

TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Department of Informatics

Chair of Information Systems

**Web application front end architecture  
and development using AngularJS  
framework**

Bachelor's thesis

Student: Stanislav Nazmutdinov

Student code: 121078IAPB

Supervisor: Raul Liivrand

Tallinn  
2015

## **Declaration of authorship**

Herewith I declare that this thesis is based on my own work. All ideas, major views and data from different sources by other authors are used only with a reference to the source. The thesis has not been submitted for any degree or examination in any other university.

-----  
*(date)*

-----  
*(signature)*

## **Abstract**

The aim of this thesis is to investigate Single Page Web Application front end development and architecture that is developed with AngularJS JavaScript framework. Both Single Page Application concept and AngularJS framework are relatively new phenomena in web application development, therefore their features need to be examined.

As the result of the work there will be developed a property rental web application. During the project development will be reviewed various AngularJS-based application features. Project development will not be limited only to AngularJS framework. AngularJS is pretty versatile and flexible and therefore provides many ways to use and integrate other JavaScript libraries as well as utilize various JavaScript development tools. Despite the fact that the main goal of this thesis is to examine application front end development with AngularJS there will be developed both front end and server-side of the application and briefly described application back end.

The thesis is in English language and contains 56 pages of text, 5 chapters, 19 figures, 6 tables

## **Annotatsioon**

Selle bakalaureusetöö põhiseks eesmärgiks on uurida Single Page Application tüüpi veebirakenduse front end'i arendamise ja arhitektuuri, mis on realiseeritud AngularJS'i raamistiku abil. Kuna mõlemad Single Page Application tüüpi arhitektuur ja AngularJS raamistik on päris uudsed fenomenid veebirakenduse arendamisel, seetõttu nende võimalused on oluline uurida.

Lõputöö tulemusena on arendatud kinnisvara rentimise veebirakenduse. Projekti arendamise jooksul uuritakse erinevad Angularil põhineva rakenduse omadused. Arendamine ei hakka olema piiratud ainult AngularJS raamistikuga. AngularJS on päris paindlik, seetõttu ta pakub päris palju erinevaid viise, kuidas kasutada ja integreerida rakenduses ka teisi JavaScript'i teeki ja rakendada igasuguseid JavaScript'i töövahendeid. Vaatamata sellele, et lõputöö põhiseks eesmärgiks on rakenduse kliendi poolse osa arendamine AngularJS raamistikuga, serveri osa ka hakkab olema realiseeritud ja lühidalt kirjeldatud.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 56 leheküljel, 5 peatükki, 19 joonist, 6 tabelit.

## Abbreviations

<b>HTML</b>	<i>HyperText Markup Language</i>
<b>CSS</b>	<i>Cascading Style Sheets</i>
<b>JS</b>	<i>JavaScript</i>
<b>JSON</b>	<i>JavaScript Object Notation</i>
<b>SPA</b>	<i>Single page application</i>
<b>DOM</b>	<i>Document Object Model</i>
<b>AJAX</b>	<i>Asynchronous Javascript and XML</i>
<b>REST</b>	<i>Representational State Transfer</i>

## Table of figures

Figure 1: Web application architecture evolution .....	14
Figure 2: Google Search interest over time towards various JS SPA frameworks .....	16
Figure 3: Property managing use case diagram.....	18
Figure 4: Booking managing use case diagramm.....	20
Figure 5: Account managing use case diagram .....	21
Figure 6: Entity-relationship diagramm .....	23
Figure 7: Look-up entities diagramm .....	23
Figure 8: Overall application architecture .....	25
Figure 9: Default grunt watch task activity diagram .....	30
Figure 10: Grunt test task activity diagram .....	30
Figure 11: Project static files structure .....	30
Figure 12: Initial Angular application source code structure .....	32
Figure 13: Improved Angular application source code structure .....	33
Figure 14: Angular application architecture.....	35
Figure 15: Angular application scope object workflow .....	38
Figure 16: My-datepicker directive produced result .....	40
Figure 17: Successful authentication sequence diagram .....	42
Figure 18: Security checks on state change sequence diagram.....	43
Figure 19: AngularJS http response interceptor sequence diagram .....	44

## **Table of tables**

Table 1: Description of property managing use cases .....	18
Table 2: Description of booking managing use cases .....	20
Table 3: Description of account managing use cases .....	21
Table 4: Defenition of entity types .....	23
Table 5: Application REST endpoints.....	25
Table 6: Application login REST endpoint .....	41

## Table of contents

1. Introduction .....	10
1.1 Thesis background and problem.....	10
1.2 Thesis objectives.....	11
1.3 Methodology.....	11
1.4 Thesis structure.....	11
2. Theoretical background .....	13
2.1 Single page application.....	13
2.2 Single Page Application architecture .....	13
2.3 AngularJS .....	15
3. Project development .....	17
3.1 Project analysis .....	17
3.1.1 Use case models .....	17
3.1.2 Entity-relationship diagram .....	22
3.2 Server-side implementation.....	23
3.2.1 Application REST endpoints.....	25
3.3 Setting up development environment.....	27
3.4 Project sources structure organization.....	31
3.5 AngularJS application architecture.....	35
3.5.1 Dependency injection .....	36
3.5.2 Scope object.....	37
3.6 Third-party JavaScript libraries integration.....	38
3.7 AngularJS application security .....	41
4. AngularJS application testing.....	46
4.1 Unit testing tools.....	46
4.2 Testing Angular filters.....	46
4.3 Testing Angular directives.....	48
4.4 Testing Angular controllers .....	49
5. Conclusion .....	51
Kokkuvõtte .....	53



References .....	55
Appendix 1. Bower dependencies .....	57
Appendix 2. Node.js dependencies.....	59

# **1. Introduction**

Nowadays web front end development advances extremely swiftly due to increasing number of different frameworks and tools that allow to develop front end more efficiently. This fact allows shifting all the presentation logic to the client-side of the application. As the result, there are constantly increasing number of new type of web applications that no longer require page refresh, provide rich user interface and at the same time are extremely testable, maintainable and scalable for future development – Single Page Application.

Undoubtedly, there are a lot of frameworks that provide their own ways of implementing Single Page Application such as AngularJS, Ember.js, Backbone.js, throughout this thesis there will be used AngularJS in conjunction with other single-purpose JavaScript libraries that all combined enable to make rich and interactive user interface.

When it comes to server-side of the application, the main requirement towards the server is that it would be able to serve and accept data in JSON format. Such capabilities do possess most of the modern server-side languages and frameworks, so the author will use technologies, which he is most experienced in. Therefore, the server-side of the application will be implemented with Java based technologies.

## **1.1 Thesis background and problem**

Single Page Applications are relative new type of web applications that do take web application front end development to a whole new level providing flexibility and at the same time well organized structure and architecture that guarantees needed code maintainability, scalability and testability.

The thesis might be particularly interesting for everybody who would like to explore the nature of Single Page Application development with AngularJS framework and other numerous JavaScript libraries and development tools.

## **1.2 Thesis objectives**

During this thesis, there will be shortly explored how web application architecture evolved and formed.

Besides, there will be examined project structure and architecture of AngularJS-based Single Page Application and considered their advantages and drawbacks.

Thirdly, there will be reviewed JavaScript development tools and discussed what they can provide for AngularJS front end developer.

What is more, there will be reviewed how to integrate third-party JavaScript libraries into AngularJS application.

Furthermore, there will be inspection of way of implementing authentication in AngularJS application.

Lastly, there will be a short overview of AngularJS-based application's suitability for unit testing.

## **1.3 Methodology**

In order to achieve above listed aims there will be developed sample project of property rental web application. Through the development of this system, there will be examined features of Single Page Application and AngularJS framework in particular.

## **1.4 Thesis structure**

Theoretical background chapter of the thesis presents an overview what Single Page Application is and describes what opportunities AngularJS framework provides in order to implement such web application functioning approach.

Project development chapter describes numerous aspects of AngularJS-based application development as well as presents short project analysis.

AngularJS application testing chapter is devoted to AngularJS application testing. There is reviewed how to test Angular application components with Jasmine testing framework.

Conclusion chapter summarizes the aims of the work and describes achieved results.

## **2. Theoretical background**

### **2.1 Single page application**

Web application development has always been a major part of software development. However, there had been a lot of restrictions in interaction with complex web applications which made them not as responsive as were for example desktop applications. As an alternative to traditional web applications were developed so called rich internet applications that are quite similar to desktop application software that run within browser via installed third-party (oftentimes proprietary) plugins.

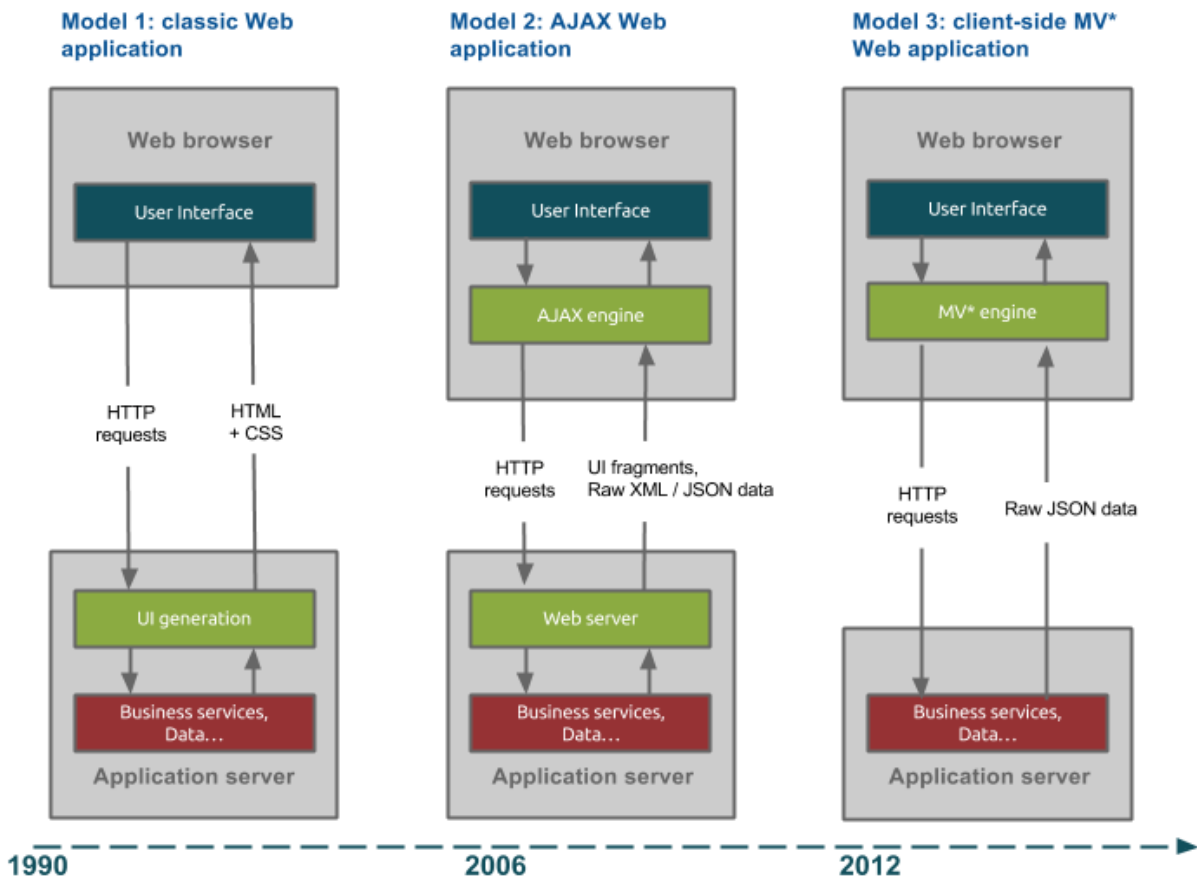
At the same time the new versions of traditional technologies used for web development such as HTML, CSS and JavaScript were evolving and progressing which made it possible to create a new generation of lightweight and flexible web applications - Single Page Applications. This new type of web application are comparable in terms of interactivity with desktop applications [1] and at the same do not require any additional plugins to function.

The only requirement that should be fulfilled in order to make Single Page Application work on the user computer is ideally the latest web browser version. This may be considered as a drawback of such applications since JavaScript support is still an issue all across the globe. There are still lots of users that do not always update their web browsers to the latest version or even deliberately turn JavaScript off for security or privacy reasons.

### **2.2 Single Page Application architecture**

First of all, we will examine different web application architectures in order to fully understand difference between various approaches and eventually comprehend the nature of Single Page Applications.

The following scheme illustrates pretty well the evolution of web application architectures [2]:



**Figure 1: Web application architecture evolution**

Model 1: classic Web application

The first diagram shows the architecture of old-fashioned web applications, which mostly do not use any fancy JavaScript technologies. Server generates a whole HTML page on each incoming from web browser request. It may also send to client some CSS and JavaScript files that a little bit enhance user experience.

Model 2: AJAX Web application

The second type of web applications, which was introduced in mid-2000s, is an improved version of the first type of web applications. Unlike classic web application, it uses not only traditional HTTP requests to retrieve whole HTML pages from the server, but also AJAX patterns so as to get code fragments or raw XML, JSON data from server-side. As the result web applications become more responsive, since some parts of web page can be reloaded dynamically using JavaScript without page refresh.

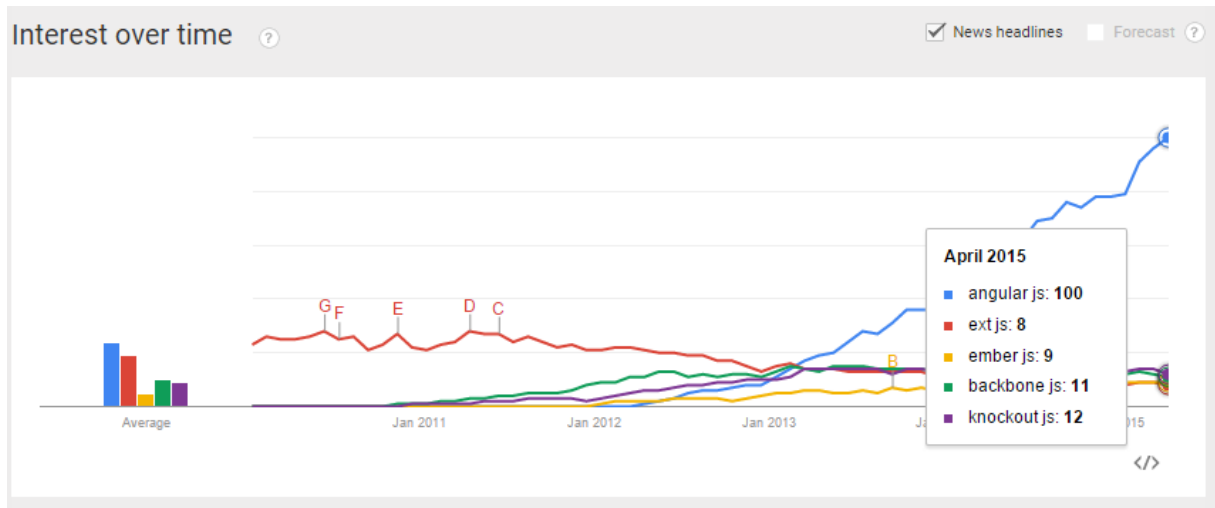
Model 3: Single Page Application

The third model shows brand new web application architecture that uses advanced JavaScript framework on the front end. In contrast to previous architectures server no longer generates views in any way, it only sends raw JSON/XML data and HTML templates (partials) to the client on demand. Client-side of the application is fully responsible for generating user interface using received from the server data. In other words, whole user interface logic is moved from server to client-side of the application. Moreover, on Single Page Applications everything operates on a single page – there is no page refreshes, as, for instance, in traditional web applications where page reloads on each request. All above listed points provide more fluid and rich user experience.

## **2.3 AngularJS**

AngularJS is a part of a new generation of open-source web application client-side frameworks that was initially created in 2009 as part of business idea of Miško Hevery and Adam Abrons [3]. Unfortunately, as a business idea, it was not successful enough. As the result, AngularJS was released as an open-source framework. Google and community maintain nowadays AngularJS.

Although it is quite a new framework, which heavily relies on the latest versions of web technologies, it has already gained some popularity among web developers. Although AngularJS framework market share is not that high in comparison to other JavaScript libraries (0.3% as of May 2015) [4], but it shows quite positive trends and has active and constantly growing community. This may be illustrated by Google Trends that analyze people interest towards different JavaScript frameworks, that allow building Single Page Applications [5]:



**Figure 2: Google Search interest over time towards various JS SPA frameworks**

Moreover, a lot of companies such as Google, Vevo, MSNBC, Sony have already taken interest in AngularJS [6].

What is more, AngularJS framework is actively maintained – the updates are quite frequent. AngularJS not only makes it possible to build well-structured, modern and responsive applications, but also provides means to extend them with various JavaScript libraries.



## **3. Project development**

In order to examine and illustrate nature of Single page applications and AngularJS framework there will be developed sample project. The project is a property rental system.

The current version of project source code is accessible on GitHub <https://github.com/PrintScr/Property-rental-system>

The project address is <http://imbi.ld.ttu.ee:443/propertyRental>

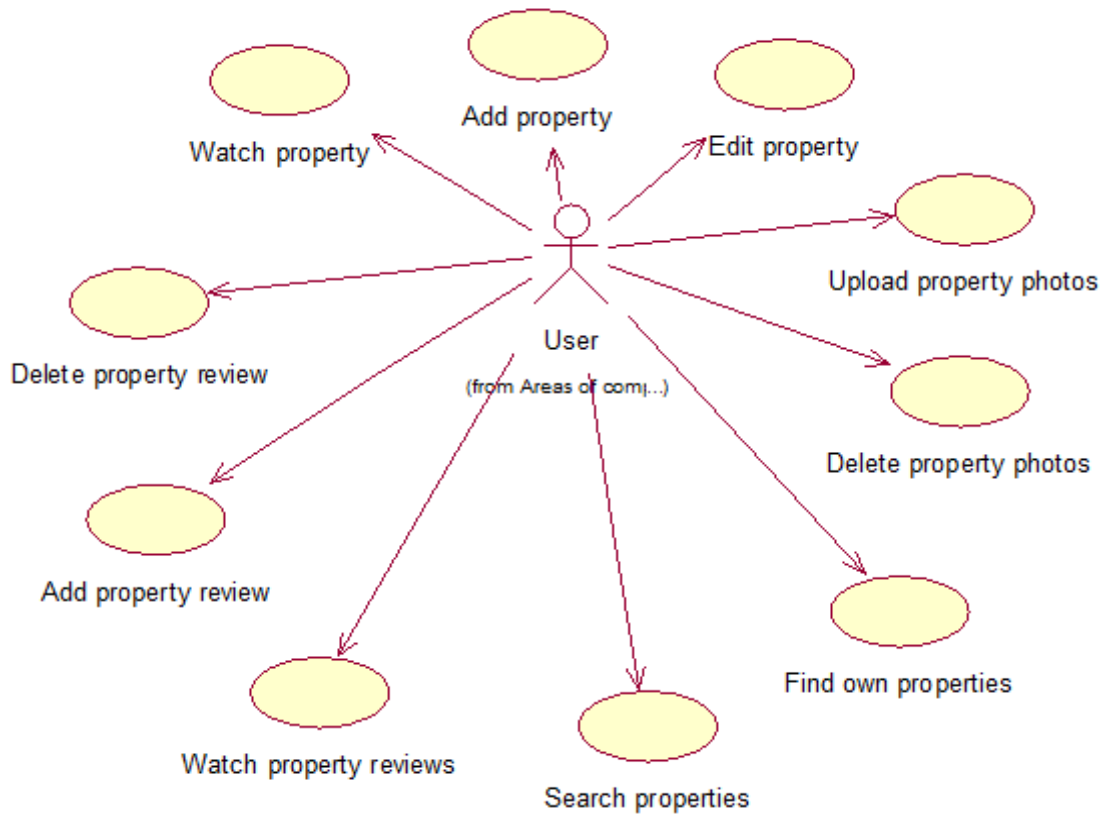
In order to ensure the correct work of the application, it is strongly recommended to use the latest versions of modern web browsers.

### **3.1 Project analysis**

Project analysis chapter presents project short analysis. Its main goal is to plan what functionality should be implemented and what data structure is the most suitable for the needs of the application.

#### **3.1.1 Use case models**

The following use case diagram illustrates use cases that are related to property managing:



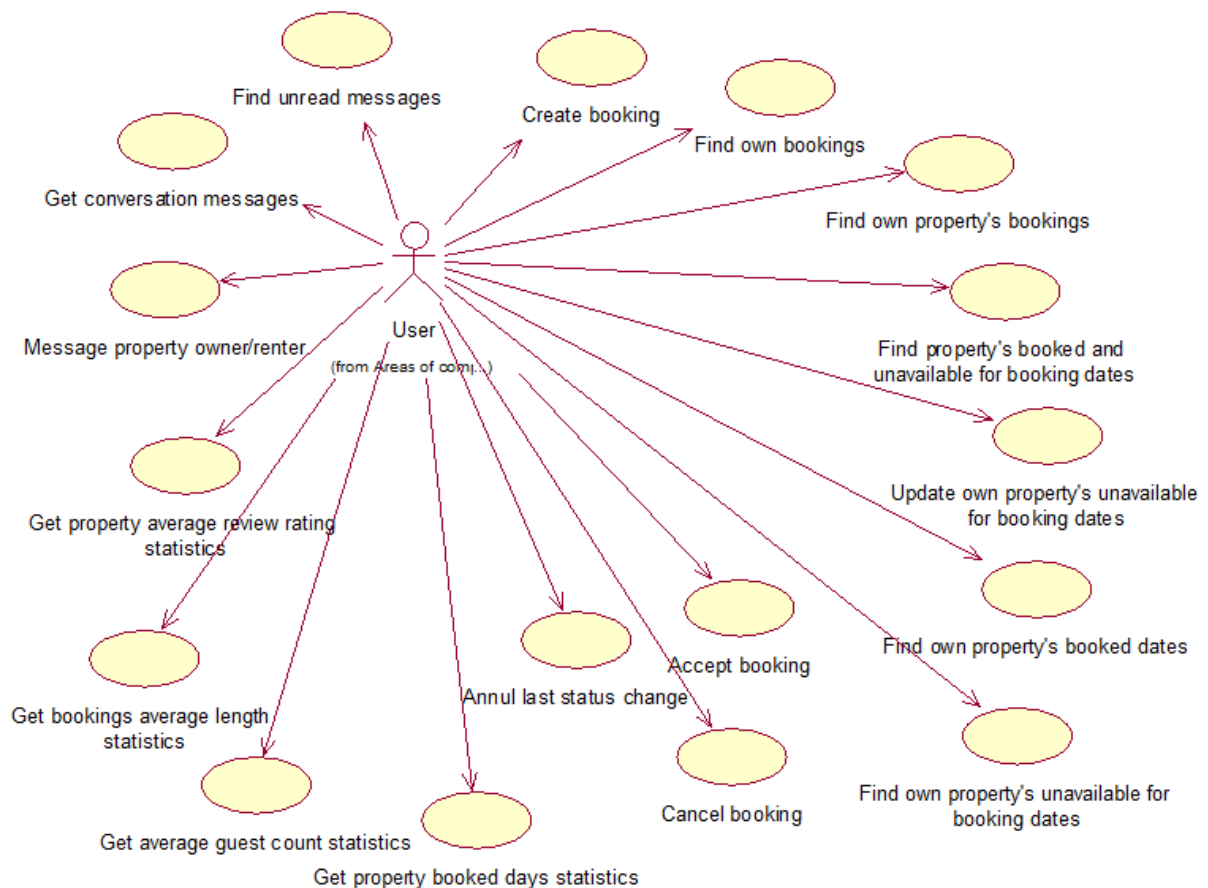
**Figure 3: Property managing use case diagram**

**Table 1: Description of property managing use cases**

Use case	Description
Watch property	Any user can select and check property information.
Add property	Registered user can add property.
Edit property	Property owner can edit his own existing property information.
Upload property photos	Property owner can upload property photos.
Delete property photos	Property owner can delete property photos.
Find own properties	Property owner can find all his properties.
Search properties	Any user can search properties by their location. Location is

	either country or city.
Watch property reviews	On each property information page, any user can read property reviews that were written by other users who have already booked a property.
Add property review	Registered user who booked the property can write review regarding the property he booked. Besides, property owner can write answer to other reviews. Initial review author cannot continue dialogue with property owner in the same discussion tree.
Delete property review	Registered user can delete his own review whenever he wishes to do so.

Figure 3 describes booking related actions:



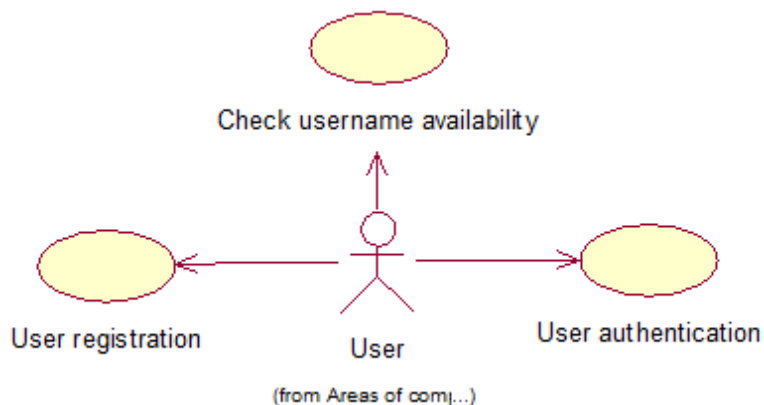
**Figure 4: Booking managing use case diagramm**

**Table 2: Description of booking managing use cases**

User case	Description
Create booking	Registered user can book property.
Find own bookings	Registered user can find all their bookings.
Find own property's bookings	Property owner can find bookings related to his specific property.
Find own property's booked and unavailable for booking dates	Any user can check dates when property is booked or unavailable for booking. These dates are grouped (user cannot distinguish them).
Update own property's unavailable for booking dates	Property owner can update his property's unavailable for booking dates.
Find own property's booked dates	Used as part of "Update own property's unavailable for booking dates" use case. Property owner cannot select these dates.
Find own property's unavailable for booking dates	Used as a part of "Update own property's unavailable for booking dates" use case. Property owner can freely select or deselect these dates.
Accept booking	Property owner can accept booking. Only booking with status "Created" can be accepted.
Cancel booking	Property owner can cancel booking. Canceled can be both "Created" and "Accepted" bookings.
Annul last status change	Used by property owner to annul last booking status change.
Get property booked days	Property owner can get statistics that shows amount of

statistics	booked days in each month for a specific property.
Get average guest count statistics	Property owner can get statistics that shows average guest count in each month for a specific property.
Get bookings average length statistics	Property owner can get statistics that shows average length of bookings in each month for a specific property.
Get property average review rating statistics	Property owner can get statistics that shows average rating of reviews in each month for a specific property.
Message property owner/renter	When registered user (renter) created a booking, he can message property owner or vice versa.
Get conversation messages	Property owner or renter can get conversation history of a specific booking.
Find unread messages	Every five seconds when registered user is logged in, application checks whether he has new messages or not.

There are also specified use cases for user authentication and registration:



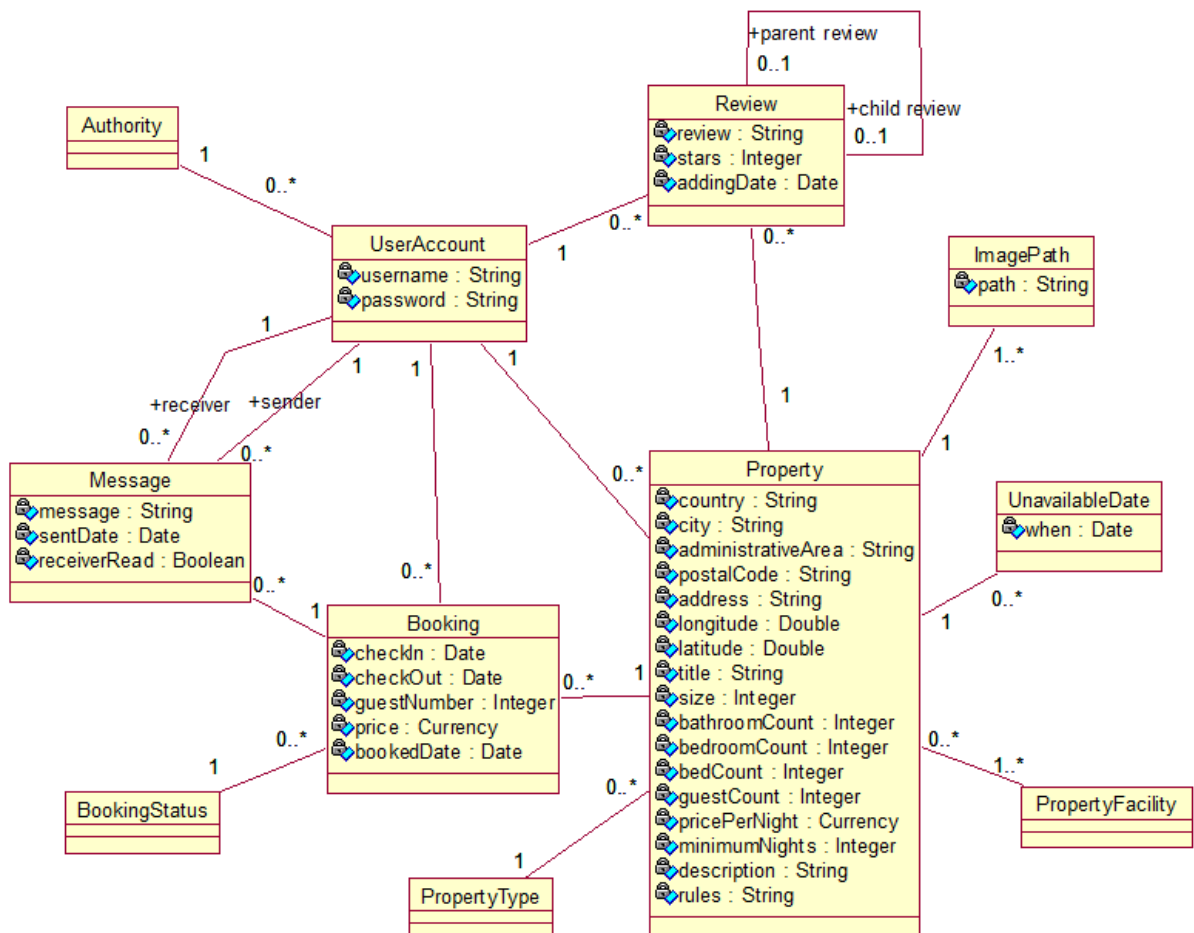
**Figure 5: Account managing use case diagram**

**Table 3: Description of account managing use cases**

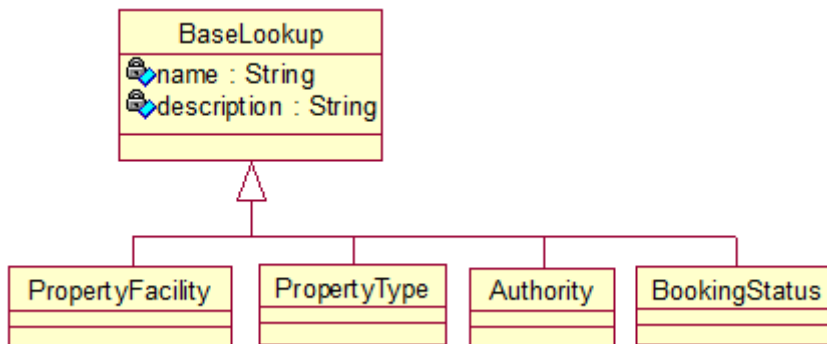
Use case	Description
User registration	Anybody can register a new user. All registered users have authority ROLE_USER, since there is no others areas of compitences in the current project.
User authentication	In order to log in as the registered user one should provide valid credentials.
Check username availability	While registration process user can check whether his username is available for registration or has already been taken.

### 3.1.2 Entity-relationship diagram

The following diagram describes entities and their relationships between each other.



**Figure 6: Entity-relationship diagramm**



**Figure 7: Look-up entities diagramm**

Table 4 describes each entity more closely.

**Table 4: Defenition of entity types**

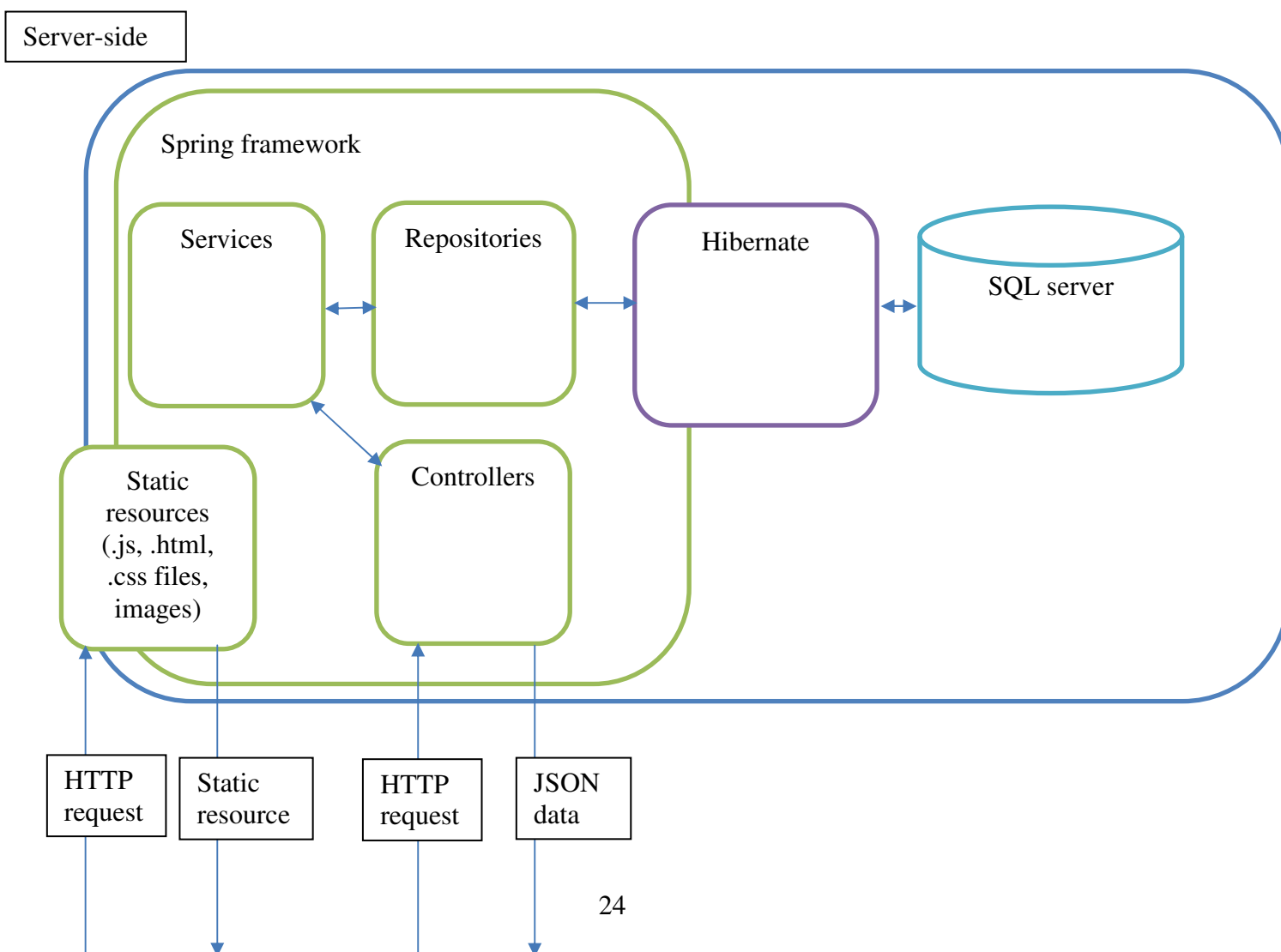
Name of the entity type	Definition
Property	Property is a real estate that a owner would like to rent out.
ImagePath	Contains necessary information about image that is related to a property.
UnavailableDate	Property’s date that is unavailable for booking.
Booking	Booking is a specific dates when a renter would like to use a property at a given price.
Review	Review is a renter’s opinion regarding property that he booked or a property owner’s comment to this opinion.
Message	Message is a text that can be sent from property owner to renter or vice versa only after booking creation.
UserAccount	User account is a registered user that can be both renter and property owner.
PropertyFacility	Property facility is something that is provided at a property for renters to use.
PropertyType	Property type represents a quality that differs specific property from others.
BookingStatus	Represents current booking status.
Authority	A user role that defines what actions in the application user is allowed to take and which not.

### 3.2 Server-side implementation

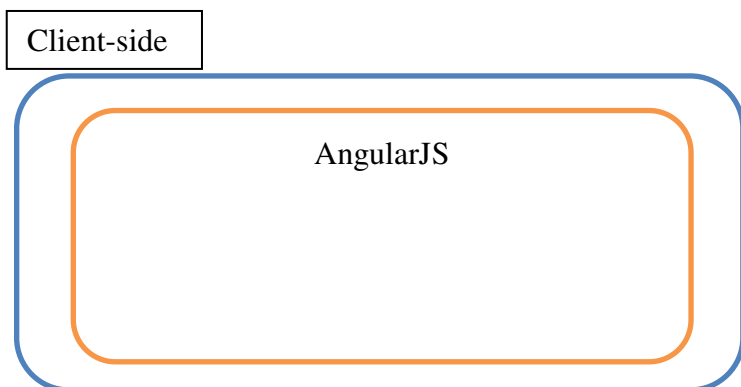
As it was already mentioned in introduction, in order to satisfy current project requirements the server should be able to serve and process JSON data. Since it is quite a common demand, almost every single technology can do that.

The server-side of the application is implemented using Java-based technologies. Apache Tomcat is used as HTTP server and Spring Framework is used as a MVC framework on the backend. Embedded SQL database H2DB is used as a data storage. In order to make interactions with database as simple as possible there is also used ORM framework – Hibernate, which runs in create-drop mode. In other words, the database schema is created from the mapped entities (and performed necessary SQL insert queries) on application startup and dropped when application is stopped. The entities are mapped corresponding to 3.1.3 Class diagram. Such way of building server-side will enable to focus mostly on the application front end, since detailed analysis of server-side implementation is beyond the scope of this thesis.

Overall application architecture looks like this (with emphasis on application back end):







**Figure 8: Overall application architecture**

The server receives HTTP requests from application front end. First of all, front end requests static resources, for example, JavaScript, HTML, CSS files, images. If Angular application sends request to the server REST endpoint, then the request is handled by Spring controller. All the server business logic is performed in Service layer. In the repository layer are done the database related actions.

### 3.2.1 Application REST endpoints

Typically, all mentioned use cases in the chapter 3.1.1 correspond to specific REST endpoint of the application.

**Table 5: Application REST endpoints**

URL	HTTP Verb	POST Body	Authority required	Use case
/api/properties/:id	GET	Empty	None	Watch property
/api/properties/:id	PUT	JSON String	ROLE_USER	Edit property
/api/properties	POST	JSON String	ROLE_USER	Add property
/api/properties/myProperties	GET	Empty	ROLE_USER	Find own properties
/api/properties/search	GET	Empty	None	Search properties
/api/properties/propertyTypes	GET	Empty	None	
/api/properties/propertyFacilities	GET	Empty	None	
/api/properties/uploadPhotos	POST	Bytes	ROLE_USER	Upload property photos

/api/properties/reviews/:id	GET	Empty	None	Watch property reviews
/api/properties/reviews/:id	POST	JSON String	ROLE_USER	Add property review
/api/properties/reviews/:id	DELETE	Empty	ROLE_USER	Delete property review
/api/bookings	POST	JSON String	ROLE_USER	Create booking
/api/bookings/myBookings	GET	Empty	ROLE_USER	Find own bookings
/api/bookings/myPropertiesBookings	GET	Empty	ROLE_USER	
/api/bookings/bookedDaysStatistics/:id/:year	GET	Empty	ROLE_USER	Get property booked days statistics
/api/bookings/bookingAvgGuestCountStatistics/:id/:year	GET	Empty	ROLE_USER	Get average guest count statistics
/api/bookings/bookingAvgRatingStatistics/:id/:year	GET	Empty	ROLE_USER	Get property average review rating statistics
/api/bookings/bookingAvgLengthStatistics/:id/:year	GET	Empty	ROLE_USER	Get bookings average length statistics
/api/bookings/myPropertyBookings/:id	GET	Empty	ROLE_USER	Find own property's bookings
/api/bookings/unavailableDates/:id	GET	Empty	None	Find property's booked and unavailable for booking dates
/api/bookings/onlyUnavailableDates/:id	PUT	JSON String	ROLE_USER	Update own property's unavailable for booking dates
/api/bookings/onlyUnavailableDates/:id	GET	Empty	ROLE_USER	Find own property's unavailable for booking dates
/api/bookings/onlyBookedDates/:id	GET	Empty	ROLE_USER	Find own property's booked dates
/api/bookings/bookingStatuses	GET	Empty	ROLE_USER	
/api/bookings/bookingStatus/:bookingId/:statusId	GET	Empty	ROLE_USER	Accept/Cancel booking, Annul last status change
/api/bookings/canSendReviews/:propertyId	GET	Empty	ROLE_USER	
/api/messages/:bookingId	GET	Empty	ROLE_USER	Get conversation messages
/api/messages	POST	JSON String	ROLE_USER	Message property owner/renter
/api/messages	GET	Empty	ROLE_USER	Find unread messages
/api/messages/markRead/:bookingId	GET	Empty	ROLE_USER	
/api/accounts	POST	JSON String	None	User registration

/api/accounts/username/:username	GET	Empty	None	Check username availability
/login?username=:username&password=:password	POST	Empty	None	User authentication

### 3.3 Setting up development environment

Web development advances nowadays extremely quickly so do various development tools that help programmers to write code more swiftly and without great effort manage project dependencies, builds and tests.

Firstly, as the main integrated development environment will be used Eclipse Enterprise Edition. It is pretty suitable for Java back end and JavaScript front end development. In order to make Angular application development a little bit more productive, there are installed some useful Eclipse plugins. These include: AngularJS Eclipse and TM Terminal. AngularJS Eclipse plugin enables AngularJS support in the JavaScript editor. TM Terminal integrates command line terminal into Eclipse. Default Eclipse solutions do not suit for interactive work with command line utilities such as Bower or Grunt.

The most JavaScript tools require Node.js platform in order to operate. Node.js is an open source, cross-platform runtime environment that uses Google V8 JavaScript engine, therefore the most Node.js applications are written in JavaScript [7]. This platform dramatically widens JavaScript capabilities by providing modules for handling file system I/O, various networking modules and other essential functions moving JavaScript to a whole new level. Moreover, Node.js comes bound together with npm. Npm – is a package manager for Node.js applications that will be used during development process. Please note, that tools downloaded with npm manager are mostly based on Node.js, hence are not front end libraries or frameworks, which web browsers can execute, but standalone applications.

Since large projects have typically lots of dependencies, it takes a lot of time to fetch and install all necessary packages. Nowadays there are several front end package managers, but by far the most popular is Bower. Although it is command line utility (as the most JavaScript development tools), it is pretty simple and straight-forward.

To start with, Bower requires Git to be installed on the computer. In order to install globally Bower package manager the following command should be executed in the command line:

```
npm install -g bower
```

To install a package you only need to run in the command line:

```
bower install angular#1.3.15
```

Bower will resolve all necessary packages (it also checks whether installing package's necessary dependencies are present or not) in `bower_components` folder. During the project development there will be used current stable version of AngularJS framework – 1.3.15. The list of project Bower dependencies may be found in Appendix 1.

Next, we need to globally install the JavaScript task runner called Grunt with npm package manager.

```
npm install -g grunt
```

Its main goal is to perform repetitive tasks such as file copying and concatenation, code minification and testing. In other words, Grunt automates workflow. This allows web developer to concentrate more on the project itself. Grunt is extremely flexible and versatile: there are plenty of configuration options in `Gruntfile.js`. What really makes it outstanding tool is the fact that it has lots of plugins which can perform a great deal of different actions. On the other hand, in contrast to Bower, it takes quite a time to understand how this tool actually works and how to use it, since it requires manual configuring. Nevertheless, time spent on configurations pays off. The list of project Node.js dependencies may be found in Appendix 2.

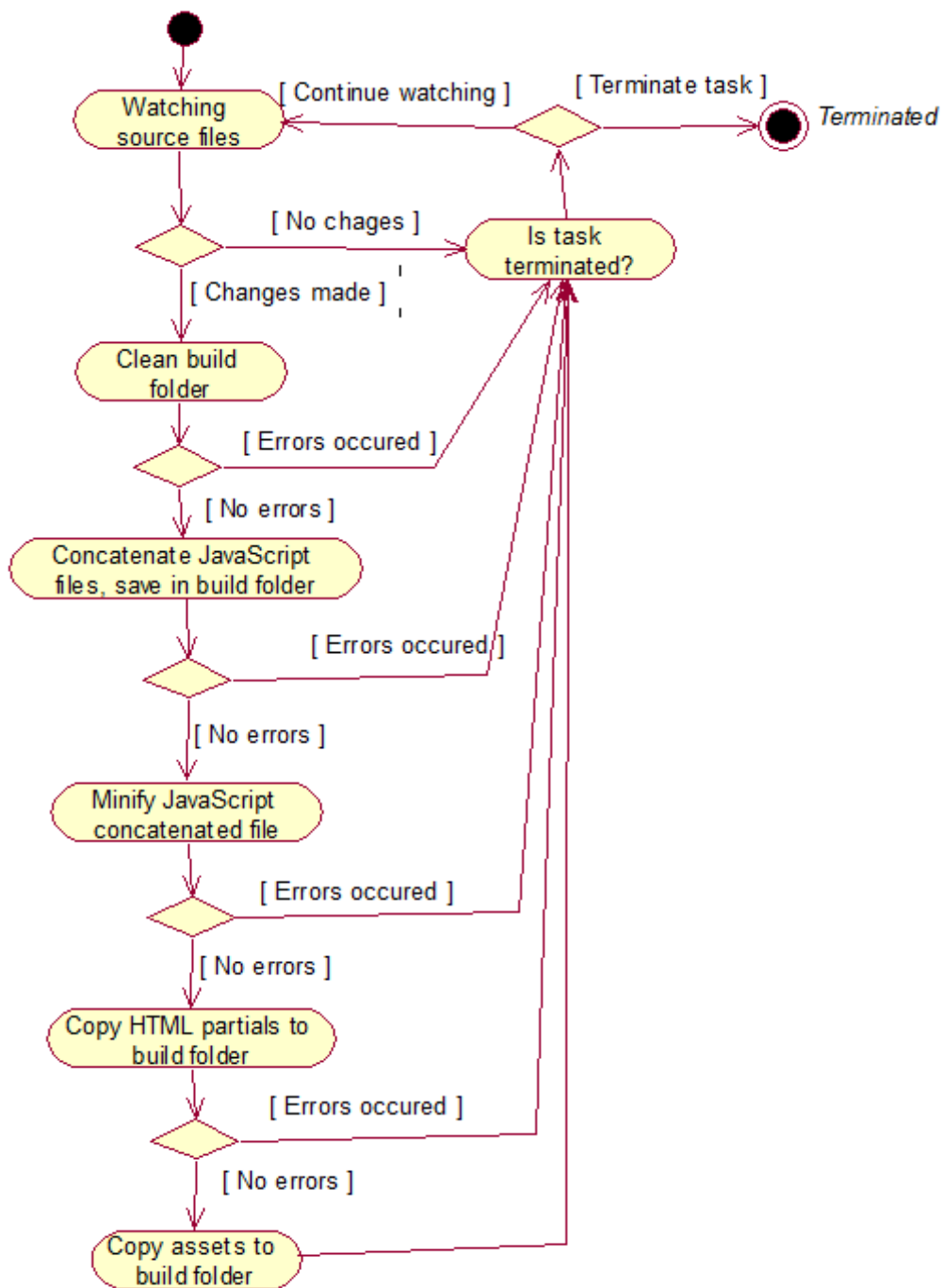
It is worth mentioning that there are actually plenty of premade AngularJS project structures such as `ngBoilerplate`, `angular-seed` and so long. Their main aim is to provide developer with everything they need in order to start coding. It frees developer from doing repetitive actions required for each new project: configuring Grunt, Karma, organizing project structure. Although these application skeletons do bring a lot to the kick starting of the Angular-based application, during this thesis there will not be used any of those. On the contrary, everything will be configured and organized manually so as to better understand the basics of the setting up JavaScript developing environment.

In this project grunt is configured to run two tasks:

grunt

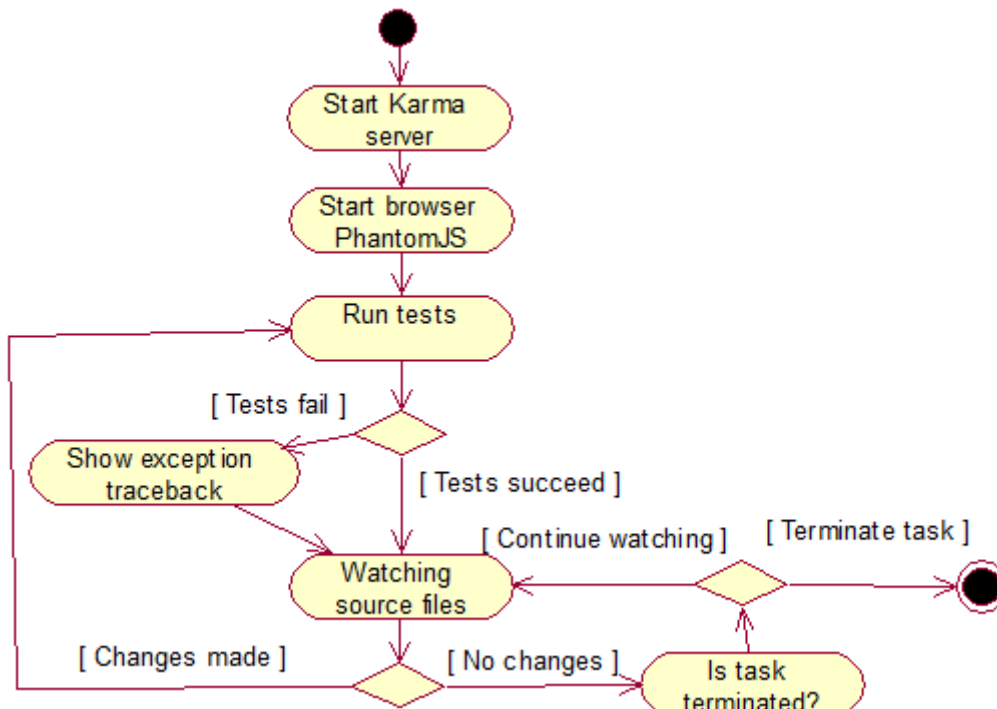
grunt test

During workflow, when default grunt task is run, it performs following actions on any file change:



**Figure 9: Default grunt watch task activity diagram**

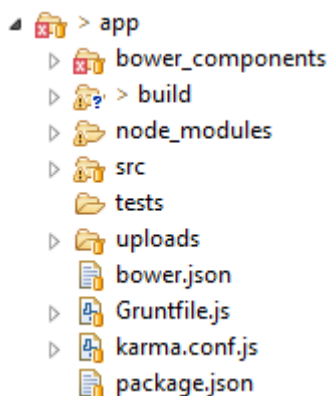
There is also separate task for a test runner:



**Figure 10: Grunt test task activity diagram**

Both tasks are running during development process simultaneously in different command line windows. They are fired on each changed made in the source code and produce their results.

As the result, the project static files structure:



**Figure 11: Project static files structure**

Let us take a look at each project directive more closely.

### Bower components folder

This folder contains project's front end dependencies.

### Node components folder

This directory contains various Node.js dependencies such as Grunt, Karma and Jasmine.

### Src folder

The project's front end source code is located in this folder.

### Build folder

This folder contains the final minified version of the project. It is served to users, not the one from /src folder.

### Tests folder

This directory contains project test files.

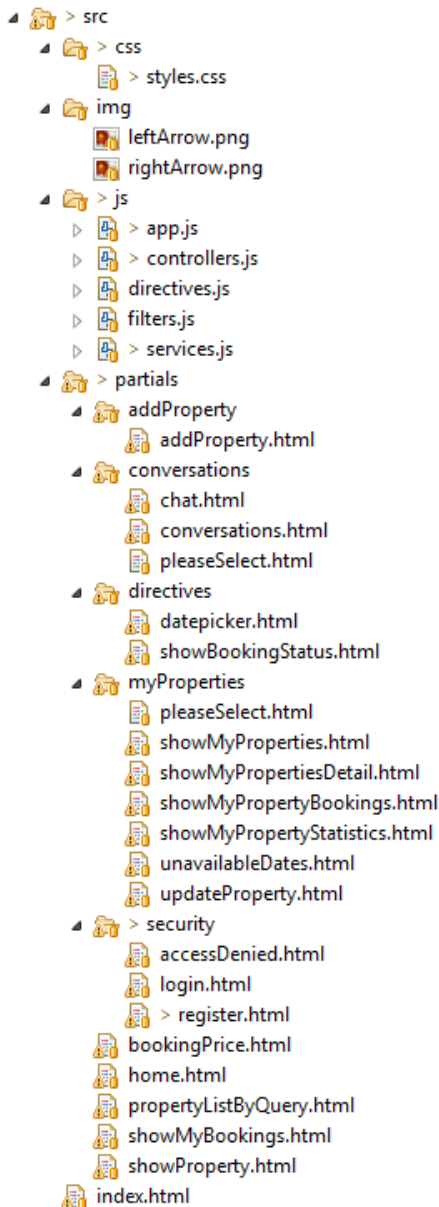
In addition, there are several configuration files for Bower (bower.json), Grunt (GruntFile.js), Karma (karma.conf.js) and Npm (package.json).

## **3.4 Project sources structure organization**

In the previous chapter were examined tools that were used during development and overall project static files structure. In this chapter we will take a look at organization of source code structure.

The structure of project has a great influence on the development process. The influence may be positive or negative depending on structure type and complexity of the project. In this chapter will be shown and analyzed both plain project structure and more advanced version that is able to cope with the growing complexity of the application.

Initially the project structure of the application's front end was just like this:



**Figure 12: Initial Angular application source code structure**

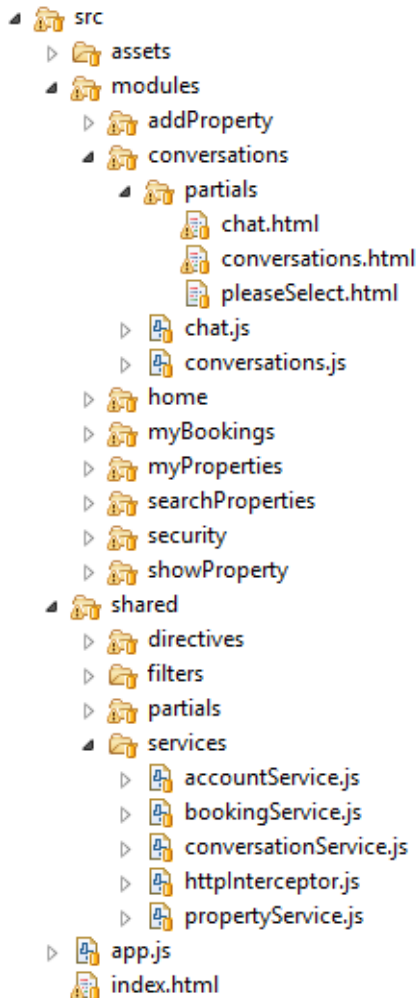
This is a typical web app structure: HTML partials have their own folder (`src/partials/`); JavaScript code is spread among files with corresponding names in `src/js/` directory; there are separate folders for CSS files and images.

At early stages of the development, such structure allowed to code rapidly. However, when the application started to grow it turned out that such project architecture is not scalable enough: the code maintainability was extremely poor, refactoring was quite hard. In fact, it desperately needed massive overhaul for a variety of reasons.



Firstly, there have been over 1200 lines of code in one `src/js/controllers.js` file, more than 350 lines of code in `src/js/directives.js` and in `src/js/services.js` JavaScript files. When it comes to main application file – `src/js/app.js` it had about 300 lines of code. Next, some partials had their own folders and some did not. All these facts made development process much harder and time consuming.

Improved version of the project structure [8]:



**Figure 13: Improved Angular application source code structure**

At the first glance, this project structure is much more complicated than the previous one, but it does provide a lot of benefits for a number of reasons.

### App.js and index.html files

These files are located at the root of the front end structure. `index.html` – is main html file that loads all the necessary files of the application. It also holds common elements that share all

pages of the application (for example, header and footer inclusions). App.js – is main JavaScript file that is responsible for AngularJS application setup.

It is necessary to mention that app.js does not contain application routing rules (configured with ui-router – routing framework for AngularJS) – they are defined in separate modules of the application. Such way of organizing application states ensures complete modularity of the application: there are quantity of absolutely standalone modules that do not depend on main app.js file and vice versa. In the main app.js file, these modules only needed to be declared as dependencies.

### Assets folder

This folder contains CSS files and images that are used by the application.

### Modules folder

The modules folder contains the actual sections of the application. Each folder represents one separate state and its nested states. JavaScript files contain state configurations and controller for a single state (whether it is parent or nested state). Folder partials contain view templates for specific states. On the one hand, it was possible to go even further by creating separate subfolders for different nested views of a parent view but it is not really necessary in case of the current project.

### Shared folder

The shared folder contains reusable components of the application: directives, filters and services. Please note, that there are several service files, as distinct from first application structure, where only one service file is used. The same way of grouping and partition is used in both directives and filters directories.

Using such project organizational approach, we decouple different component of the application, guaranteeing modularity and scalability.

To sum up, there are great varieties of different approaches that may be suitable for all sorts of projects: whether it is small tutorial or complicated system. One however should take into consideration each approach's advantages and drawbacks before deciding which structure to follow. There is no agreed upon approach how to organize AngularJS app structure. The

above-described final structure of the application is just one of the numerous variations, but it fully fulfilled this project requirements.

### 3.5 AngularJS application architecture

AngularJS framework provides a lot of flexibility and good support of separation of concerns: presentation logic, business logic and presentation state are nicely separated from each other. As it was stated by Igor Minar, one of the key developers of AngularJS framework, AngularJS application architectural pattern used to be closer to MVC (Model View Controller) pattern, but now it is closer to MVVM (Model View ViewModel). On the other hand, he prefers to call AngularJS to be so-called MVW (Model View Whatever) framework – it does not strictly categorize framework, but identifies it just with MV\*-type frameworks [9].

The diagram illustrates the architecture of the application [10].

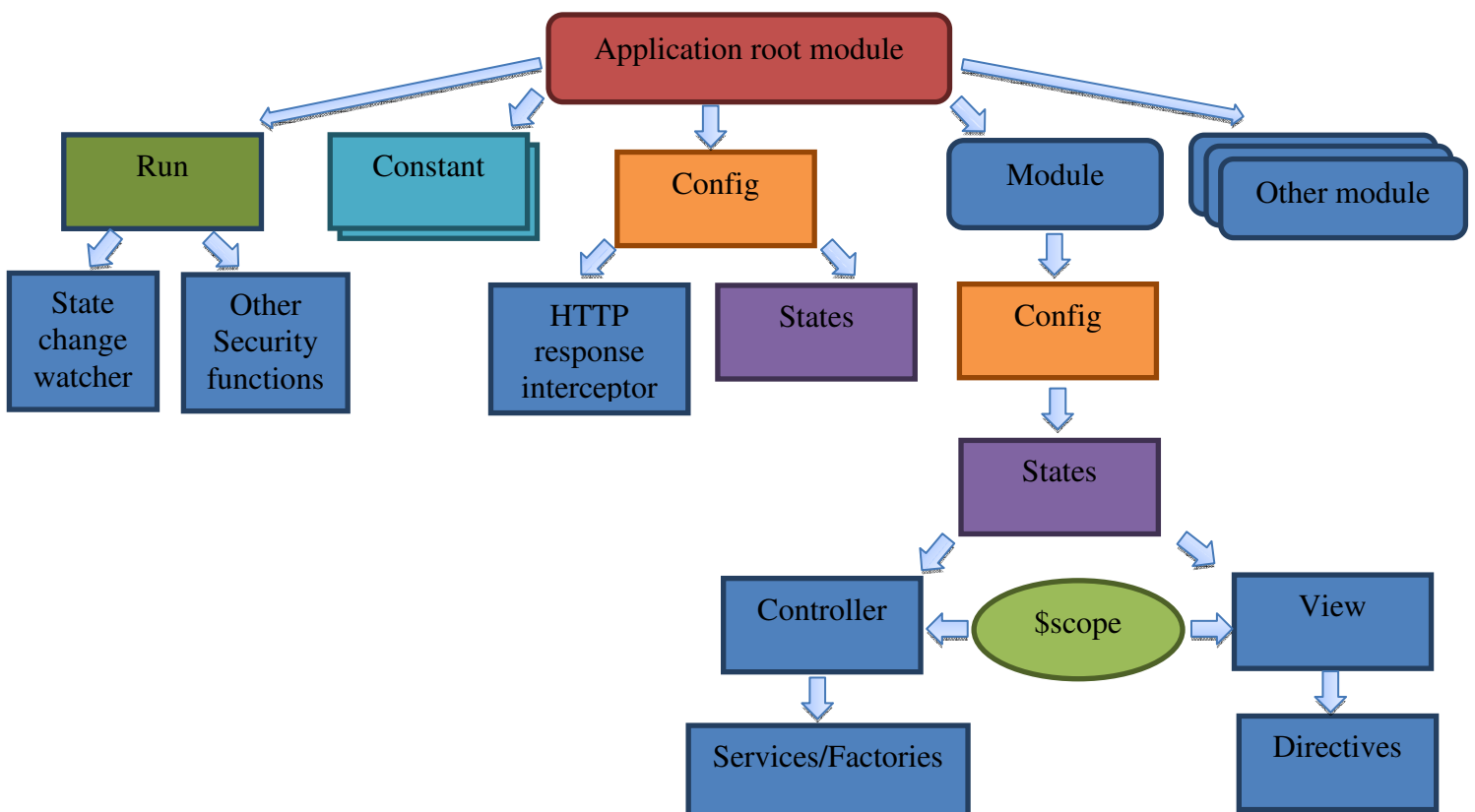


Figure 14: Angular application architecture

The application root module consists of following components: application modules, constants declarations, configuration phase and run block.

Firstly, in the application root module are declared all used submodules.

Secondly, when AngularJS bootstraps it applies all declared constant definitions. In the current application there are two constants which contain application URL and common part of the REST endpoints URL.

Therefore, AngularJS applies registered configuration blocks. Root application configuration block contains a default state which user is redirected to if requested path does not match any of the URLs specified. There is also declared HTTP responses interceptor that catches responses from the server with error codes 401 Unauthorized, 403 Forbidden and behaves accordingly.

Run block is the code that runs once application has been configured. There is declared state change watcher that on each state change alters page title and performs necessary security checks determining whether user is authorized to access given state or not.

Each submodule is able to use full list of angular features. In order to raise decoupling of application modules states are configured not in the application root module but rather in the concrete submodule. There are also configured controllers for a specific state.

The main goal of the state is to bind controller and view together by creating \$scope object that share controller and view.

When controllers are initialized Angular dependency injection subsystem provides controllers with necessary services and factories. Angular also compiles directives in the templates to produce view.

### **3.5.1 Dependency injection**

Dependency injection is cornerstone of AngularJS framework that makes it possible to create loosely coupled applications [11].

The most application components are not standalone and do depend on other modules.

Dependencies are typically defined in inline array annotation:

```
1. propertyService.factory("PropertyService", [ "$resource", "API_URL", function($resou  
   rce,API_URL) {  
2.     //factory code  
3. }]);
```

Module dependencies can also be defined without passing an array which consists of dependency names and function but just one function with its parameters.

```
1. propertyService.factory("PropertyService", function($resource,API_URL) {  
2.     //factory code  
3. });
```

However, such approach has a disadvantage: in this case, after application code minification all the dependency names will be lost and dependency injector will not be able to find required dependencies. On the contrary, the first code block will work properly since string annotations are not minified and AngularJS is able to locate necessary dependencies:

```
1. propertyService.factory("PropertyService",["$resource","API_URL",function(a,b){...}]  
   );
```

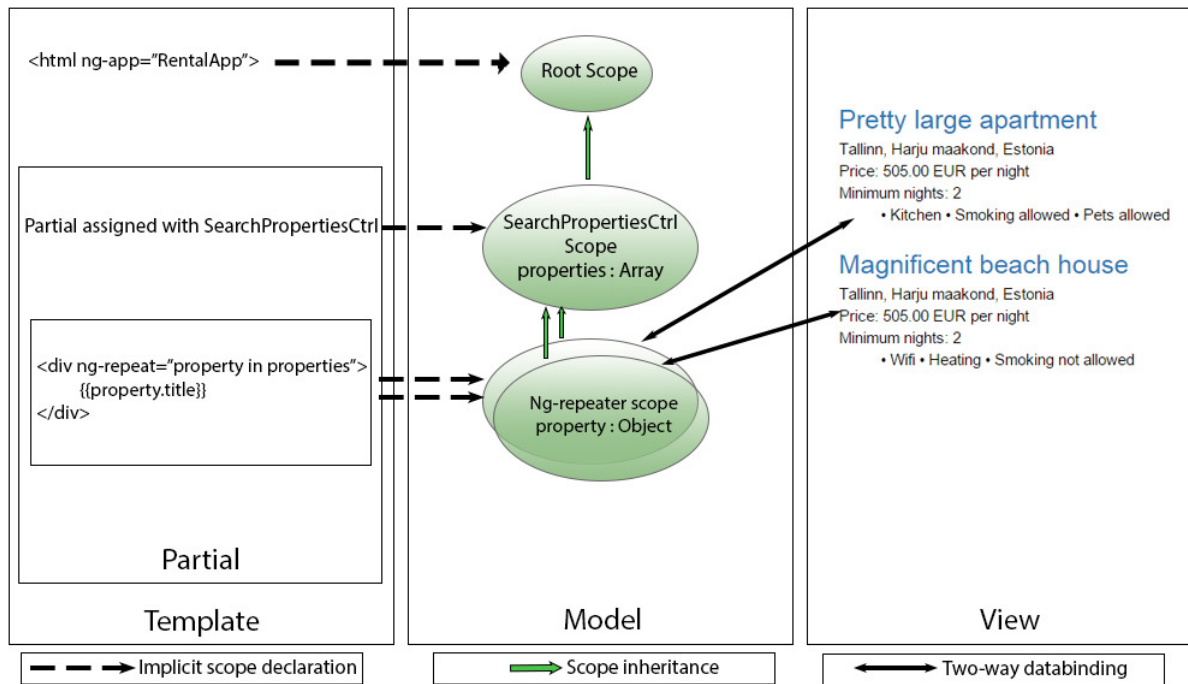
Because of the fact that the project code minification is performed by Grunt, only the recommended way of declaring module dependencies is used in the project.

### 3.5.2 Scope object

When it comes to \$scope object, it is undoubtedly crucial part of the AngularJS application. It is so called glue between controller and view – a layer where all the current variables and functions are stored. Both controller and view can access same scope.

Scopes can be nested as much as needed in order to limit access to some application properties while providing access to shared model properties [12].

Let us have a look at the following diagram to understand how scopes work [13]:



**Figure 15: Angular application scope object workflow**

First, there is root scope that is globally accessible in any module of the application. It is created as soon as application root module is specified.

When each controller is loaded it creates own child scope that inherits all properties from the root scope. Moreover, when AngularJS in the HTML partial (which is assigned using ui-router – AngularJS routing module) compiles directive ng-repeat it creates child scope for this directive. However, one should be aware that the child scope properties are not accessible from the parent scope. Yet it allows child scopes to use their own private properties without interfering with parent scope.

Whenever controller or view makes change in the scope object, the other side is notified about it and gets updated data [14]. This is how two-way data binding between controller and view works. It is performed automatically by Angular.

### 3.6 Third-party JavaScript libraries integration

Although AngularJS provides a great deal of built-in functions and objects it cannot always alone fulfil all the project requirements. Therefore rises the need for other JavaScript libraries.

Typically, libraries that do not change HTML DOM and are only accessed in JavaScript code can freely be used in all AngularJS modules. For instance, Moment.js – a JavaScript library that makes manipulations with dates much more convenient than the standard Date JavaScript object.

When it comes to libraries that do need to manipulate HTML DOM, they are usually implemented in the Angular directives. In fact, it is recommended to integrate them in the application with directives in order to improve Angular application decoupling and testability. Directives – are HTML DOM elements that tell AngularJS’s HTML compiler to attach a specified behavior to that DOM element [15].

Consider the following example. During the project development, in order to successfully implement “Update property’s unavailable for booking dates” use case there was a need in using bootstrap-datepicker external JavaScript library that enables multirate picking. The datepicker is configured in a jQuery-way:

```
1. $(' .datepicker').datepicker({
2.     orientation:"top auto",
3.     startDate: '0d'
4. })
```

Since using jQuery in the Angular controllers is considered as bad practice, it should be moved into a separate directive:

```
1. propertyDirective.directive("myDatepicker",function(){
2.     return {
3.         restrict:"E",
4.         scope:{
5.             unavailableDates:"=",
6.             bookedDates:"=",
7.             updateUnDates:"&"
8.         },
9.         templateUrl:"shared/directives/partials/datepicker.html",
10.        link : function(scope){
11.            //a lot of code omitted
12.            $(' .datepicker').datepicker({
13.                orientation:"top auto",
14.                startDate: '0d'
15.            })
16.            //a lot of code omitted
17.        }
18.    };
19. });
```

Firstly, this particular directive can only be used as a HTML element (not attribute inside other tags for example, line 3). Secondly, the directive defines its own isolated scope (not child scope, lines 4-8). It means that it does not inherit any properties from parent scope but

operates only with the passed to the directive variables and functions. All jQuery logic is moved into the directive link function (lines 10-17).

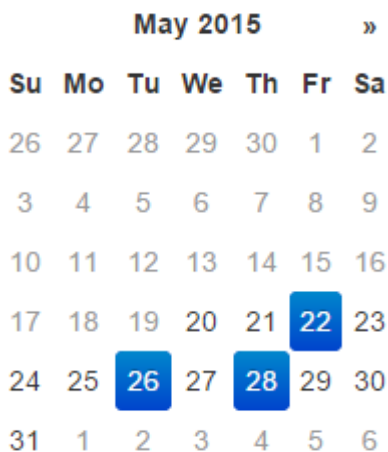
Besides, directive also references to a separate template (line 9) that is used by jQuery to produce datepicker:

```
1. <div class="input-append date">
2.   <div class="datepicker" data-date="" data-date-format="dd/mm/yyyy" data-date-
   multodate="true"></div>
3.   <input type="hidden" id="datepicker_data_input"/>
4. </div>
```

The directive is used in the application templates in a following way:

```
1. <my-datepicker class="col-md-12" ng-
   if="currentUnDates.$resolved && currentBookedDates.$resolved"
2.   update-un-dates="updateUnDates(dates)"
3.   unavailable-dates="currentUnDates"
4.   booked-dates="currentBookedDates"></my-datepicker>
```

The result of compilation of my-datepicker directive is shown below:



May 2015							»
Su	Mo	Tu	We	Th	Fr	Sa	
26	27	28	29	30	1	2	
3	4	5	6	7	8	9	
10	11	12	13	14	15	16	
17	18	19	20	21	22	23	
24	25	26	27	28	29	30	
31	1	2	3	4	5	6	

### Figure 16: My-datepicker directive produced result

As seen from above, directives are just part of HTML that perform some hidden from user actions. The user only needs to specify required attributes. Moreover, AngularJS directives are extremely flexible – they can be nested as much as necessary, for example, ngIf directive plays here crucial role: it does not allow AngularJS to compile myDatepicker directive as long as there is not resolved data from the server. Otherwise, the directive will receive empty array and produce nothing.



### 3.7 AngularJS application security

There are various ways to implement authentication and authorization in AngularJS-based application. They depend on level of security needed, server-side technologies used and how scalable application should be.

For the purposes of the current project there will be used authentication via cookies and session. The reason for this is that there are actually only one client: web browser that is able to operate cookies. Were there other clients such as smartphones, other web services this approach would not be so elegant. This is one of the reasons why HTTP sessions usage is not considered as the best practice in the REST applications.

As a server-side security framework is used Spring Security, since it fulfils all the requirements, though it needs a bit of modification in its standard behavior to work with single page application [16][17].

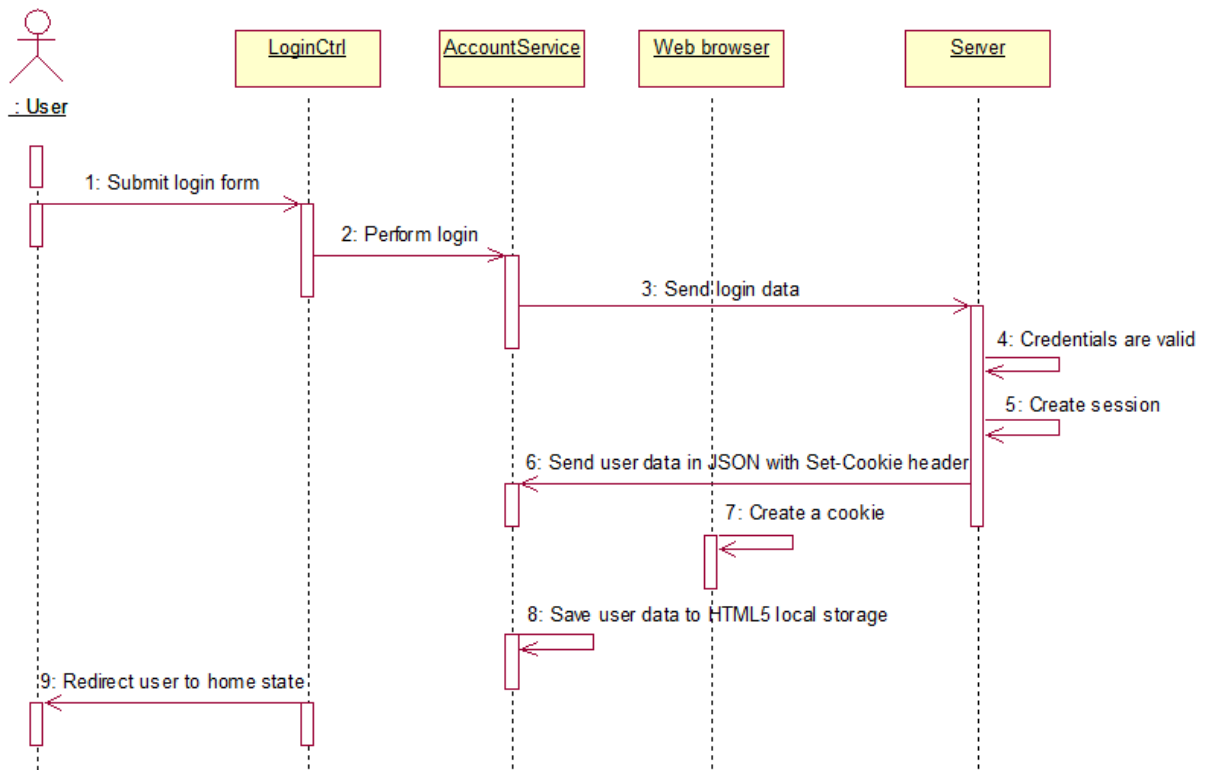
The authentication is implemented in a following way:

In the single page application current logged in user's username and authority is stored in a web browser's HTML5 local storage.

When user attempts to log in his credentials are sent to the following REST endpoint:

**Table 6: Application login REST endpoint**

URL	HTTP Verb	POST body	Authority required	Use case
/login?username=:username&password=:password	POST	Empty	None	User authentication



**Figure 17: Successful authentication sequence diagram**

As it shown in Figure 16, if user with provided credentials exists, server creates new session. Then, it returns HTTP response with logged in user's username and authority. Response also contains Set-Cookie header. User related data is then saved in HTML5 local storage by AngularJS application. In addition, because of Set-Cookie header, web browser automatically creates new cookie with received from server values. If authentication fails, server returns error code 401 Unauthorized and authentication is not performed.

In order to restrict users from accessing states that they are not allowed to enter in each state configuration are specified required authorities:

```

1. $stateProvider.state("addProperty",{
2.     url:"/addProperty",
3.     //code omitted
4.     data : {
5.         authorities:['ROLE_USER']
6.     }
7. });
  
```

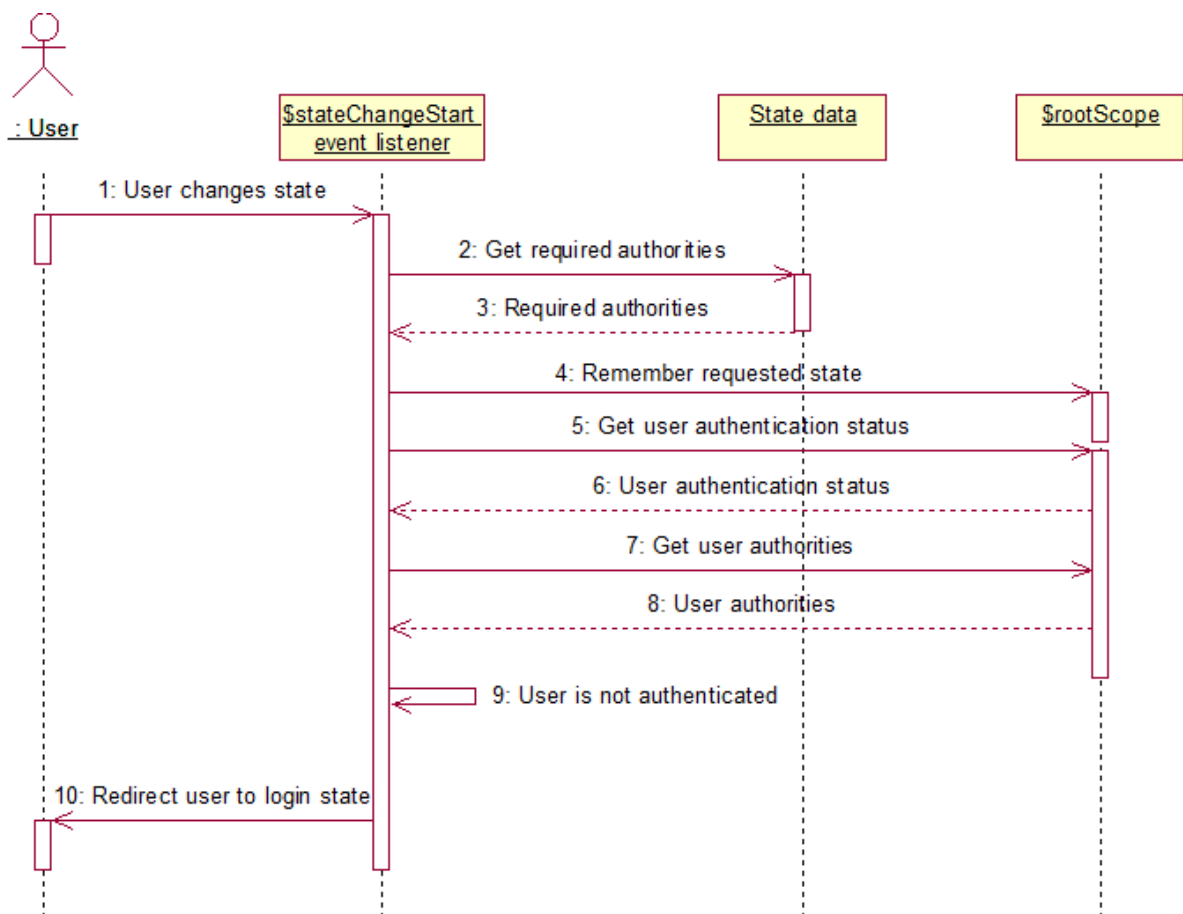
This information is used by a state change watcher, which runs in the root of the application:

```

1. $rootScope.$on('$stateChangeStart',
2. function(event, toState, toStateParams, fromState, fromStateParams){
3.     //code omitted
4. });
  
```

Although all registered users in the application have same authority (ROLE\_USER) authority check is still implemented just to show one of the ways of restricting users to access some application states in AngularJS application.

The following sequence diagram shows what actions are performed if user is not authenticated.



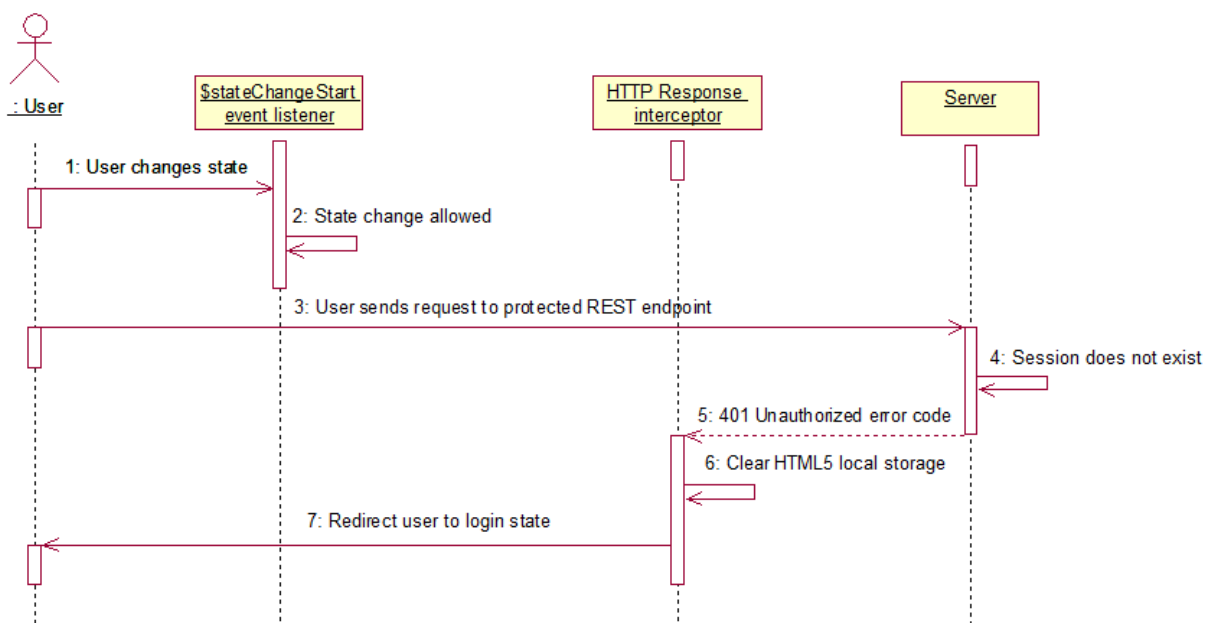
**Figure 18: Security checks on state change sequence diagram**

To begin with, there are defined \$stateChangeStart event which is fired when state transition begins. Firstly, it gets what authorities are required to access given state. Secondly, it gets current user authentication status and authorities. Then it checks whether current user authenticated or not. If user is not authenticated, then the requested by user state is saved in the application \$rootScope and user is redirected to login page. The requested state is saved in order to redirect user to it after successful authentication.

Theoretically, there might also happen such situation, that user is authenticated but does not possess required authorities. In this case he is redirected to access denied state. However, in the current project all registered users have same role, therefore it can never happen. In fact, there is no need in role control in the project front end but it is still implemented just to show that user roles control is possible to implement in the AngularJS-based application.

There is also absolute necessity in controlling user's authorities on the server-side of the application. There are several reasons for that. Firstly, users can access server REST endpoints directly bypassing the front end. Moreover, user can open application after session expiration but with still existing data in HTML5 storage, therefore frontend will not notice any changes and treat current user as a logged in user. This is completely undesirable behavior.

In order to resolve these issues the server on each request checks user authorities and if they are not sufficient returns 401 Unauthorized or 403 Forbidden error codes. This not only solves the first problem but also prepares the ground for the solution of the second problem. Consider the following diagram:



**Figure 19: AngularJS http response interceptor sequence diagram**

Front end needs to get to know that user is not actually logged in and clear HTML5 local storage. This may be achieved with configured in the application root HTTP Response

interceptor. Whenever Angular application receives 401 Unauthorized error from the server (as the result of actually unauthorized user tries to access secured REST endpoint), it clears stored in local storage information and redirects user to login state. In case of error 403 Forbidden (if user does not possess sufficient authorities) it simply redirects to access denied error page.

## 4. AngularJS application testing

### 4.1 Unit testing tools

The Angular way of separating different components of the application makes it easy to test a great variety of application modules and components.

As a main testing framework will be used Jasmine in conjunction with Karma and Grunt.

Jasmine – is an open source unit testing framework for JavaScript. It does not depend on other JavaScript frameworks or web browsers. It is able to test both JavaScript versions used in website development and JavaScript used in Node.js-based applications.

Tests written with Jasmine are usually executed using Karma. Karma – is a JavaScript test runner that tells us which tests failed or passed.

Karma has pretty good integration with Grunt which allows to run Karma using Grunt workflow tool. The activity diagram of Grunt testing task is shown in chapter 3.2 Setting up development environment.

All above mentioned tools are based on Node.js runtime environment therefore required to be fetched with npm package manager.

In order to test AngularJS application there should be Angular mocks package installed:

```
bower install angular-mocks
```

This package provides us necessary classes for unit testing.

With Jasmine it is possible to test a lot of different components of the application: controllers, custom directives, custom filters, services [18].

### 4.2 Testing Angular filters

For instance, we have simple custom filter that truncates text if its length is greater than the length given to the filter:

```

1. propertyFilters.filter('truncate', function(){
2.     return function(text,limit,showMore){
3.         //code omitted
4.     };
5. });

```

The filter's first parameter is text to be truncated, second parameter is maximum length of the text and showMore must equal string "Show more" otherwise text will not be truncated. The filter returns String value.

A typical Jasmine test:

```

1. describe('Truncate filter unit testing',function(){
2.     var $filter;
3.     var truncateFilter;
4.     var text = "Test me please";
5.
6.     beforeEach(function(){
7.         module('RentalApp');
8.     });
9.     beforeEach(inject(function(_$filter_){
10.         $filter = _$filter_;
11.         truncateFilter = $filter('truncate');
12.     }));
13.
14.     it('returns full text if not Show more', function(){
15.         expect(truncateFilter(text,5,"Show less").length).toEqual(text.length);
16.     });
17.     it('returns full text if Show more and length is greater than text
18.     length', function(){
19.         expect(truncateFilter(text,500,"Show more").length).toEqual(text.length);
20.     });
21.     it('returns truncated text if Show more and length less than text length', funct
22.     ion(){
23.         expect(truncateFilter(text,6,"Show more")).toEqual("Test...");
24.     });
25. });

```

Describe code block represents a test suite (a group of tests). It has two parameters: a string, that is a short description of the suite and a function, which is test implementation (line 1). There is no restrictions in nesting describe blocks.

beforeEach code block runs before each test execution (lines 6-12). Typically, it is used to perform test initial setup.

module('RentalApp') in the first beforeEach block indicates which module should be loaded (line 7). In this case, it is AngularJS application main module.

The inject function inside second beforeEach block main goal is to get necessary dependencies into the test via AngularJS dependency injector (lines 9-12). Please note the underscores (\_) at the beginning and the end of \$filter parameter. It is a little trick that is used

in order to preserve the same name (\$filter) of the variable inside the test suite. The injector unwraps underscores from around the parameter and injects correct dependency. To resolved truncate filter, we just pass its name to \$filter variable.

A single test is defined within it block which executable code ends with one or more expectations (lines 14-22). They are pretty much self-explanatory: expect function takes a value, called the actual, and chained with matcher function that receives expected value.

### 4.3 Testing Angular directives

Moreover, it is possible to test custom directives with Jasmine, for example, checking generated DOM.

For example, there are custom directive that counts the booking price and amount of nights and generates <span> tag using partial:

```
1. <span ng-if="totalPrice">{{nightCount}} nights - {{totalPrice}} EUR</span>
```

The test suite:

```
1. describe('Count booking price directive unit testing', function() {
2.   var $compile,
3.       $rootScope,
4.       compiledResult;
5.
6.   beforeEach(module('RentalApp'));
7.   beforeEach(module('myTemplates'));
8.   beforeEach(inject(function(_$compile_, _$rootScope_){
9.     $compile = _$compile_;
10.    $rootScope = _$rootScope_;
11.    var checkIn = moment("02/05/2015", 'DD/MM/YYYY');
12.    var checkOut = moment("9/05/2015", 'DD/MM/YYYY');
13.    compiledResult = $compile("<count-booking-price night-price='5' check-
14.      in='"+checkIn+"' check-out='"+checkOut+"'></count-booking-price>")($rootScope);
15.    $rootScope.$digest();
16.  }));
17.  it('Should produce span element', function() {
18.    expect(compiledResult.find("span").length).toEqual(1);
19.  });
20.  it('Should calculate property night count and price', function(){
21.    expect(compiledResult.find("span").html()).toEqual("7 nights - 35 EUR");
22.  });
23. });
```

Please note that there is loaded module 'myTemplates' (line 7). It contains all html partials of the project. The module generated by Karma plugin karma-ng-html2js-preprocessor. It is



required to use it so that AngularJS directives would not try to fetch needed partials from the server (and fail to do so), but from AngularJS \$templateCache storage.

In order to compile given directive it is necessary to pass directive HTML code to special AngularJS service called \$compile (line 13). Furthermore there is also called \$rootScope.\$digest() in order to simulate scope life cycle and therefore finalize the directive generation (line 14). Then it is possible to test DOM elements that are generated by the directive (lines 17-22).

Jasmine together with Karma are extremely powerful tools that can test a lot of aspects of the AngularJS-based applications thanks to JavaScript natural flexibility, AngularJS framework architecture principles and dependency injection system.

## 4.4 Testing Angular controllers

AngularJS controllers play undoubtedly crucial part in the application, hence they are also needed to be tested. In this chapter is shown how to test controllers that interact with server REST endpoints.

```
1. describe('ShowPropertyCtrl controller unit testing',function(){
2.     var API_URL, $httpBackend, scope, $controller;
3.     var stateParams = {propertyId:1};
4.
5.     beforeEach(function(){
6.         module('RentalApp');
7.     });
8.     beforeEach(module(function ($urlRouterProvider) {
9.         $urlRouterProvider.otherwise(function(){return false;});
10.    }));
11.    beforeEach(inject(function(_$controller_, $rootScope, _$httpBackend_, _API_URL_){
12.        $controller = _$controller_;
13.        $httpBackend = _$httpBackend_;
14.        API_URL = _API_URL_;
15.        scope = $rootScope.$new();
16.
17.        $controller('ShowPropertyCtrl',{scope:scope,$stateParams:stateParams});
18.        $httpBackend.when('GET',API_URL+'properties/reviews/1').respond([{id:1}]);
19.        $httpBackend.when('GET',API_URL+'bookings/canSendReviews/1').respond({can:true});
20.        $httpBackend.when('GET',API_URL+'properties/1').respond({id:1,description:"descripti
21.        ons",rules:"rules",imagePaths:[{path:"img1.jpg"},{path:"img2.jpg"},{path:"img3.jpg"}
22.        ]});
23.        $httpBackend.when('GET',API_URL+'bookings/unavailableDates/1').respond([{date:'27/06
24.        /15'}]);
25.        $httpBackend.flush();
26.    }));
27.
28.    describe('$scope.mainImgUrl initial value',function(){
29.        it('should set first image as main image',function(){
```

```

26.         expect(scope.mainImgUrl).toEqual("img1.jpg");
27.     });
28. });
29. describe('$scope.nextImg', function(){
30.     it('should set second image as main image', function(){
31.         scope.nextImg();
32.         expect(scope.mainImgUrl).toEqual("img2.jpg");
33.     });
34. });
35. describe('$scope.prevImg', function(){
36.     it('should set third image as main image', function(){
37.         scope.prevImg();
38.         expect(scope.mainImgUrl).toEqual("img3.jpg");
39.     });
40. });
41.
42. afterEach(function() {
43.     $httpBackend.verifyNoOutstandingExpectation();
44.     $httpBackend.verifyNoOutstandingRequest();
45.     $httpBackend.resetExpectations();
46. });
47. });

```

In order to execute the controller explicitly, there should be used `$controller` service (line 16). The test suite holds controller scope that is used to access controller functionality (line 2 and 16). Notice that during tests controller is not bound to any state, ui-router routing module do not participate in its creation and execution. Therefore, it is needed to mock `$stateParams` object and pass it to controller so that controller would normally work (line 16).

When controller bootstraps it fetches required data from the server. In order to simulate HTTP server-side implementation there should be used special service `$httpBackend` which is provided by angular-mocks module. In the `$httpBackend` service is specified which URL it fakes and what data should be returned to the controller (lines 17-21).

Moreover, there occurs a problem if the project uses AngularJS ui-router. Namely, because of following configuration in the root application configuration code block:

```
1. $urlRouterProvider.otherwise("/");
```

It specifies default application state where to redirect users if requested state cannot be found. This configuration is applied during tests forcing application to make GET call asking for HTML partial of default state. One of the possible solutions is to eliminate this configuration in the `beforeEach` block (lines 8-10) [19].

Lastly, after each test was run `$httpBackend` service should be reset (lines 42-46). Other parts of the test suite are pretty similar to the previous examples.

## 5. Conclusion

The main goal of the thesis was to explore the architecture of Single Page Application and nature of AngularJS-based applications. There have been reviewed a lot of aspects of AngularJS application such as project modules organization, AngularJS application architecture, integration with third-party JavaScript libraries and implementation of authentication. In the course of the work, there were also overviewed how suitable is AngularJS applications for unit testing with Jasmine testing framework. Lastly, there were reviewed different JavaScript development tools.

In order to achieve abovementioned objectives there have been developed sample project – Property rental system.

AngularJS is extremely powerful and flexible framework that provides a lot of freedom to developer. Developer is free to choose which application front end architecture and structure to build. Right architectural choices ensure further scalability and maintainability of the application.

In AngularJS application authentication and authorization may be implemented in a various ways. One of them is using traditional session and cookies in conjunction with HTML5 local storage and AngularJS components.

What is more, AngularJS framework provides all the necessary means to integrate third-party JavaScript libraries into the application. This AngularJS feature was utilized in the project to the fullest extent.

When it comes to testing AngularJS application, it turned out that AngularJS is test-friendly framework – the most application modules can be tested: from filters that simply modify given input to directives that compile HTML code.

There are constant growth of JavaScript-based development tools that allow developer to focus mainly on application itself rather than on the performing routine and monotonous tasks such as running tests, downloading project dependencies, etc. In the scope of the thesis, there were used Bower, Grunt and Karma. They did prove to be extremely useful and customizable.

To sum up, all the thesis objectives were successfully achieved. On the other hand, some of them could have been examined more precisely, for instance, AngularJS application authentication concerns. One of the possible areas of further research could be usage of various JavaScript development tools in AngularJS project.

## Kokkuvõtte

Bakalaureusetöö põhiseks ülesandeks oli Single Page Application arhitektuuriga ja Angularil põhinevate veebirakenduste omaduste ja disaini uurimine. Lõputöö käigus uuriti Angulari raamistikku kasutavate rakenduste erinevaid aspekte: projekti osade struktureerimine, rakenduse arhitektuuri, integratsioon teiste Javascript'i teekidega ja autentimise realiseerimine Angulari rakenduses. Töö käigus uuriti kui hästi Angulari rakendus sobib unit testimiseks, kasutades Jasmine testimise raamistikku. Käsitleti ka mitmesuguseid Javascript'i arendamise töövahendeid.

Uurimise käigus ehitatud rakenduse valdkonnaks oli kinnisvara rentimise rakendus.

AngularJS on väga võimalusterikas ja paindlik raamistik mis pakub arendajale päris palju vabadust. Arendajal on suhteliselt suur vabadus valida rakenduse kliendi poole arhitektuuri ja projekti struktuuri ülesehitust. Õiged valikud tagavad, et rakendus on skaleeritav ja hooldatav ja antud töö üheks eesmärgiks ja tulemuks oli nende valikute väljaselgitamine.

AngularJS rakenduses on võimalik erinevalt rakendada kasutajatunnustust. Üks viis on kasutada traditsioonilist sessiooni koos HTML5 local storage'ga ja AngularJS'i komponentitega.

AngularJS võimaldab integreerida rakendusse teisi Javascripti teeke, ka seda võimalust töö käigus uuriti ja kasutati.

Uuriti AngularJS'i raamistiku poolt pakutavat tuge testimisele. Angular on testimise-sõbralik süsteem - peaaegu kõik rakenduse elemendid on testitavad: alustades filtritest mis lihtsalt muutuvad antud sisendit ja lõpetades directive'tega mis toodavad HTML koodi.

Tänapäeval arenevad Javascripti arendamise tööriistad kiiresti. Nad vabastavad arendajaid rutiinsetest ja monotoonsetest tööst nagu testide käimapanek, teekide allalaadimine jne. Projektis kasutati töövahendeid Bower, Grunt ja Karma. Nad on kahtlemata päris kasulikud ja vajadustele kohaldatavad.

Arvan et tööle püstitatud eesmärgid on saavutatud. Selgusid ka teemad millega võiks tegeleda edasi, näiteks AngularJS rakenduse kasutajatuvastamise protsess. Edasiseks uurimisvaldkonnaks võiks olla ka Angulari koostöö erinevate Javascripti arendusvahenditega.

## References

1. Single-page application. – Wikipedia. [WWW] [http://en.wikipedia.org/wiki/Single-page\\_application](http://en.wikipedia.org/wiki/Single-page_application) (08.04.2015)
2. François Petitit (2014) The new Web application architectures and their impacts for enterprises – Part 1 – [WWW] <http://blog.octo.com/en/new-web-application-architectures-and-impacts-for-enterprises-1/> (15.04.2015)
3. AngularJS. – Wikipedia. [WWW] <https://en.wikipedia.org/wiki/AngularJS> (09.04.2015)
4. Market share yearly trends for JavaScript libraries for websites. – W3Techs. [WWW] [http://w3techs.com/technologies/history\\_overview/javascript\\_library/ms/y](http://w3techs.com/technologies/history_overview/javascript_library/ms/y) (14.05.2015)
5. Google Trends – Web search interest: angular js, ember js, backbone js, knockout js – Worldwide, Jan 2010 – May 2015 – Google Trends [WWW] <http://www.google.com/trends/explore?q=angular+js#q=angular%20js%2C%20Ext%20JS%2C%20ember%20js%2C%20backbone%20js%2C%20knockout%20js&date=1%2F2010%2064m&cmpt=q&tz=> (17.05.2015)
6. Dan Siepen (2014) – Top 15 sites built with AngularJS. – [WWW] <http://coderfactory.co/posts/top-sites-built-with-angularjs> (14.05.2015)
7. Node.JS. – Wikipedia. [WWW] <https://en.wikipedia.org/wiki/Node.js> (14.04.2015)
8. Adnan Kukic. (2014) AngularJS Best Practices: Directory Structure. – [WWW] <https://scotch.io/tutorials/angularjs-best-practices-directory-structure> (10.04.2015)
9. Igor Minar. (2012) MVC vs MVVM vs MVP. What a controversial topic that many developers can spend hours and hours debating and arguing about. – [WWW] <https://plus.google.com/+AngularJS/posts/aZNVhj355G2> (17.05.2015)

10. Marco Rinck (2014) How to structure large angularJS applications – [WWW] <http://entwicklertagebuch.com/blog/2013/10/how-to-structure-large-angularjs-applications/> (16.04.2015)
11. AngularJS: Developer guide: Dependency injection – [WWW] <https://docs.angularjs.org/guide/di> (19.04.2015)
12. AngularJS: Developer guide: Scopes – [WWW] <https://docs.angularjs.org/guide/scope> (18.04.2015)
13. AngularJS: Tutorial: 2 – Angular templates – [WWW] [https://docs.angularjs.org/tutorial/step\\_02](https://docs.angularjs.org/tutorial/step_02) (20.04.2015)
14. AngularJS: Developer guide: Data binding – [WWW] <https://docs.angularjs.org/guide/databinding> (18.04.2015)
15. AngularJS: Developer guide: Directives – [WWW] <https://docs.angularjs.org/guide/directive> (19.04.2015)
16. Dave Syer (2015) The Login Page: Angular JS and Spring Security Part II – [WWW] <https://spring.io/blog/2015/01/12/the-login-page-angular-js-and-spring-security-part-ii> (21.04.2015)
17. Christopher Henkel (2014) Web Development Using Spring and AngularJS - Tutorial 12 – [WWW] <https://www.youtube.com/watch?v=fx7hoza7wIA> (21.04.2015)
18. AngularJS: Developer guide: Unit testing – [WWW] <https://docs.angularjs.org/guide/unit-testing> (23.04.2015)
19. Karma test breaks after using ui-router. – [WWW] <https://github.com/angular-ui/ui-router/issues/212> (04.05.2015)



## Appendix 1. Bower dependencies

Dependency	Version	Description
jquery	2.1.3	Although AngularJS framework has its own implementation of some jQuery functions that is called jQLite, jQuery is still needed as dependency for other JavaScript libraries that are used in the project.
bootstrap	3.3.2	Bootstrap is extremely powerful framework for developing web applications, used mainly for user interface development.
lodash	3.4.0	JavaScript utility library.
moment	2.9.0	JavaScript library for date manipulations.
bootstrap-datepicker	1.4.0	Datetimepicker with Bootstrap styles.
angular	1.3.15	JavaScript framework that allows to create Single page applications.
angular-resource	1.3.14	AngularJS module that provides high-level abstraction for interactions with RESTful server endpoints.
angular-bootstrap	0.12.1	AngularJS directives for some Bootstrap elements.
angular-ui-router	0.2.13	Routing framework for AngularJS. Connects URLs to specific states that load required HTML partials, controllers and other data.
angular-google-maps	2.0.13	Google Maps AngularJS integration.
ngAutocomplete	1.0.0	Google Places Autocomplete AngularJS integration.
angular-bootstrap-	0.3.12	AngularJS datetimepicker with Bootstrap styles.

datetimepicker		
ng-file-upload	3.2.4	AngularJS directive for file uploading.
highcharts-ng	0.0.8	Highcharts library AngularJS integration.
angular-mocks	1.3.15	Angular module for testing AngularJS application.

## Appendix 2. Node.js dependencies

Dependency	Version	Description
grunt	0.4.5	JavaScript task runner. Automates a lot of aspects of JavaScript development.
grunt-contrib-concat	0.5.1	Plugin for Grunt. Used to concatenate multiple JavaScript files into one.
grunt-contrib-uglify	0.9.1	Plugin for Grunt. Used to minify JavaScript files.
grunt-contrib-copy	0.8.0	Plugin for Grunt. Used to copy files.
grunt-contrib-watch	0.6.1	Plugin for Grunt. Used to run predefined tasks.
grunt-contrib-clean	0.6.0	Plugin for Grunt. Used to delete files.
karma	0.12.31	Test runner for JavaScript.
jasmine-core	2.2.0	JavaScript testing framework.
karma-jasmine	0.3.5	Plugin for Karma in order to integrate Jasmine framework.
karma-ng-html2js-preprocessor	0.1.2	Plugin for Karma. Used to gather application HTML templates and put them into AngularJS templates.
karma-phantomjs-launcher	0.1.4	Plugin for Karma. Used to run PhantomJS – web browser without graphical user interface where all the tests are executed.