**Technische Universität München**

**Department of Informatics**

Interdisciplinary Project

2014/2015 Winter Term

**Developing, Integrating and Visualizing a Driver Scheduling Web-Platform Connected to Demand Forecasting Logic**

**28.02.2015**

|  |  |
| --- | --- |
| **Supervisor** | : Christian Ruf  : Alexander Döge |
| **Submitted by** | : Mertcan Yigin 03646727 mertcan.yigin@tum.de |
|  | : Emin Guliyev 03647136 emin.guliyev@tum.de |
|  |  |
|  |  |

Table of Contents

1. Abstract 3

2. Project Overview 4

2.1 Introduction 4

2.2 Purpose of The Web Platform 5

2.3 Scope of The Web Platform 5

2.4 Functional Requirements 6

2.4.1. User Stories 6

2.4.1.1 Admin Calendar User Stories 6

2.4.1.2 Driver Calendar User Stories 7

2.5 Non-Functional Requirements 8

3. Solution Description 9

3.1 Dependencies 9

3.2 System Architecture 10

3.2.1. Models10

3.2.2. Controllers 11

3.2.3. Views 16

3.3 Persistent Data Management (DB Architecture) 17

3.4 Demand Forecasting 18

4. Future Work 19

5. Conclusion 20

6. Screenshots 21

6.1 Example Report Files 21

6.2 Admin User View 22

6.3 Driver User View 26

7. References 28

8. Documentation 28

8.1 Build 28

8.2 TODO for future 29

8.3 Contents 30

**1. Abstract**

In this Internet world ordering from the Internet gain so much importance. It is so simple that it may be done siting in front of the computer. People do not need to go to the shops and carry all the stuff to their home. Almost every seller now has this online order possibility. They get the order from Internet and deliver the goods. Although taking the order is not so complicated, the delivery part is emerging a new market. Seller wants to deliver as much as goods to customers and the buyer want to get it as soon as possible. This duality is not always so simple to find a solution.

Generally sellers either use their own delivery systems, which are suitable if they are giving their service to a small area. Thus for bigger seller companies this solution is not feasible every time. That is why they sometimes make arrangements with local delivery companies. Those delivery companies are the bridges between buyers and sellers. They carry the goods from one point to another point. They are taking care the delivery problem so that they assign their drivers in an efficient way.

VOLO is the company that deals with the delivery of the goods between sellers and buyers. They have their own drivers and assign them according to orders. However there are of course some challenges that they face with. They have to keep track of their available drivers and their available times. Moreover they do not know how many drivers they need to have at specific time and date. Since the daily delivery is not same for 365 days. That’s the reason they also want to have a demand forecasting system to anticipate how many drivers they need for the upcoming days.

The goal of the project is not only to develop a stand alone web platform with Angular.js technology to take care of the driver calendar and also create a demand forecasting algorithm based on previous demands.

**2. Project Overview**

**2.1 Introduction**

VOLO is a smart local delivery network that uses technology to organize a crowd shipping fleet. This crowd approach allows for a flexible availability of drivers according to capacity required. In this IDP project, the goal is adding a web platform to the VOLO system, which connects to their backend, provides calendar to drivers to select their available times and also provides second calendar to company to assign drivers to delivery. Furthermore this project also contains a forecasting algorithm that helps company to arrange the number of fleet they need for a specific day to optimize and maximize the profitability.

**2.2 Purpose of the Web Platform**

Before our application driver scheduling is done on a very manual basis with a goggle docs sheets where all of Volo’s drivers enter their times of availability for one week. Each new driver has to be added manually for each day and week and there is only little overview about how many drivers work each shift and whether it is sufficient for the expected demand that week. In addition to this, the demand planning is also very basic and only based on some very limited assumptions.

The purpose of this web platform is to provide an easy and efficient way to enter their times of availability. Also it is required to visualize and to give some overview to the company relating the times entered by the driver. In addition to that an improved demand forecasting should be implemented. This forecasting should be based in historic data from the company’s database.

**2.3 Scope of the Platform**

The platform is implemented as a separate stand-alone project to give an ease to use for drivers just opening it directly from one link and also not to create overhead to the existing web platform of the company. Although it is stand-alone project, the platform should connect to the existing database, take the information from there and also perform update operations. In addition to this the platform also gives functionality to support user management for the admin user.

As an extra to the user management functionality in the web application, we also provide a reporting feature for admin users. There are 2 types of reporting in the system. One of them is the overview of the drivers that are involved in a specific month. The second one is the information table of the drivers in the system.

Volo Calendar web application is implemented by Angular.js framework and Postgres database, will be deployed to Heroku. The reasons for that are both to keep integrity with the existing backend the company has and to follow new developments on the web environment.

**2.4 Functional Requirements**

In this part, the functional requirements of the platform will be explained. The user stories for the platform cover all the functional requirements and clarify them. These user stories also explain the scope and the purpose of the platform in detail. The user story will be divided into 3 parts to keep it easy to understand: admin calendar user, driver calendar user and demand forecasting algorithm.

**2.4.1 User Stories**

**2.4.1.1 Admin Calendar User**

* Admin User can login via his existing credentials for the Volo.
* Admin User is determined as an Admin and the designated page and functionality are shown.
* Admin User can see all the drivers in the database in Manage Users page.
* Admin User can add a new driver in Manage Users page.
* Admin User can choose the drivers working status either as a Mini-job for a contracted drivers or Flexible drivers for not contracted drivers.
* The required time that drivers need to fulfill and the payment is changed automatically according to his status. All the calculations are done differently.
* Admin User can mark a driver as deleted but it is not deleted really. The driver is shown as a “deleted” driver in Manage Users page.
* Delete/Edit/Create functionality on driver users should be in corporate with the API.
* Admin User can see more detailed information when the driver user is clicked.
* Admin User can also edit all the information stored for the driver user.
* At the driver users list the name, telephone, e-mail, contract status, active/inactive status, planned working hours, finished working hours, is done working hours, the difference in working hours can be seen as a table in Manage Users page.
* This Admin User Calendar page supports all the filtering, searching, paging functionality that comes from the API.
* In Admin User Calendar view, the admin can see the entire user that is registered for the specific time and date.
* Admin User can determine the maximum number of driver that is needed for the next week for every hour.
* Admin User can remove a driver from specific time and/or assign a new driver.
* Admin User can get the report for specific month to see an overview about drivers and how many hours they fulfilled in that month.
* Admin User can get the report to have the information of the all drivers in one table as a .pdf file.

**2.4.1.2 Driver Calendar User**

* Driver User can login via his existing credentials for the Volo.
* Driver User is determined as a Driver and the designated page and functionality are shown.
* Driver User sees the calendar for the next week. This calendar view can be changed to any other week for upcoming 3 months.
* Driver User can choose his available time for the specific week.
* Driver User can see his already selected times and de-select if necessary.
* Driver User can see how many hours he needs to fulfill, how many hours he already has fulfilled and how many hours he selected.
* Driver User can save his available times also for next week or for next month or for forever. So that the time he chooses automatically updated according his choice.
* Driver User can see his profile page and also edit the page.
* In his profile page the necessary information needed by the company is stored. For example; name, mailing address, banking numbers, telephone number, the vehicle he has etc.

**2.5 Non-Functional Requirements**

* **Usability**: An employee of Volo should be able to use the platform without any user manual. The platform should have the same logic as the previous Excel sheet system.
* **Reliability**: There should not be any inconsistency between the data that is provided in the platform to the user and the data in the database.
* **Performance**: When the Driver User selects and saves the time that is chosen should be saved in the database without any performance issue because another driver can choose the same time.
* **Implementation**: The implementation of this web module is decided to be in Angular.js. To both work as a stand-alone project but also integrated with the existing system the backend is written by Java Sprint MVC. Also to keep the consistency with the real database the database that is used while developing is created with Postgres.

**3. Solution Description**

In solution description part, the technical background and details of developed platform will be explained.

**3.1 Dependencies**

As stated before this project is developed with Angular.js by Google. Angular.js is an open-source web application framework maintained by Google to address many of the challenges encountered in developing single-page applications. Its goal is to simplify both development and testing of such applications by providing a framework for client-side model–view–controller (MVC) architecture, along with components commonly used in rich Internet applications. MVC structural pattern includes three separate parts.

Model is the first part of the MVC. Model is the data that the application needs. In this project, each driver and calendar fields in database has a model definition in the application. By the database functions, controllers can manipulate the model (data).

View is the second part of the MVC and what the user sees in the screen. User can see modals (data) as outputs and be notified by changes. Angular.js helped us to create these pages as HTML pages (views). Angular.js lets us extend HTML vocabulary for our application. The resulting environment is extraordinarily expressive, readable, and quick to develop.

Controller is for The C letter of MVC. Each model has its own controller and the controller manipulates/changes/updates the model (data). Each controller should be provided with some predefined functions to the controller. These functions are used CRUD operations and URLs for the views. By the help of the REST API and the connection with the backend via Java, all changes made by the controller in the model are applied also in the database at the same time. It provides a complete data consistency between application and database. For all URLs and all functions, the related path should also be added to route file.

**3.2 System Architecture**

**3.2.1 Models**

The platform has two important data definitions (models), which are “User” can be an admin or a driver, “Order” which is used for demand forecasting. User model points a user that is created by the Volo company for logging in to the platform. The “Order” model has the order that Volo received. These are the persistent data is stored in the database. Additionally there are many different models that are kept for the View Models to show different information to the user. Since that models are not stored persistently in the database and does not important role for the platform these models are excluded.

* **User:** User model has the basic attributes to identify the user like e-mail and password. To determine that the user is an admin or not there is a field “admin”. The rests are the field that are required by Volo to reach the driver via mailing address and also for payments it keeps the banking information.
* **Order:** This data model stores the required information of the orders. This information will be used later for the future demand forecasting based on the date, time and the location of the orders. Moreover clusters can be created by this information and can improve demand forecasting. Moreover to work better on this information we have sub model that is OrderDayStatistics. This field keeps how many orders have been ordered on which time in the day. There are number of orders are kept for 0-24 time interval in this information.

**3.2.2 Controllers**

The most important parts of the developed platform are the controllers which modify and manipulate views and models.

The “Admin Calendar Controller” has some methods that operate CRUD methods for the admin calendar view.

* **Get Calendar View Model:** This method queries the last three months. This is required by Volo to see the last three months values on the calendar. Customizes and initializes the calendar view via using these values.
* **Get Month Statistics For Admin User:** After getting the last three months and initializes the calendar, via this controller the values are inserted to the calendar to show it to admin.
* **Get Admin Calendar Week:** This method gets the current week and the required drivers values for the days that come from the forecasting.
* **Get Detailed Admin Day Statistics:** This method gets the assigned drivers for the week.
* **Insert Driver Day Statistics:** This insert method allows admin user to add new drivers to the week manually.
* **Delete Driver Day Statistics:** This delete method allows admin to delete drivers that are assigned to week.
* **Insert Or Update Detailed Admin Day Statistics:** This method ensures to save the changes that are done by admin user on the calendar.
* **Forecasting Controller:** This controller shows the manual forecasting to admin user. Also this controller makes available for admin user to interact with forecasting algorithm. Admin user gives the parameters to this controller and this controller returns the result of the forecasting and shows it in the calendar.

The “Driver Calendar Controller” has also some methods that operate CRUD operations for the driver calendar view.

* **Get Month Statistics For Driver User:** This method gets the statistics of the driver. Drivers according to their contract have to fulfill some hours for every month. Assignments are allowed if they have left hours for this month.
* **Get Driver Calendar Week:** This method gets the detailed information for the week. Gets the dates if that driver has already been assigned.
* **Insert Or Update Driver Calendar Week:** This method ensures to save the changes that are made by the driver.
* **Set Next Week Calendar For Driver:** This method allows driver users to save the selected dates and times also for the next week.
* **Set Next Monthly Calendar For Driver:** This method allows driver users to save the selected dates and times also for the next month.
* **Set Next Annual Calendar For Driver:** This method allows driver users to save the selected dates and times also for the next year.

The “User Management Controller” has also some methods that operate CRUD operations for the driver calendar view.

* **Get Users:** This post method quarries the entire users in the system with a pagination feature.
* **Get User By Email:** This get method gets the one exact user specified by the email address.
* **Get User By Id:** This get method gets the one exact user specified by the id.
* **Insert Or Update User:** This post method puts the updated or newly created user to the database of the Volo.

These controllers manipulate and modify the models by the help of the URL addresses. The spring MVC framework has to be notified about the URL addresses (paths). Because of that, some modifications should be applied in the route. We specified in the spring MVC which path creates a POST or a GET HTTP request. In the following figure, the request definitions in are shown for the developed platform.



Figure 01: Admin Calendar Controller

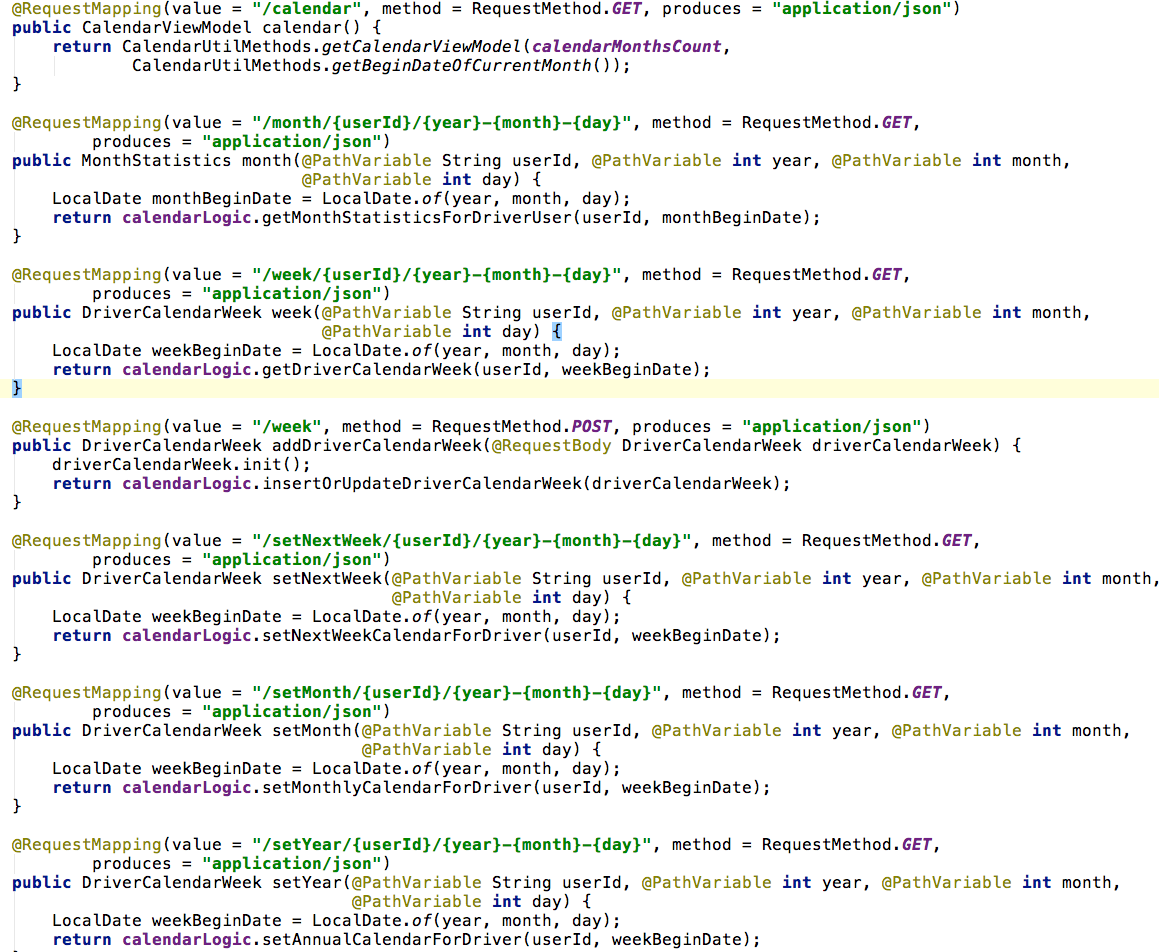


Figure 02: Driver Calendar Controller

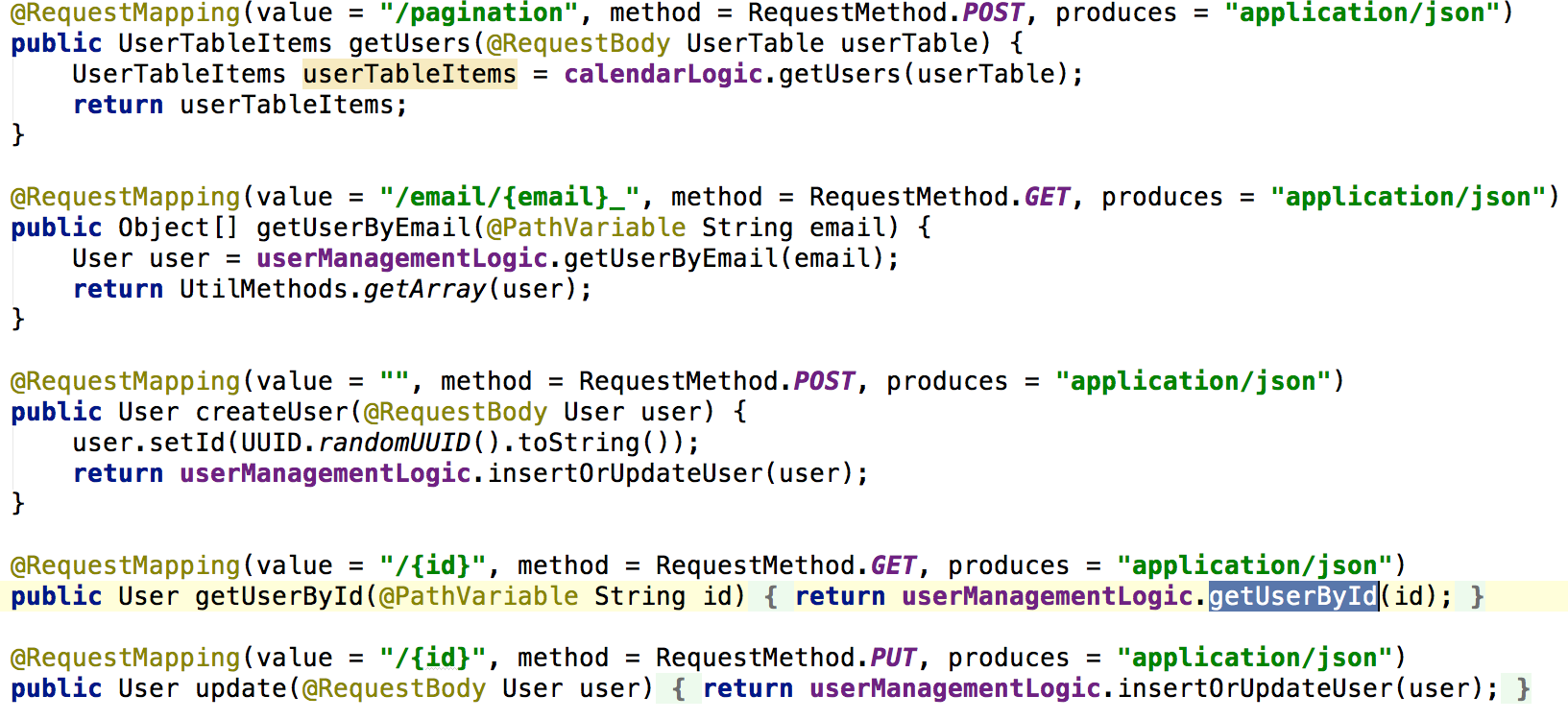


Figure 03: User Management Controller

**3.2.3 Views**

The platform has also some view specifications. In this platform, there are mainly 2 views. One of them is for admin user; the other one is for driver user.

**Admin View**

* **Calendar View:** The calendar view is the first view that welcomes the user when he logs in. Because of that, this view can be considered as the main view of the platform. Here admin user can see the months, assigned drivers and also missing drivers for the days. So that admin can add new drivers if needed.
* **Forecasting View:** This view shows the potential needed drivers for the week. The data comes from the demand-forecasting algorithm. But of course can be manipulated by the admin.
* **Users View:** Here admin user can see the users that are registered to the system. He can search, filter and paginate the users. Admin can see more detailed information of the users by clicking on them. He can delete the drivers or can add a new one.
* **Reports View:** As stated before there 2 types of reporting available for admin users. From this view admin user can choose the date for which he wants to have a report. Moreover he can also choose which report he wants to download as a .pdf file.

**Driver View**

* **Calendar View:** The calendar view is the first view that welcomes the user when he logs in also for the driver user. This view is the main view for the user. Here driver user can see the months, weeks and choose his available times and save them. He can also sees how many hours he has fulfilled, how many hours he is missing and how many hours he has just selected.
* **Edit Profile View:** This view allows driver users to update their profile if there are any changes in his mailing address or banking account. Driver directly let Volo these changes by using this view.

**3.3 Persistent Data Management (DB Architecture)**

The persistent data management is very important for the Volo, because the business of the company is the dealing with is running on these data. The database of the company and the database of our web platform are using the same infrastructure that is Heroku Postgres. In our platform we also created with the same technology to create our database because later when we come to thee integration part the only thing is changed the URLs. The rest of the system is not touched. Each Model has its own table in the database.

The first table is the “user” table has 19 columns such as ID, email, password, name, admin, deleted, street, address, plz, city, telephoneNumber, transportType, iban, bic, contractType, base64Image, doneHours, diffPrevHours, locationId.

The second table is the “order” table which has 3 columns such as ID, date, wekDayIndex.

Here we created a non-relational database. Since they are different information that we used for different purposes.

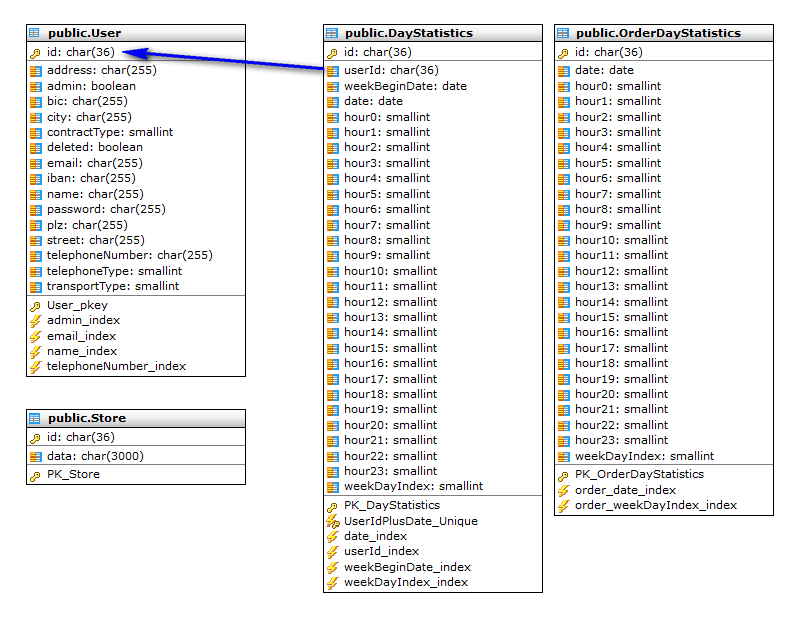


Figure 6

**3.4 Demand Forecasting**

This section gives the detailed information about how the forecasting works. There are 3 steps to make demand-forecasting work. In the forecasting view for admin user there is a panel to enter the parameters for the forecasting. What forecasting does is basically it asks to user which date he wants to pick as a starting point. This algorithm takes the necessary data from the api from the starting point to actual date. To have a good forecasting result we used an m sigma rule that is used for data mining projects. This rule is used to determine the outliners and filter them out. There might be some exceptions in the data. These outliners may affect the result. In order to make the forecasting as correct as possible we first wanted to identify these outliners. If we look into more detail, we first calculate the mean and also the standard deviation from the data we have. Then we look at every data one by one. By calculating the “ (Value – mean) / standard-deviation)” we have the outliers degree (sigma value) for every data. As said before we get this sigma from the user. Then whichever data is bigger than this sigma, meaning it is not in our range, we filter them out. After these processes we have the data that we want to work on it. After this stage we calculate the mean with the remaining date and get the forecasting value for every other day. The steps for user are:

1. Get the data from api, with a beginning date and store them locally.
2. Get sigma value for filtering value from the user. Then calculate the values of (Value-Mean)/standard deviation. Then filter outliers out if their value is bigger than the given sigma value (m-sigma rule).
3. Run the Forecasting algorithm: this calculates the average of the remaining values after the outliers are ruled out.
4. Visualize the forecasting values in the calendar. These values can be changed manually.

**4. Future Work**

Although the project is done and ready to use there are some future works to improve the platform more. There are always space to improve a project and we are aware of that. That is why we wanted to put a section for future work to guide later development. In this section we add what can be done for future work not only from our side but also from api side.

* Firstly we can say it is needed to complete and to test the rest api calls, after all the needed functionality is provided. This work is also commented as to-do tasks inside the code.
* The second work is the server side validation. This is needed because from our application it is not possible to send spy requests. This can be done only by using http request tools. Though security is configured well inside the code, there are some places that you have to check if changing object is valid. As an example profile editing should be done onto current user.
* Finally; completion the missing rest api calls in Volo side is needed. There are some calls that is not provided by api but implemented in the code.

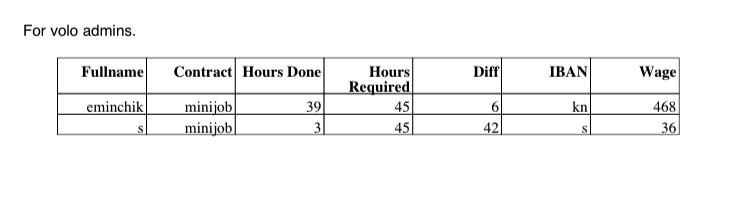
**5. Conclusion**

Volo has its own web system. The project aimed to add a new platform to this existing system. Then Volo changed it to be stand-alone platform. The new platform helps Volo to manage its delivery system on a web platform to make the communication between the company and drivers easy. The old system was to use excel sheet to see the available times of the drivers. As you can imagine this both time consuming and not very easy to use for both sides. The implemented platform fulfills the functional and non-functional requirements of the desired project.

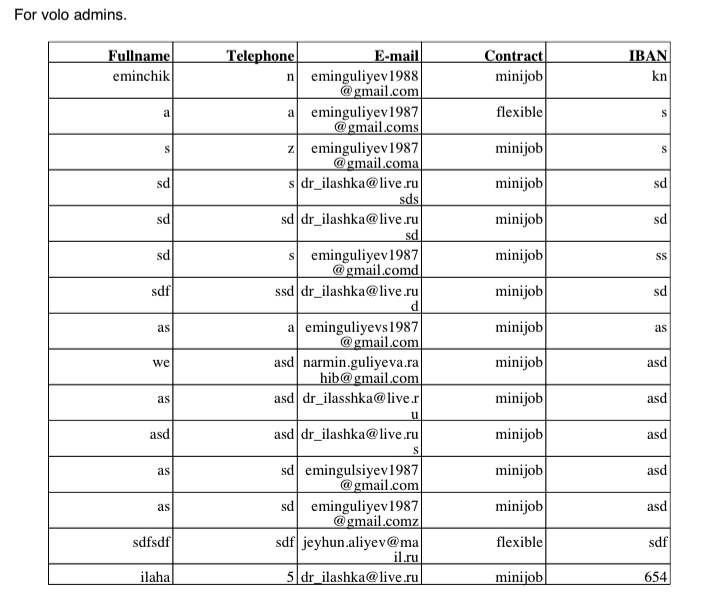
Besides fulfilling functional and non-functional requirements, developing a stand-alone platform with an existing system api was very unique and important experience for the developers. To work on a new technology, Angular.js, was very challenging and instructive. As an Interdisciplinary Project, this project achieved the goal.

**6. Screenshots**

**6.1. Example Report Files**

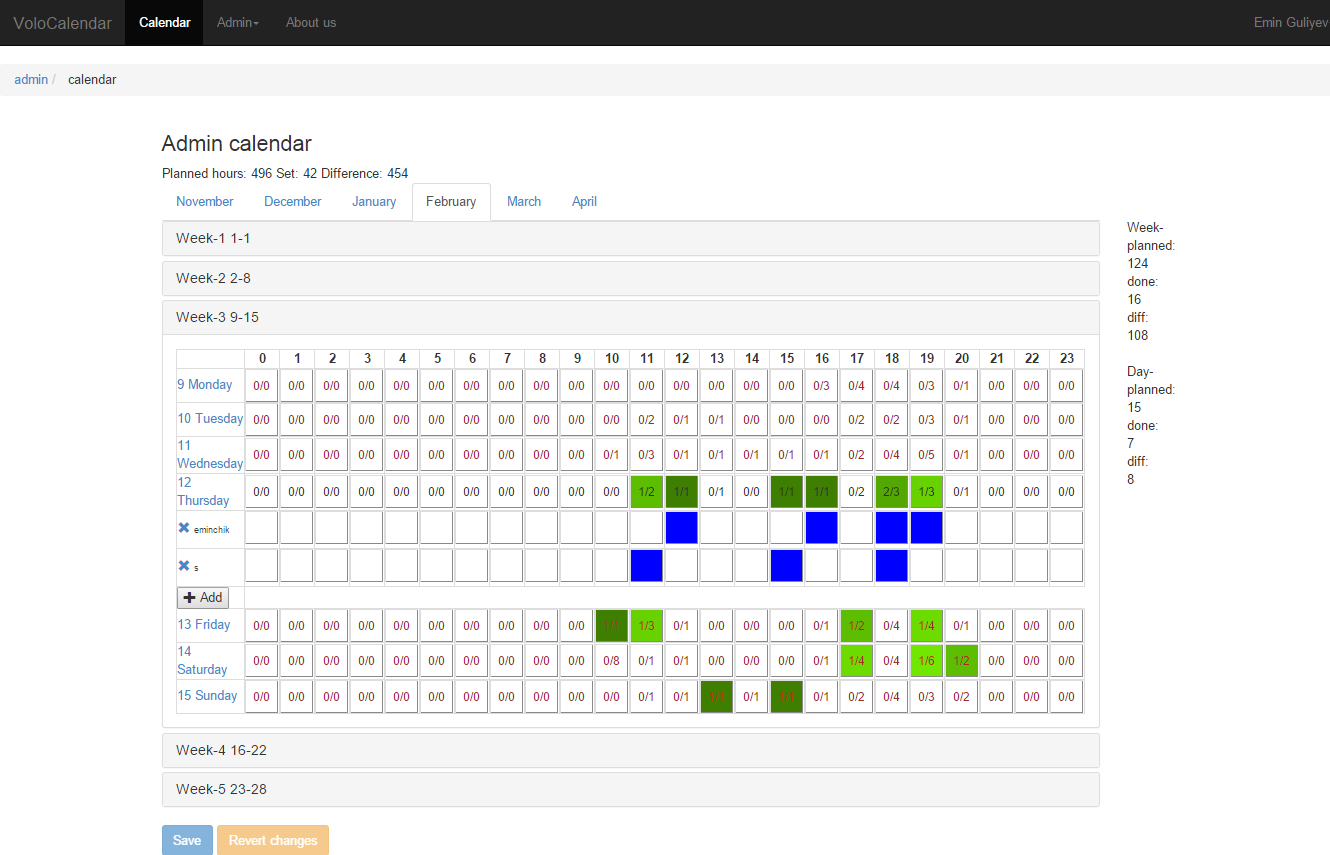
****

Screenshot 1: Report-1

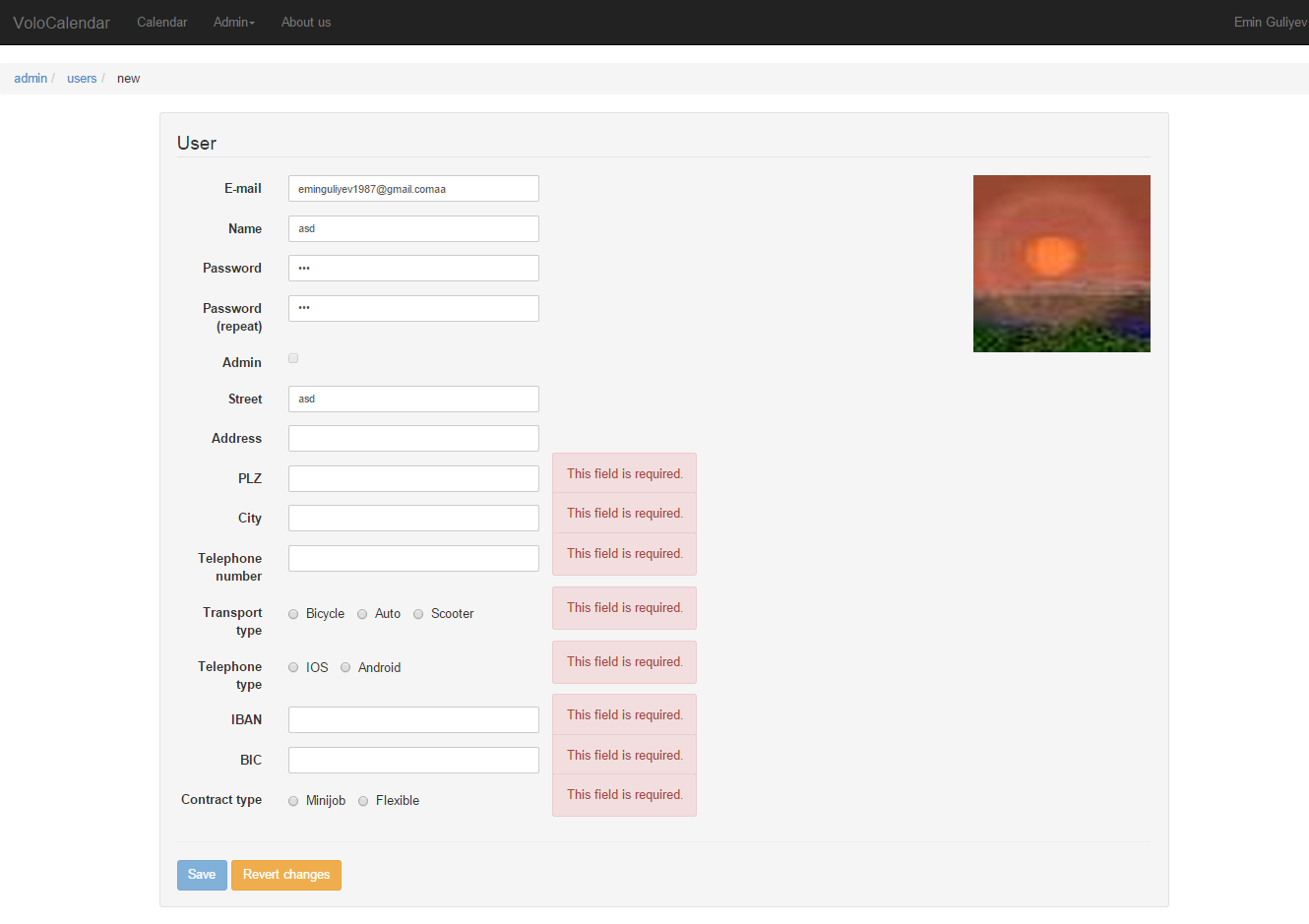


Screenshot 2: Report -2

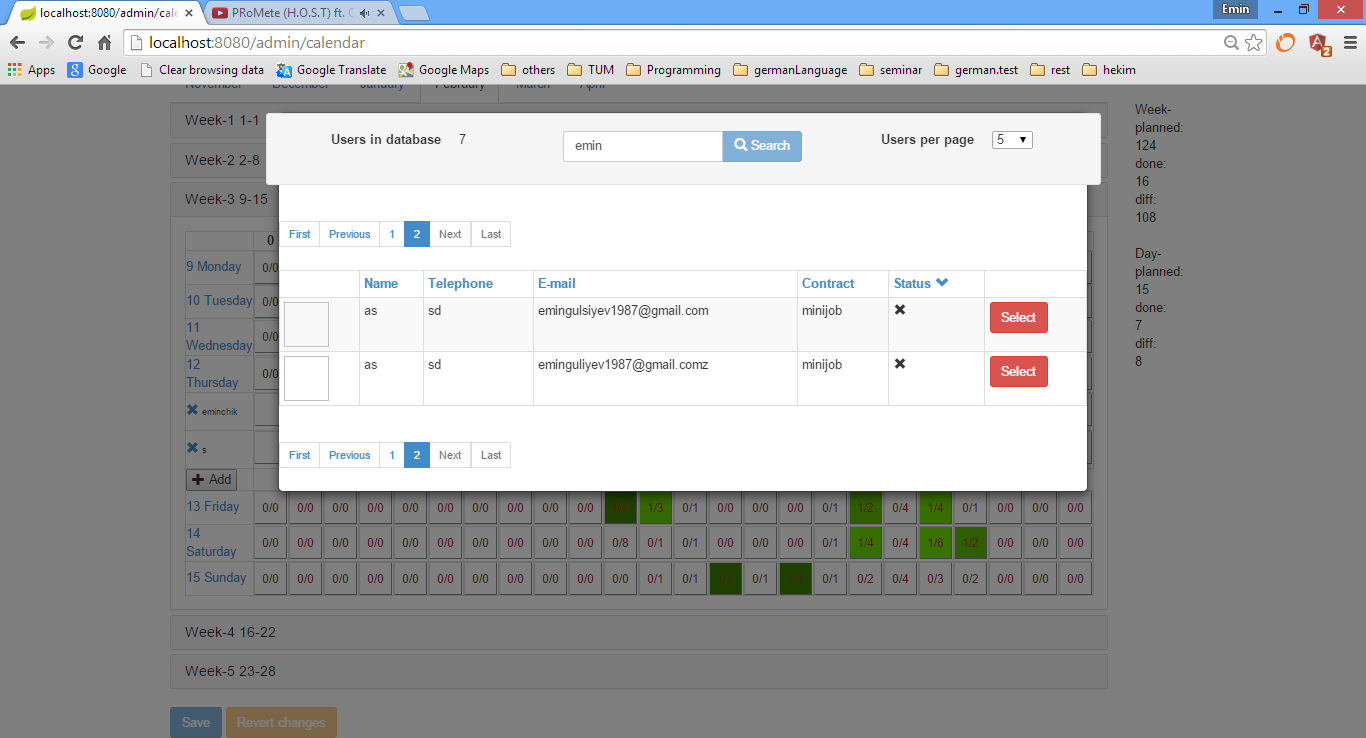
**6.2. Admin User Views**



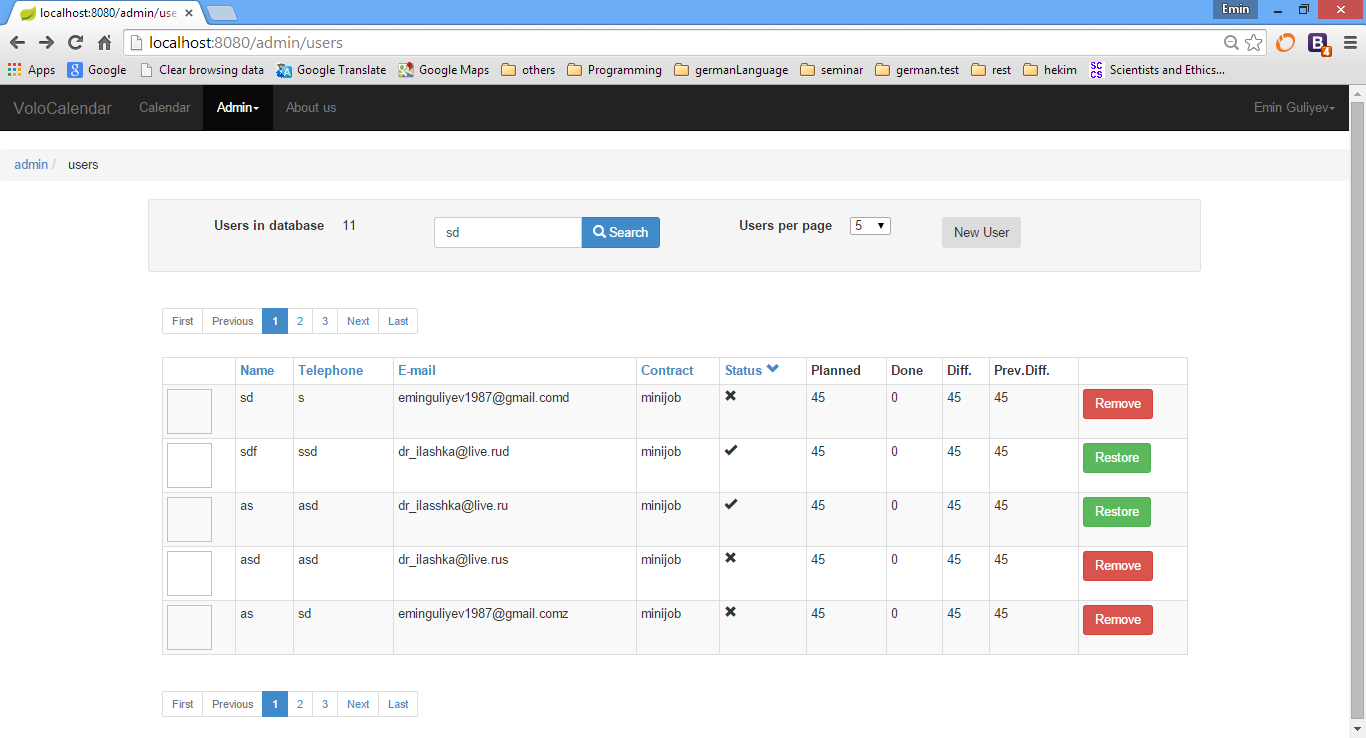
Screenshot 3: Admin Calendar View



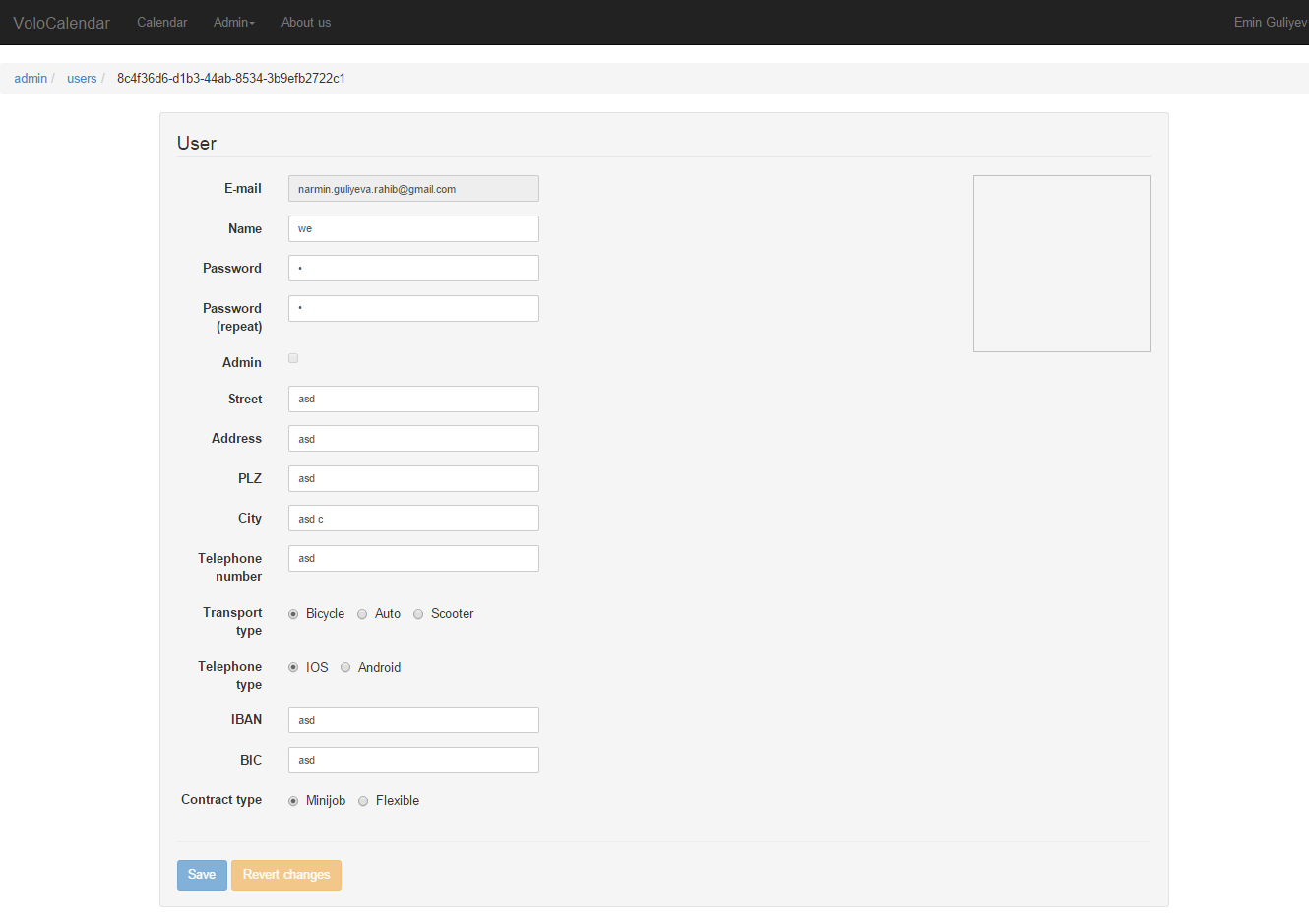
Screenshot 4: Admin Create New Driver View



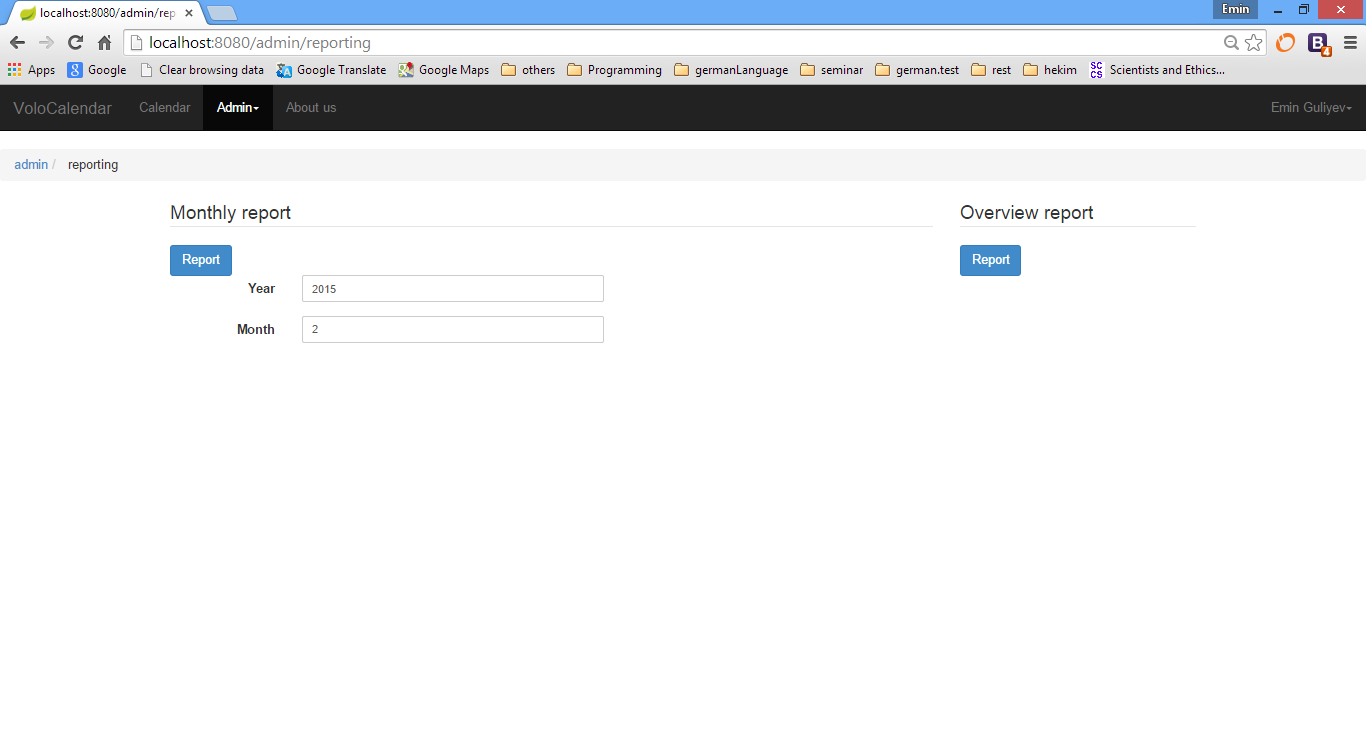
Screenshot 5: Add Driver To The Current Day, In Admin Calendar View



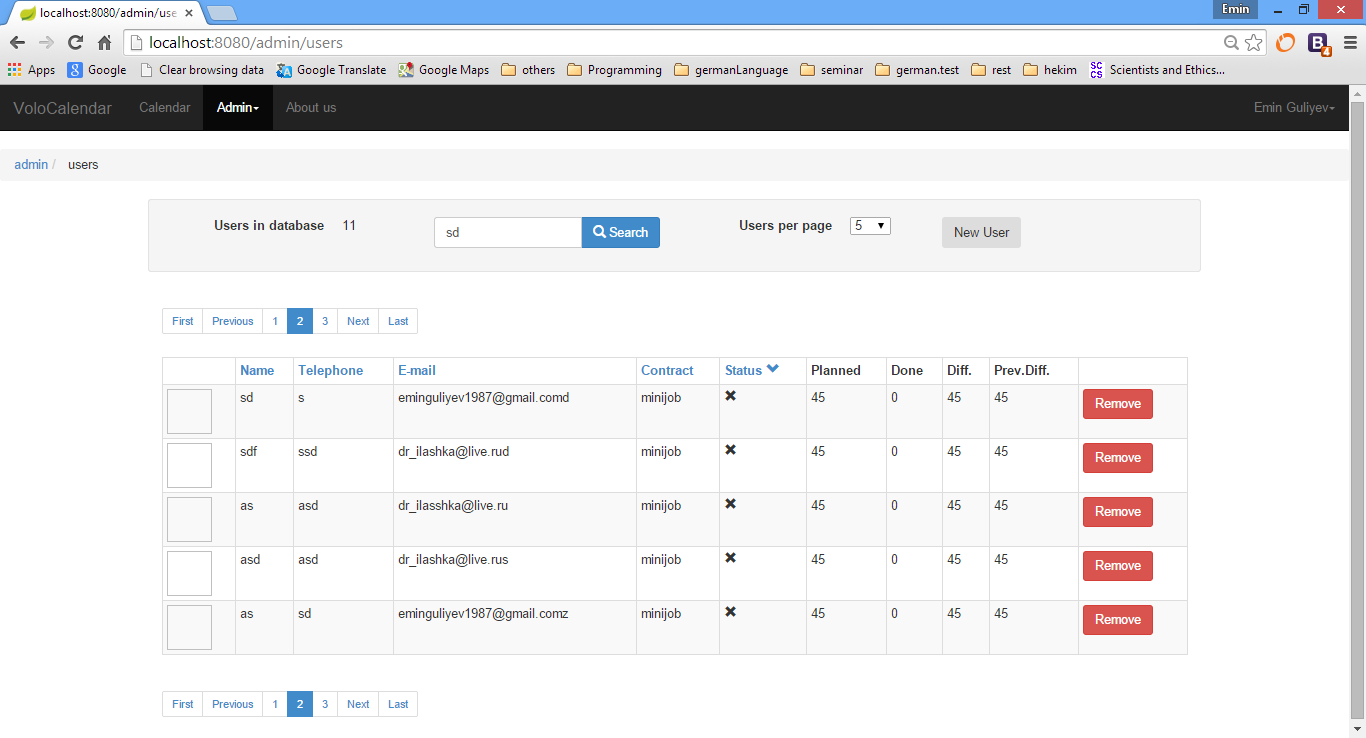
Screenshot 6: Delete And Restore User



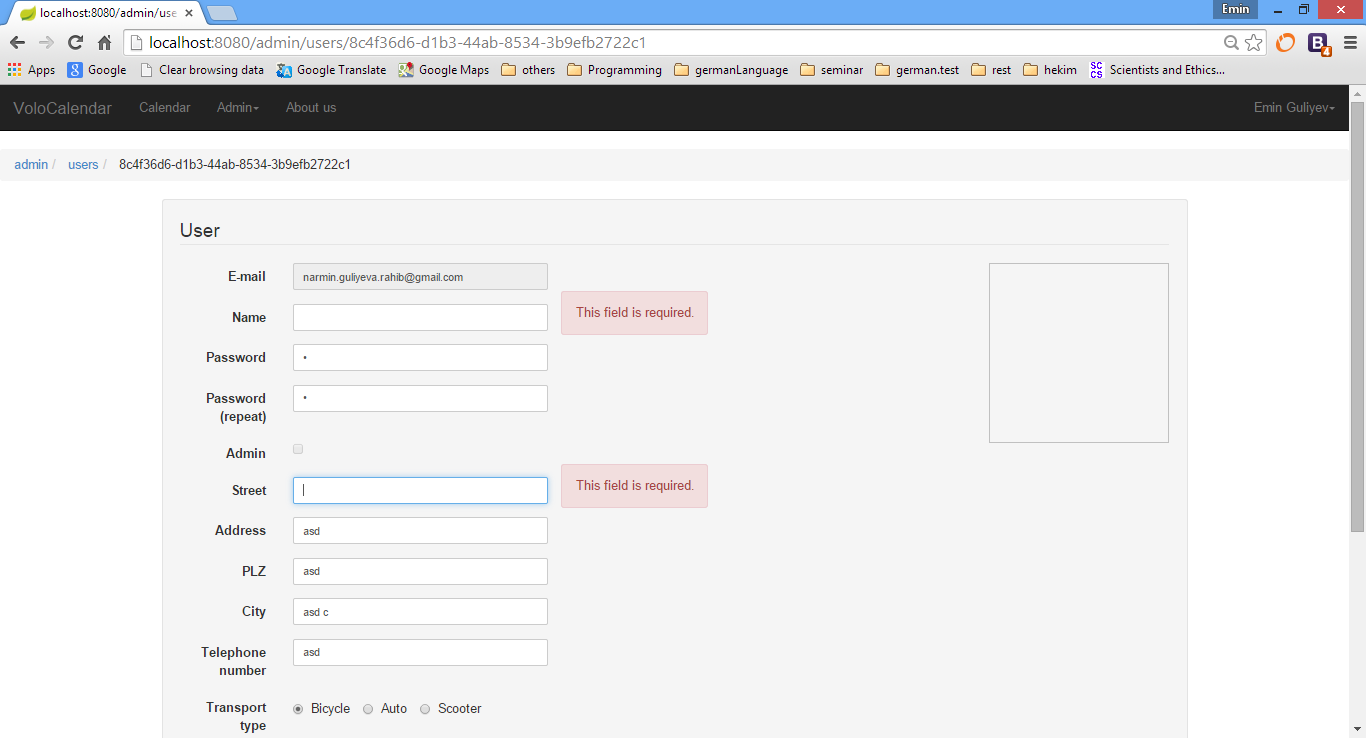
Screenshot 7: Edit User View



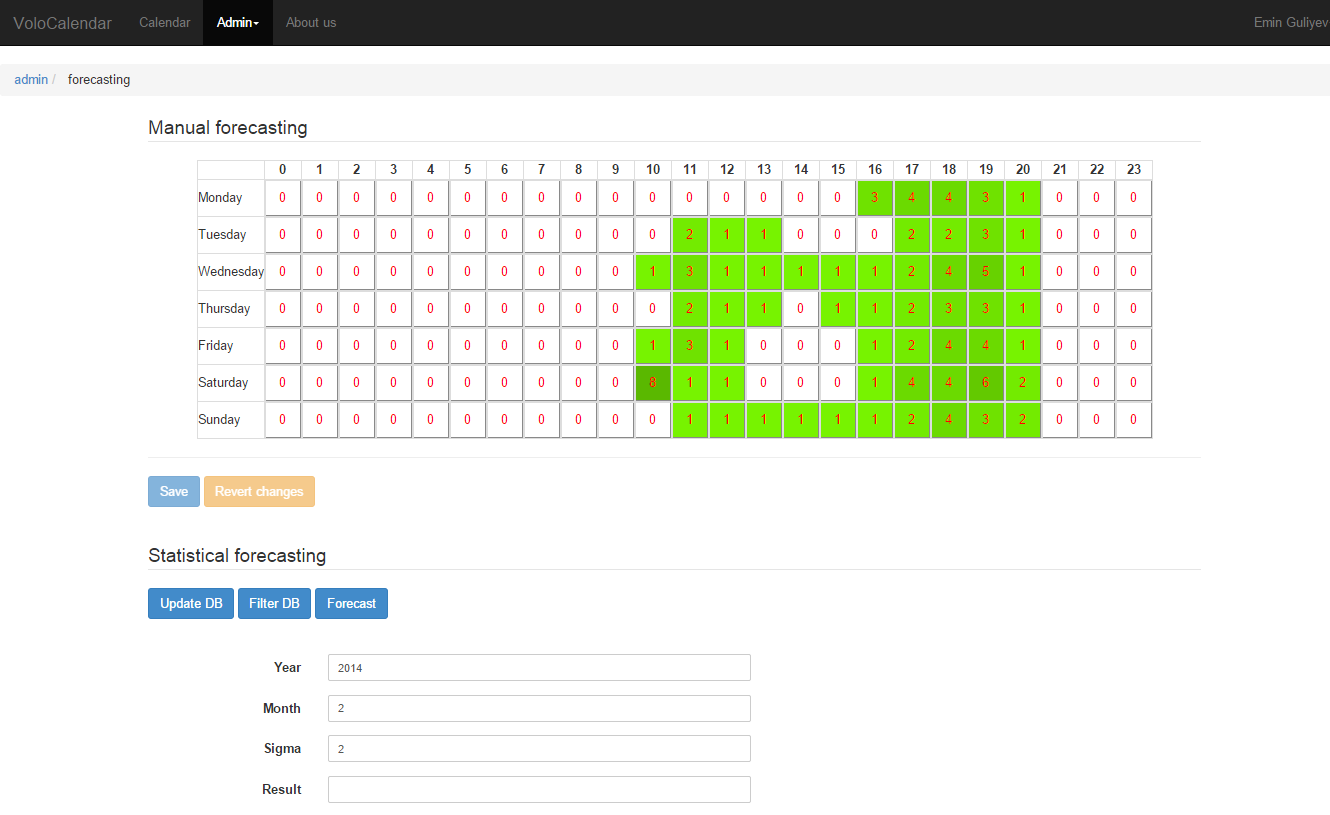
Screenshot 8: Report View



Screenshot 9: User List View

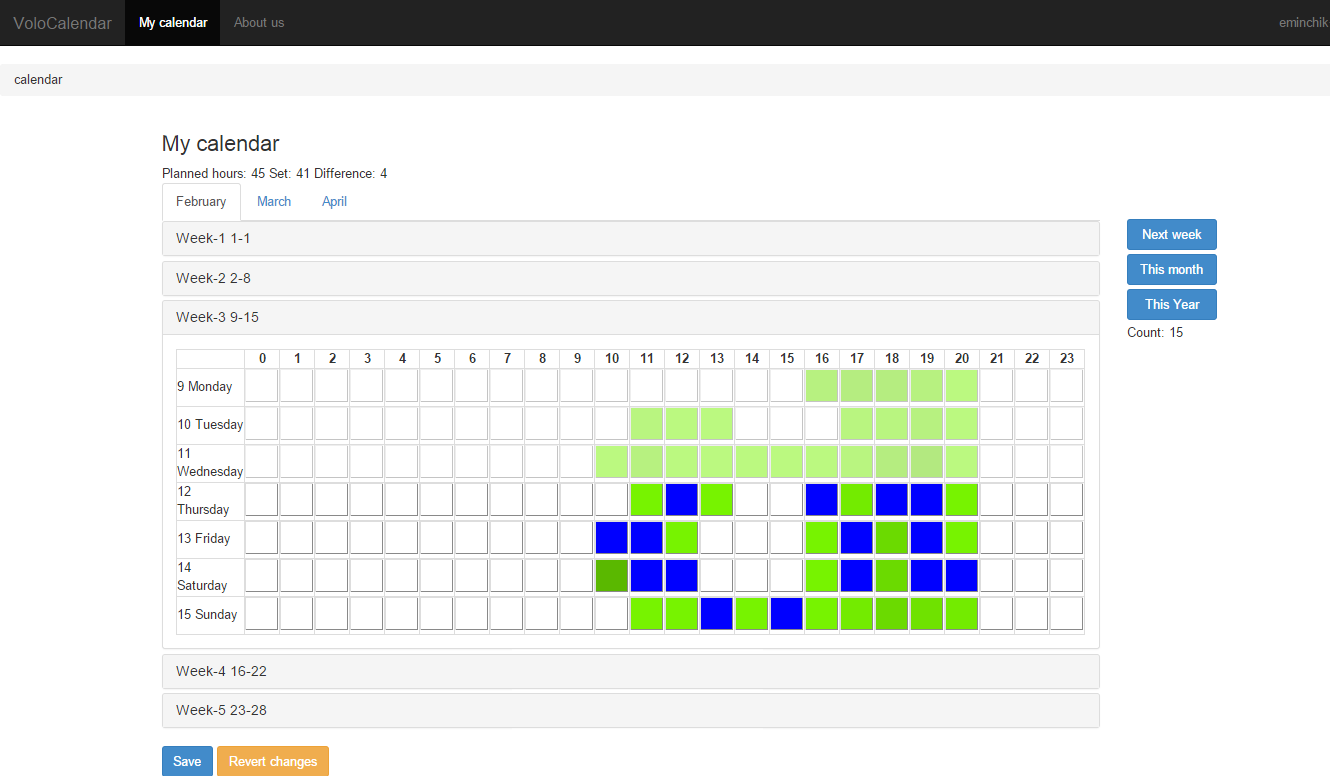


Screenshot 10: Validations

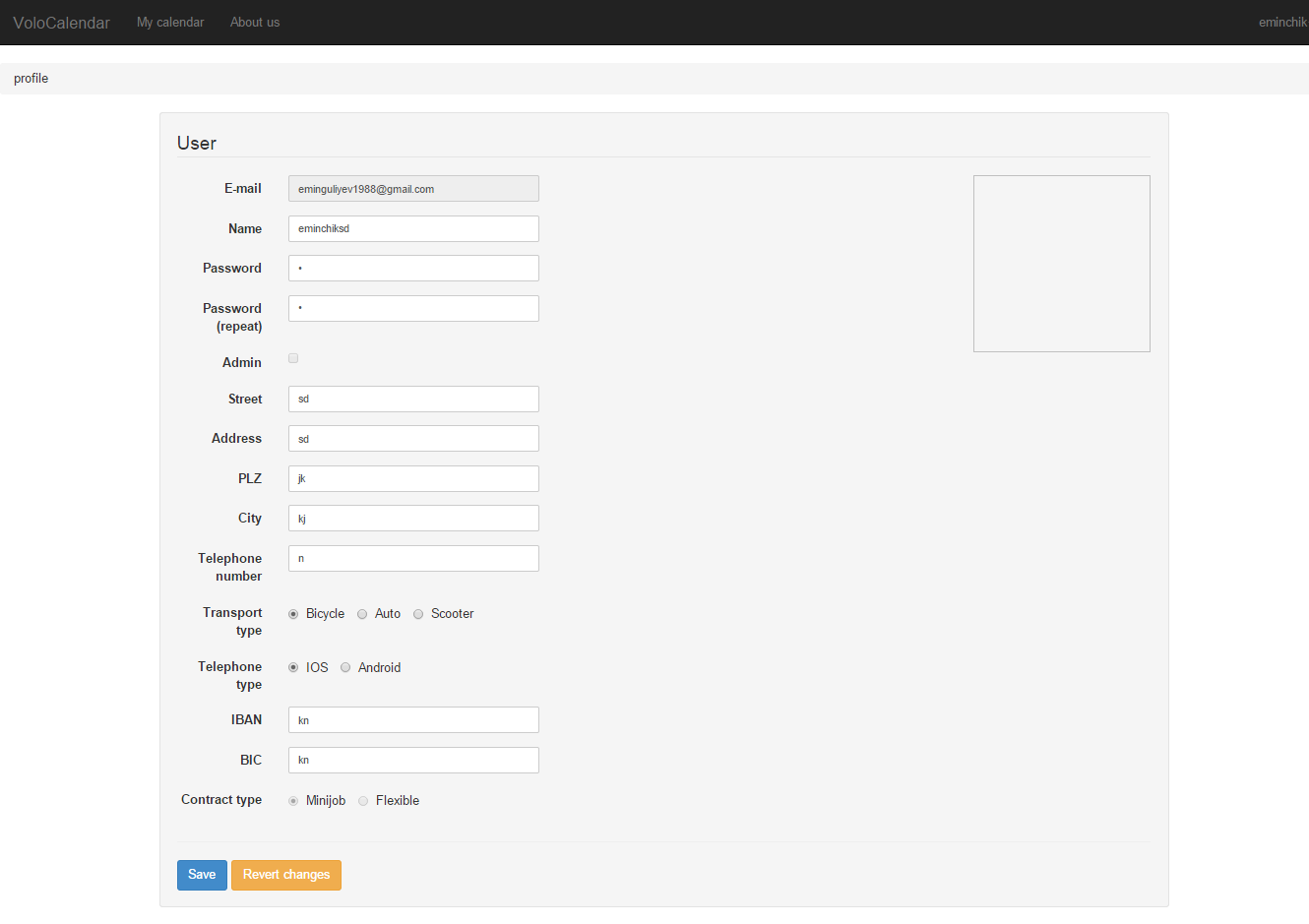


Screenshot 11: Forecasting View

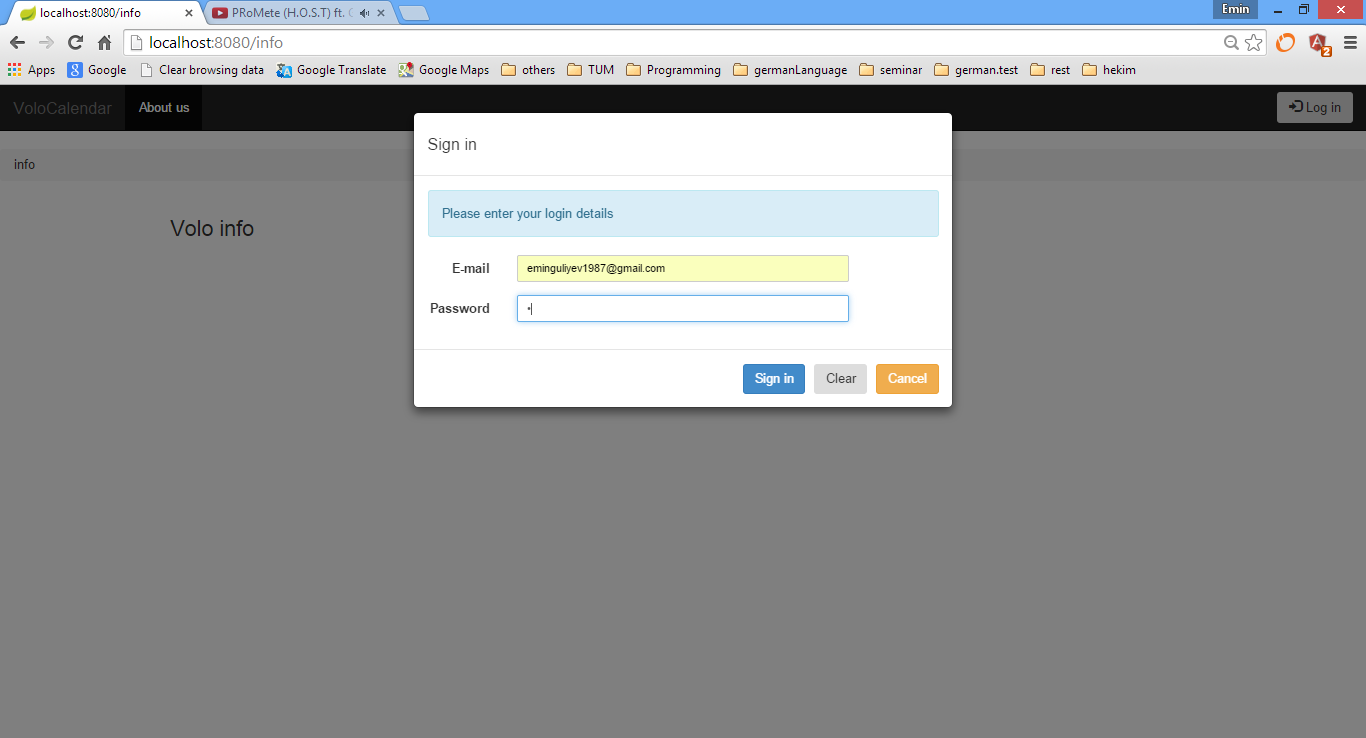
**6.3. Driver User Views**



Screenshot 12: Driver Calendar View



Screenshot 13: Edit Profile View



Screenshot 14: Login View

**7. References**

1. Thomas A. Runkler, Data Analytics, Models and Algorithms for Intelligent Data Analysis
2. Pawel Kozlowski Peter Bacon Darwin, Mastering Web Application Development with AngularJS
3. Brad Green Shyam Seshadri, AngularJS

**8. Documentation**

VoloCalendar is web project for user management, order forecasting, working time scheduling, reporting. The project is developed for Volo company. App based on the example app from the book: Mastering Web Application Development with AngularJS. We changed backend from nodejs to spring mvc, replaced angularjs, angular-ui, bootstrap.css with last variant then developed current app.

**8.1. Build**

First of all before you begin please disable all the proxies that you have in your network. You need to install Node.js and then the development tools. Node.js comes with a package manager called [npm](http://npmjs.org/) for installing NodeJS applications and libraries.

* Install Intellij IDE 13.1.2+ and open project(if dependencies broken, fix them by adding all jar files in VoloCalendar/lib folder)
* [Install node.js](http://nodejs.org/download/)
* Install Grunt-CLI and Karma as global npm modules from command line:

npm install -g grunt-cli karma

* Install local dependencies (from the project root folder) from command line:

cd VoloCalendar\angular

npm install

(This will install the dependencies declared in the client/package.json file)

* Build client application from command line:

cd VoloCalendar\angular

grunt build --force

* Continuous Building from command line

The watch grunt task will monitor the source files and run the default build task every time a file changes:

cd VoloCalendar\angular

grunt watch --force

* Install postgresql and GUI for it(pgAdminIII)
* Run database script to create schema : Volocalendar/documentation/db.sql
* Set database parameters in VoloCalendar\web\WEB-INF\config\ asanRepositoryContext.xml file
* In VoloCalendar\src\volo\voloCalendar\util\Settings.java set path variables properly(you can take testData.json from VoloCalendar\documentation folder)
* Build project
* Build Artifact(Build/Build Artifacts…/VoloCalendar:war/Clean, Build/Build Artifacts…/VoloCalendar:war/Rebuild)
* Configure one Tomcat server inside Intellij IDE and run project

**8.2. TODO for future**

All todo tasks in java code, nothing in javascript/html/css. After completion Volo service you can switch to rest api call scenario:

* Server side validation(just validate if sent object assumed to be editable and editable like that)
* TODO 1.1: uncomment after adding mail\_location\_id to Driver class by Stefan.
* TODO 1.2: uncomment after adding mail\_location\_id to Driver class by Stefan
* TODO 2: DE is ok?
* TODO 3: set next 2 values accordingly to your file system, do not forget to use proper file separator
* TODO 4: delete these two lines and usages
* TODO 5: comment next uncomment after next
* TODO 6: Order filtering, get only orders up to lowerBound
* TODO 7:comment next line
* TODO 8:uncomment next line
* TODO 9: implement it, we need to get user role on authentication, for example by "isAdmin" boolean header as shown
* TODO 10.1: filtering, filtering should be done by email, name and phone fields, fix after and according server side is done
* TODO 10.2: filtering, filtering should be done by email
* TODO 10.3: filtering, filtering should be done by status
* TODO 11: implement it, mapping between local field name and server side sort parameter desired value
* TODO 12: implement, somehow you should provide api to change password of driver, or you can add password directly to Driver object in put request(update of whole driver)

**8.3. Contents**

# Documentation folder:

db.sql – script to create database

Backup.postman\_dump – postman dump for testing Volo side rest api

testData.json – lines of date strings that is the dispatched\_at fields of real order data of Volo until 2015 February

documentation.docx – documentation file for developers

Volo\_IDP.docx – IDP report(user manual and overview for developers)

# Lib folder:

All jar dependencies for project

# Web folder:

## Resources folder:

angular.js – standard angular library last version

bootstrap.js – standard angular-ui library last version

bootstrap.css – standard bootstrap style library, but custom changes added to the end of file

jquery.js – standard jquery library last version

mongolab.js – custom library for enlarging JavaScript objects that converted from json, rest methods added to make coding easy as : obj.save/obj.update/obj.remove

angular-app.js - grunt compiled file, that holds all javascript files in angular folder and all html files as javascript strings

## WEB-INF folder:

### Config folder:

log4j.properties – log4j properties file, but currently custom logging does not exist

applicationContext.xml – application context, just container that links other specific configuration file

voloRepositoryContext.xml – spring-data configuration

main-servlet.xml – common configuration for spring controllers

security-app-context.xml – spring security configuration

### Views folder:

main.jsp – main page that is returned every time when user write any bookmark able url in address bar of browser and press enter

# angular folder: holds all javascript/html custom files

gruntFile.js – grunt build file as “maven” for javascript, defines javascript source file strunture and all commands like “build, watch, etc.”

## vendor folder:

holds all standard libraries as angular, angular-ui, jquery and custom library mongolab-resource.js all of them will be copied to web/resources folder on build or watch grunt command execution

## src folder:

### common folder:

#### resources folder:

users.js – defines user javascript class as mongolab-resource(for definition look on the top) class + one additional getFullName method

#### directives folder:

##### **crud folder:**

edit.js – defines crud-edit directive(CRUD operations for any mangolab-resource object, in our case user object), for usage look at angular/src/app/admin/users/ users-create.tpl.html or users-edit.tpl.html

crudButtons.js – defines crud-buttons directive( just save and revert changes buttons html content) ), for usage look at angular/src/app/admin/users/ users-create.tpl.html or users-edit.tpl.html

crud.js – container for above two

#### security folder:

for explanations please read “Mastering Web Application Development with AngularJS” book , chapter 7: Securing Your Application. All code taken from source code of book and changed little bit according to app logic

#### services folder:

breadcrumbs.js – breadcrumbs implementation

crud.js – some additional functions, for usage look at angular\src\app\admin\users\admin-users-list.js

crudRouteProvider.js – for creating CRUD routes for mongolab-resource class, for usage loof at angular\src\app\admin\users\admin-users.js

exceptionHandler.js – global exception handler

httpRequestTracker.js – defines if there is ongoing request/reply at current moment

notifications.js – notifications to user(for example , Driver deleted/added/updated successfully/unsuccessfully) for current or next page

localizedMessages.js – for localization aims, after translation parametres inserted into proper positions((hello.string, [‘Emin’]) 🡪 (Servus, {name}!, [‘Emin’]) 🡪 Servus, Emin!), key-value pairs saved in angular\src\app\app.js

i18nNotifications.js – enlarges notification.js using localizedMessages.js to create localized notifications, used everywhere in project

utilMethods.js – utility methods for file🡪base64String conversion, week day/month names exchange, object get/set(just save object and get it from memory when you need), colour shading operations

### app folder:

app.js – container module that links all other modules, defines header of page activities and notification activities

header.tpl.html – referenced from main.jsp, controller inside app.js

notifications.tpl.html – referenced from main.jsp, activities inside app.js

#### home folder:

home.js – if current user admin home page is admin/calendar , if driver home page is /calendar , if not authenticated user home page is /info

#### info folder:

just static information page, not required authentication

#### calendar: only for drivers

driver calendar page view/controller/directive

#### admin folder: only for admins

admin.js – container for all admin pages

##### calendar folder:

admin calendar page view/controller/directive

##### **forecasting folder:**

forecasting page view/controller/directive

##### **reporting folder:**

reporting page view/controller

##### **users folder:** taken from “Mastering Web Application Development with AngularJS” book source code and changed according to app logic

user management(user CRUD) pages(all users sorting/filtering/paging, edit user, add user ) view/controller/directive

# src folder:

## META-INF folder:

persistence.xml – entity classes list and some jpa configuration

## volo folder:

### voloCalendar folder:

#### controller folder: all spring controllers

just taking http request, extracting controller parameters and forwarding request to services(\*Logic.java classes)

AdminCalendarController.java – admin calendar activities

DriverCalendarController.java – driver calendar activities

ForecastingController.java – manual and statistical forecasting activities

LoginController.java – get current user, login, logout, update profile(only for drivers) activities

MainController.java – return main.jsp for all bookmark able url entries typed on browser address bar or selected from saved bookmarks, forwards urls starting with “static” to resources folder to get images, fonts and other html resources

ReportsController.java – active drivers by month(monthly) and all users(overview) reports

UserManagementController.java – user sorting, filtering by name/email/phone number, paging, marking as deleted, restoring, adding(with unique email address), editing, viewing activities

#### Dao folder:

All spring data interfaces having methods that either is implemented dynamically by spring or acting as proxy to named queries of jpa level

#### Entity folder:

Persistent objects and enums belonging to them

#### externalModel folder:

All Volo server side objects that is receiving/sending in rest api by Volo backend, used only for converting to/from local objects

#### Security folder:

CustomEntryPoint.java – do not redirect 401 Unauthenticated exceptions to login page as default spring mvc platform does, but just send this exception to client side (angular gear in browser)

#### Service folder: logical operations implemented here

CalendarLogic.java – common logic for driver and admin calendar activities, parent class for DriverCalendarLogic.java and AdminCalendarLogic.java, used in any case if current user is driver or admin

DriverCalendarLogic.java – if current user is driver

AdminCalendarLogic.java – if current user is admin

ForecastingLogic.java – forecasting logic

LoginLogic.java – authenticates current user, used by LoginController.java

ReportLogic.java – reporting logic

UserManagement.java – common interface of two user management services(UserManagementLogic.java and UserManagementLocalLogic.java)

UserManagementLogic.java – implements rest api calls to Volo service

UserManagementLocalLogic.java – default UserManagement implementation, work with local postgresql table User

#### Util folder: helper classes

CalendarUtilMethods.java – utility methods for calendar activities, not connects to database

UserGrantedAuthority.java – holder class for role information of user

UtilMethods.java – json to/from object conversions, authenticating rest api calls, etc.

Settings.java – static values for future pages that will change this values and save them in database not in code

#### viewModel folder:

All models that are not persisted to database

##### **Admin folder:**

View model classes used in only admin calendar activities

##### **Driver folder:**

View model classes used in only driver calendar activities

##### **Common folder:**

View model classes used in both admin and driver calendar activities

##### **Forecasting folder:**

Forecasting view models

##### **User folder:**

UserTable.java – holds user action parameters for html table that holds user list in admin/user page, see comments in file for explanations of fields

UserTableItems.java – holds drivers for only one proper page and count of all drivers