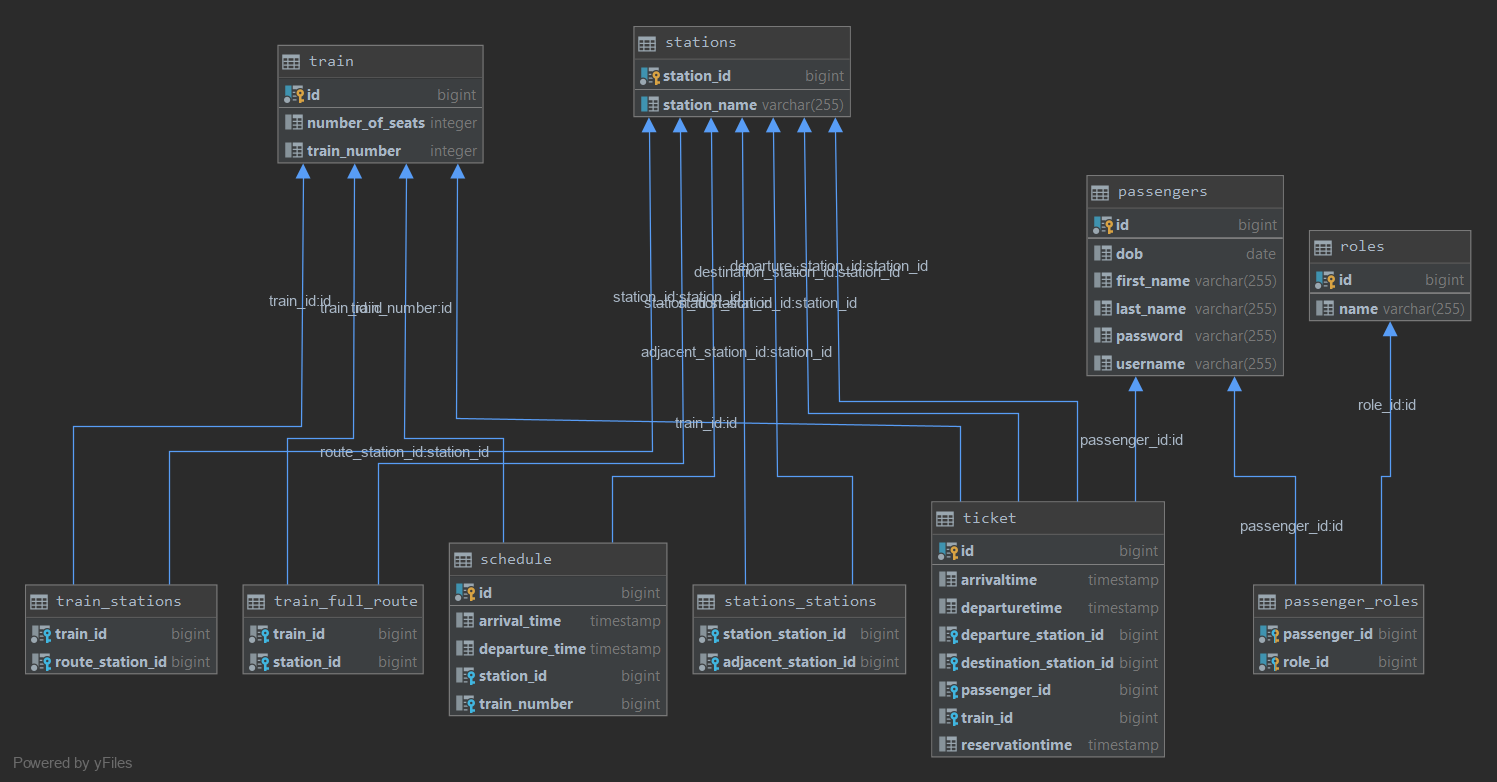
* 1. First Part:
     + Application server – Tomcat
     + Database PostgreSQL
     + Build tool – Maven
     + Spring framework
     + Hibernate
     + Mockito + JUnit
     + Log4j
  2. Second Part:
     + Application server – WildFly
     + Build tool – Maven
     + EJB
     + JSF
     + Primefaces
     + ActiveMQ
     + WebServices(REST)

# Additional features

Added opportunity for passengers to find complex routes between two stations (if there are no straight routes) select suitable route, and if passenger wants, purchase tickets for two trains at once.

# Database scheme

|  |  |
| --- | --- |
| Table name | Description |
| **trains** | Core information about train. |
| **stations** | Core information about stations. |
| **passengers** | Core information about stations |
| **train\_stations** | Utility table which connect tables ‘train’ and ‘stations’. Contains information about stations where train stops. |
| **train\_full\_route** | Utility table which connects tables ‘train’ and ‘stations’. Contains information about all stations train passes. |
| **schedule** | Connects tables ‘train’ and ‘stations’ and contains time when train arrives to station and departure from it |
| **stations\_stations** | Utility table. Contains information about adjacent stations |
| **ticket** | Contains all information about ticket |
| **roles** | Contains roles |
| passenger\_roles | Utility table. Connects tables ‘passengers’ and roles, contains information about passenger roles |



# Model implementation

Station has unique id, name, list of adjacent stations and list of schedules of this station.

Schedule has id, station, train, and time when train arrives to station and departures from it.

Train has id, list of stations where train stops, named route, and list of all stations train passes named full route. Also train has list of schedules.

Ticket encapsulates all meaningful information such as: train number, departure time, arrival time, passenger name, departure station and destination. Also, ticket has reservation time. When passenger clicks “purchase ticket”, but yet didn’t confirmed a purchase, ticket becomes reserved, and if passenger will change his mind and cancel purchase, it will be reserved for a short time and then become free again.

Passenger is just a passenger: username and password for authentication, first name, last name, date of birth and role for authorization.

# Modules and their interaction

First module represents a railway information system and has a rest-api which provides information about station schedules on the current date. Second module represents a timetable of station. It has rest client to retrieve information from the rest-api of the first module. Modules communication implemented through Active MQ. Fist module creates a topic and every time schedule changes sends a name(s) of station(s) where changes happens to this topic. Second application subscribes to this topic and, each time it receive a message from topic, requests rest-api of the first app and get updated timetable.

# UI

It was my first experience in creating UI. So nothing special, I just added some css to my forms and tables to make it look not so ugly-default, put navigation bars to separate jsp file and import it everywhere it was necessary. Pagination was added to tables, and in “My Tickets” I add lazyloading to list of tickets.

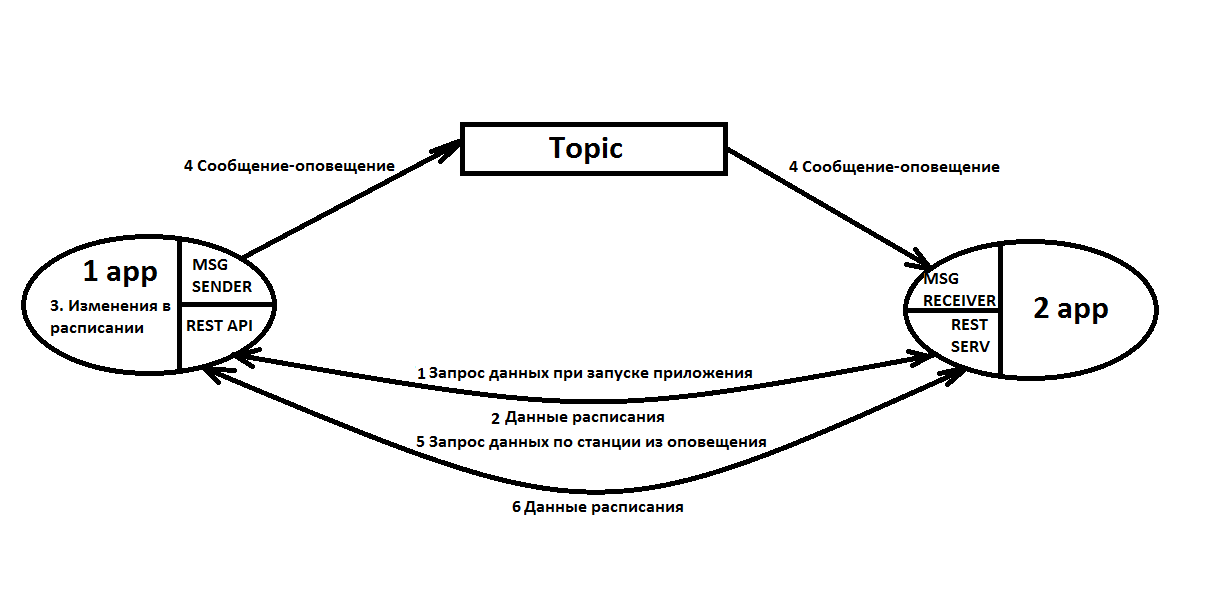
# Services

|  |  |
| --- | --- |
| Service name | Description |
| **PassengerService** | Implements passenger search and registration operations. |
| **ScheduleService** | Schedule searching/saving operations |
| **StationService** | Searching stations/ routes between stations etc. |
| **TicketService** | Ticket searching/checking purchase conditions/purchasing/reservation operations |
| TrainService | Train searching/adding/cancelling/route searching operations |

# DAO

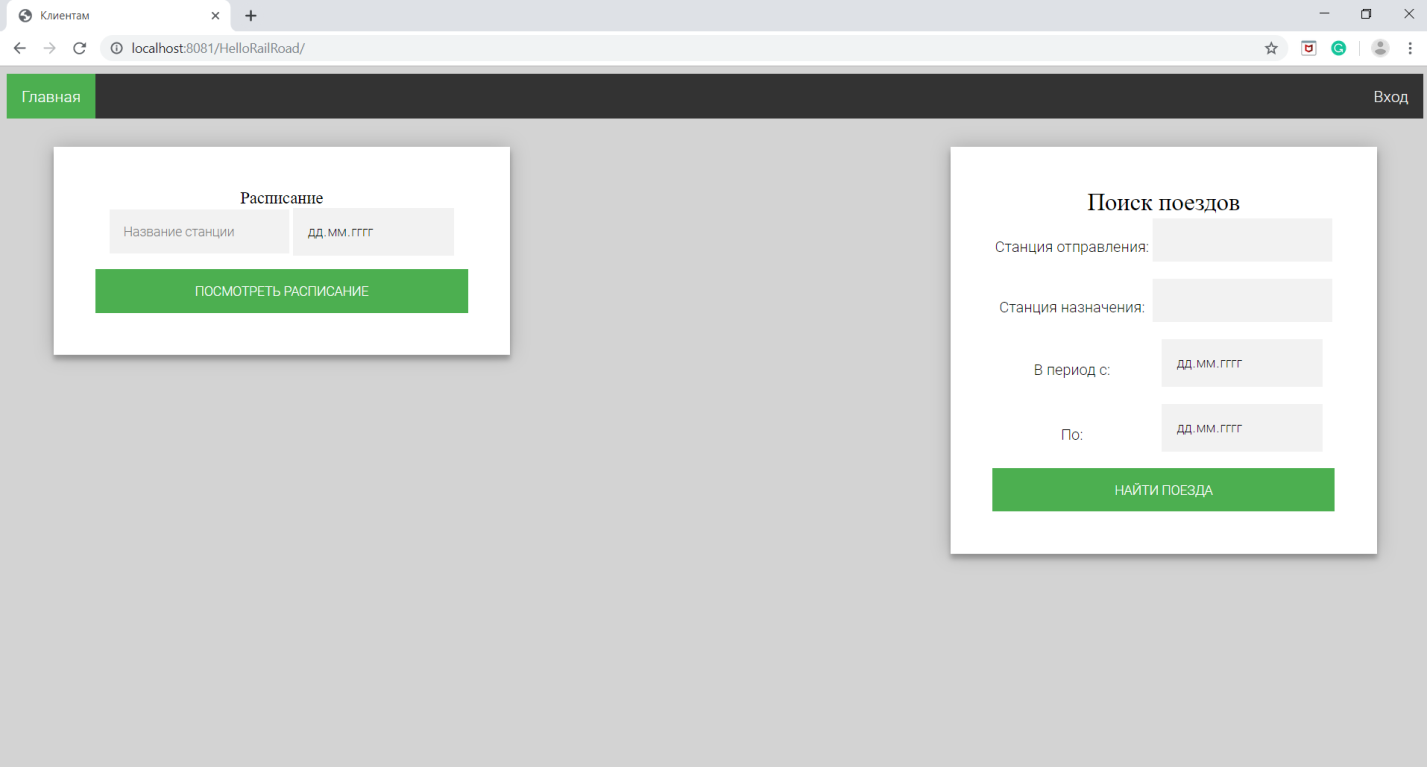
There are 6 entity classes: Passenger, Role, Train, Station, Ticket, Train, Schedule. All of them mapped to the tables in database. Controllers interacts with services, services requests data from DAO layer, process it and return to the controller. Transaction handling provided by spring framework, by using @Transactional.

# Applications interaction scheme

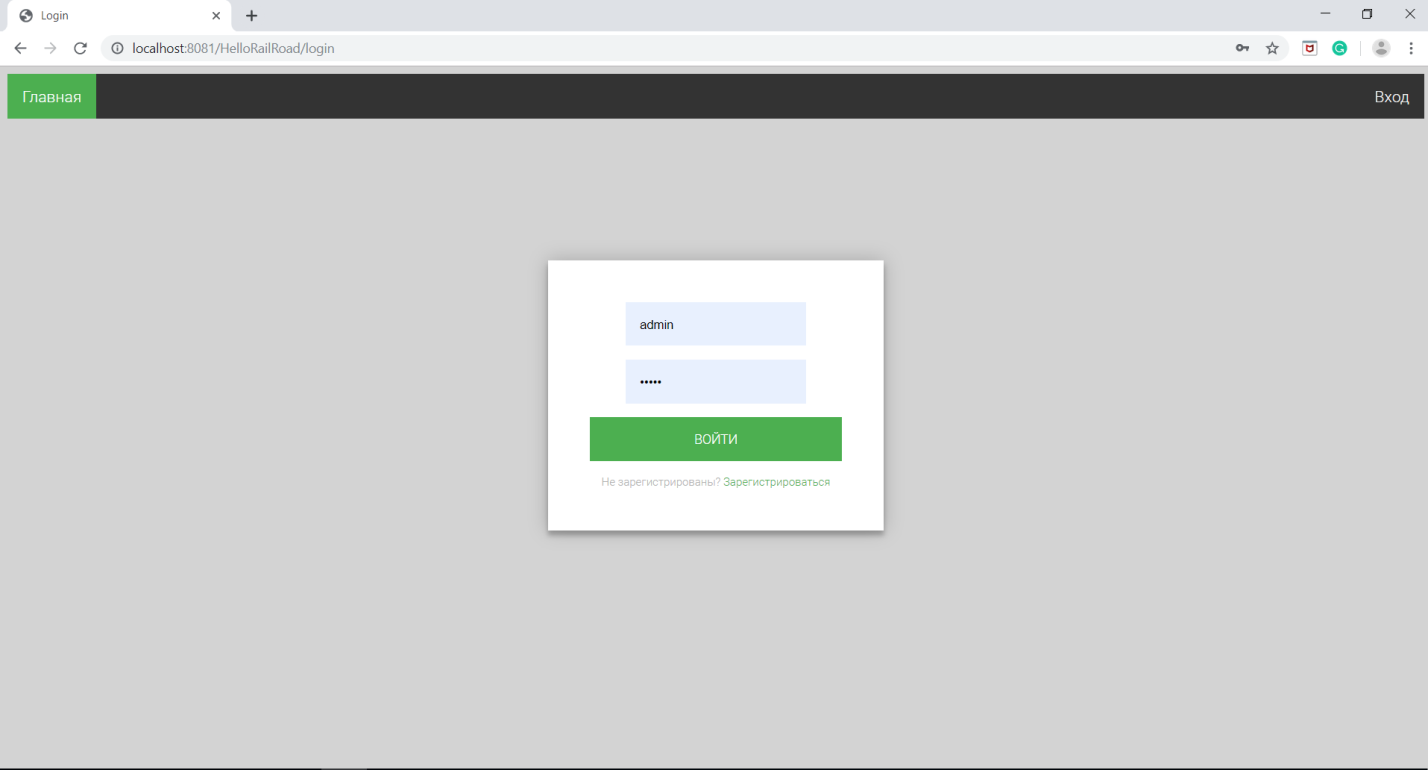


# Screenshots

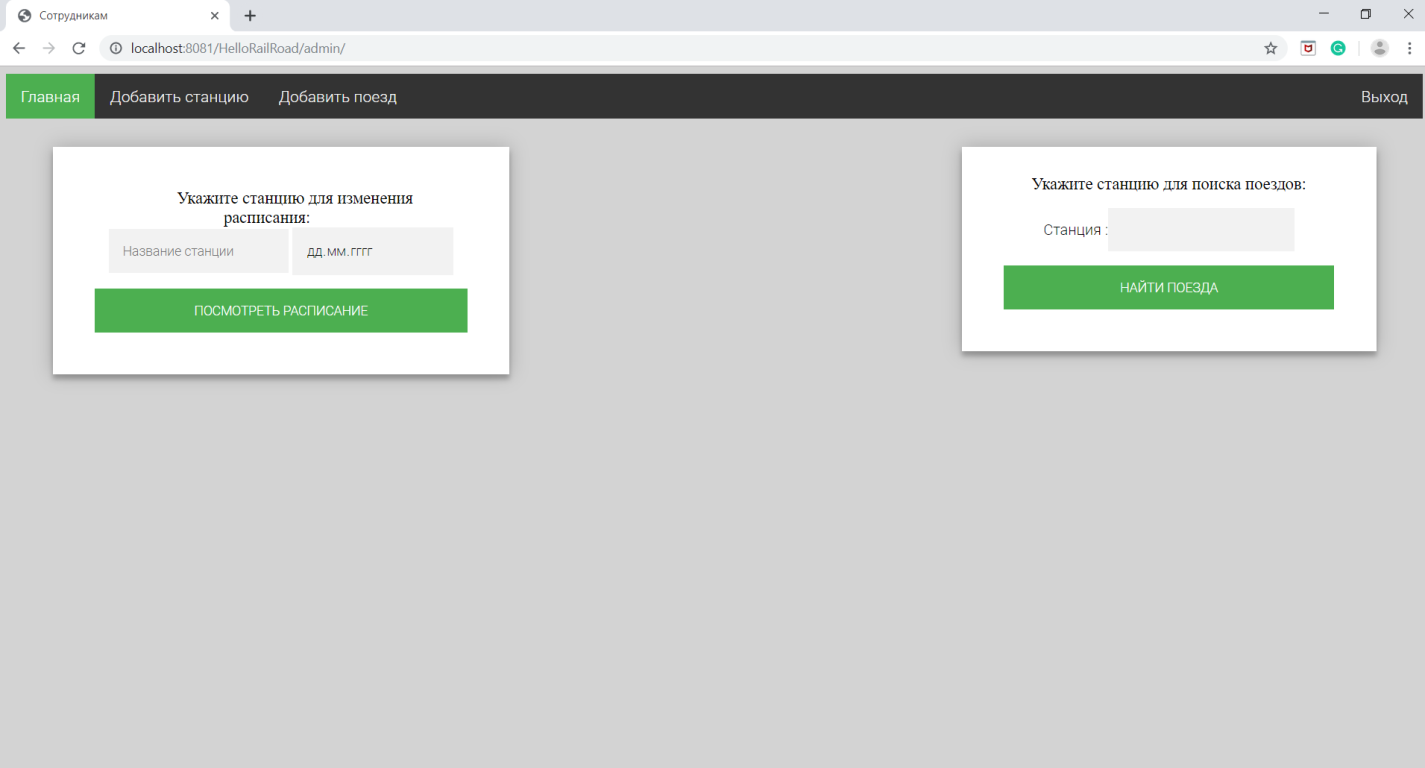
Home page for clients:



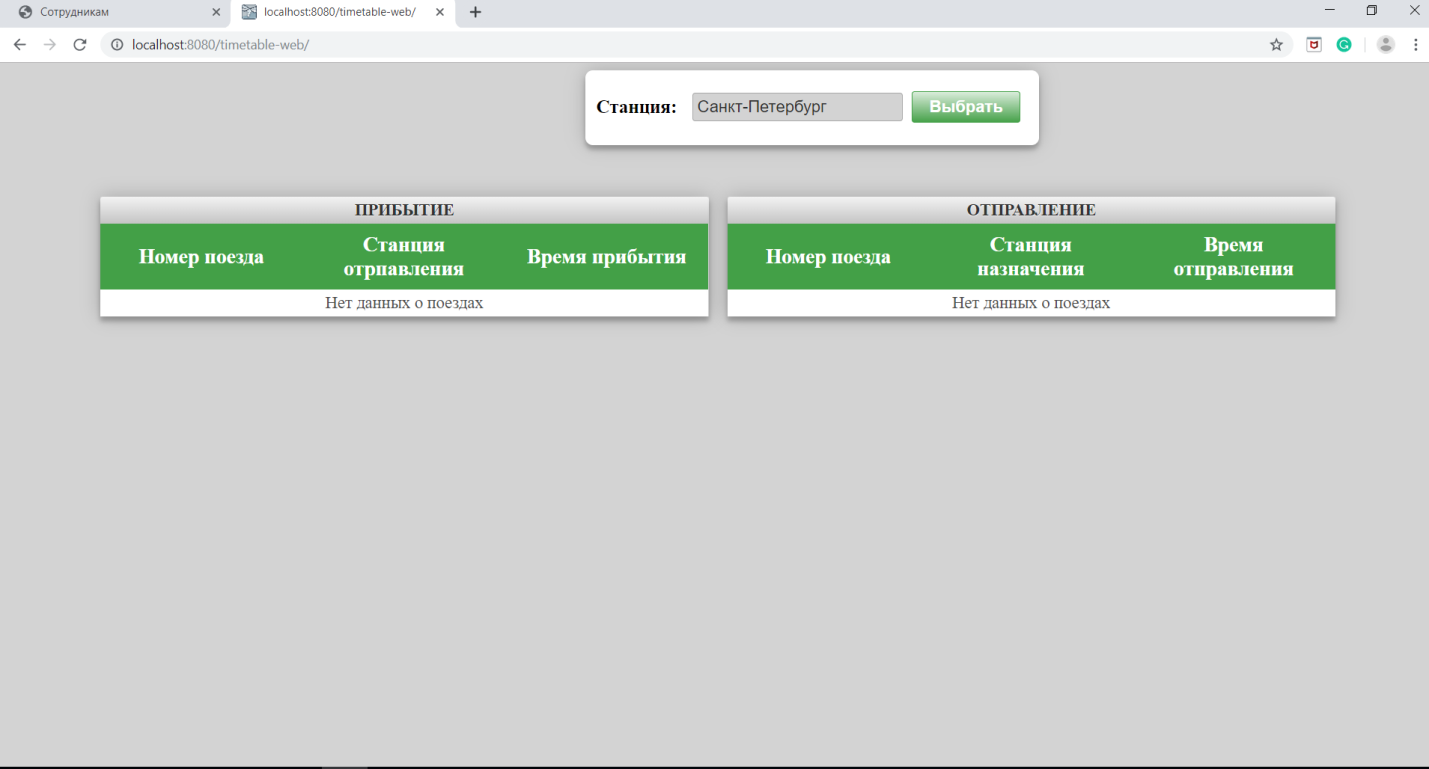
Login page:



Employee home page:



Second app main page:



# Junit tests

PassengerServiceImplTest – testing methods of PassengerService.

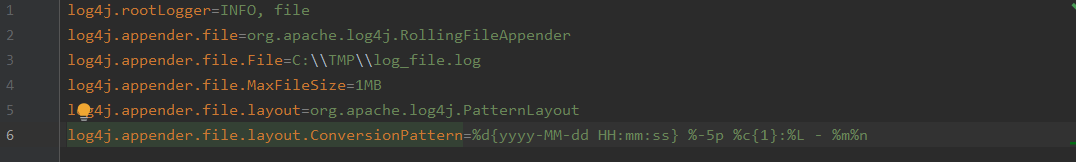
StationServiceImplTest – testing methods of StationService.

TicketServiceImplTest – testing methods of TicketService.

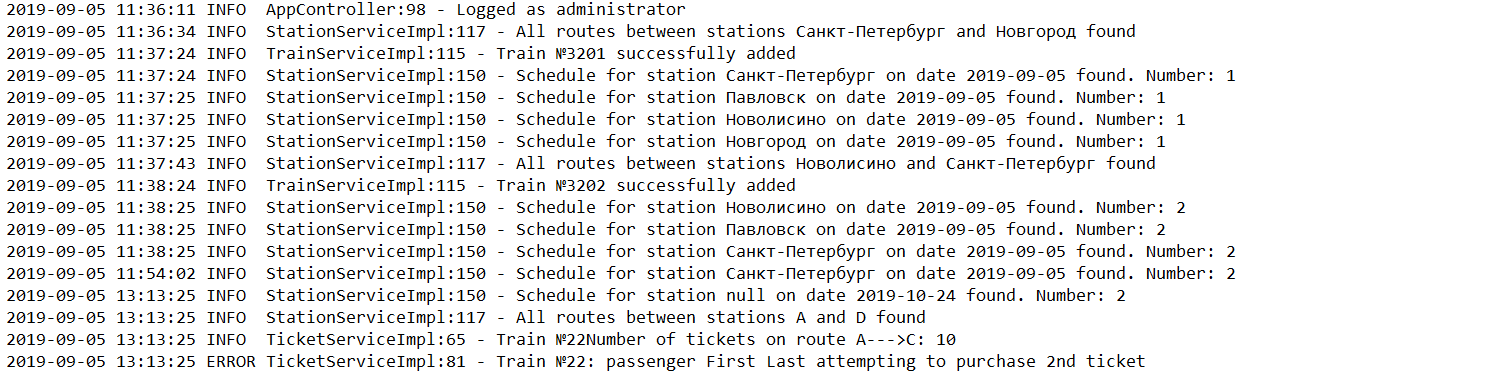
TrainServiceImplTest – testing methods of TrainService.

# Logging

I used log4j, here is my logging configuration



And here is example of log:



# Sonar Screen

