LAUREA
UNIVERSITY OF APPLIED SCIENCES
Together we are stronger

Development of a front-end application using AngularJS: 1UP Media company case

Dao, Vu

2016 Leppävaara

Laurea University of Applied Sciences
Leppävaara

Development of a front-end application using AngularJS: 1UP Media company case

Vu Dao
Degree Programme in BIT
Bachelor's Thesis
December, 2016

Laurea University of Applied Sciences
Leppävaara
Business Information Technology

Abstract

Vu Dao

Development of a front-end application using AngularJS: 1UP Media company case

Year 2016 Pages 48

In this era of a data driven economy, customer behaviour data can be decisive for a business's success as it allows optimisation to services and products. While big businesses can afford an investment on expensive system to gather customer data, small and medium businesses find it difficult. Realising a potential market space, 1UP Media Oy commissioned Lumi platform, a cloud based platform that is modularised, affordable and precise.

The thesis starts by introducing the company and the project background. Then the thesis product with its objectives and requirements is described. The thesis framework, waterfall model, is used for it's straight-forward in execution and management. All concepts that the application is based on are explained, such as cloud computing, client-server, front-end application, and application framework. Additionally, it explains the design, implementation of the application and analyse testing result.

The main objective of the product is an application that provides people counting data to the user as well as enables the user to manage resources and access to the data. Additionally, the application should be structured to allow extra data services which potentially would be developed later either by 1UP Media or third-party developers. To fulfil the requirements, AngularJS, a modern framework, was used for its Model-View-Controller architecture allows developer to separate an application business logic and behaviour logic from its presentation. As long as the application is divided into cohesive loose-coupled components which are carefully documented, new services can be added with minimal effort.

The thesis product was integrated into the platform completing a pilot version of it, which was deployed for the customers and further developed based on additional user feedback. All the desired features and attributes were successfully delivered to the user during the live testing process, including an extra component requested during the testing period and quickly developed by the project team. These results act as a proof for the success of the project and the practices used in the product development. The application is prepared for expanding and modifying as more users interact with it.

Web development, Application framework

Table of contents

1 Introduction

In recent years, web applications have seen enormous changes with new trends and best practices emerged. One of them is the tendency to move increasingly application logic and interaction to the client web browser (client), leading to the client-side application getting bigger, so is the need for libraries and frameworks to enable developers to build them fast and highly performant. Among modern front-end frameworks, AngularJS appears as a popular choice. Therefore, when 1UP Media OY, the company I am currently working for, started a project to create a cloud-based B2B service platform, AngularJS was chosen as the framework for the modern web application.

This paper covers the process of creating a front-end web application using AngularJS framework. It comprises of nine sections. In the first section I describe the basics of the project including the company background and project background. The thesis product and its objectives and requirements are introduced in the second section. The third section covers the theoretical framework used in the project. The fourth section, knowledge base, explains necessary terms and concepts for understanding the content of the paper. The fifth and six section discusses the design and the actual implementation of the application. After that, testing and maintenance process is covered before conclusion section wraps up the paper.

1.1 Company background

1Up Media OY is a private start-up company that operates in business to business software as a service area. The company has a small and young development team. The combination of a young energetic team and a fast-moving, hard-contested industry makes for a comfortable but challenging environment. Currently the company’s main project is Lumi, a cloud based platform to collect and provide critical business data to industrial customers.

1.2 Lumi platform project

Lumi is a platform that connects different hardware devices and software applications to collect, display data and provide valuable insight for a smart urban living environment. Lumi consists of several modules – a combination of hardware and software focus on collecting, and displaying a particular type of data. For the prototype project the focus is on the people counting module.

1.2.1 User

The platform targets small and medium businesses which interest in a multi-services platform including people counting. The reason is that 1UP Media is a small company and needs to build its customer base, also the company market research shows that there is insufficient supply in that area. Besides, starting with small and medium businesses in Helsinki region like libraries, museums and transport companies make it easier for us to handle infrastructure demand and improve our product with steady pace.

End users can view counting data and manage resources involved, such as hardware devices and staff members. Hardware supply companies and 1UP Media are administrator users, to supply the service to end users and monitor it.

1.2.2 System design

Hardware devices include mini computers, IP cameras and Internet modem, all of which will be handled by the hardware supply companies and the users. Software applications are divided into data collection software running in the mini computers, a back-end API running on a cloud server and a front-end application running on the user's browser.

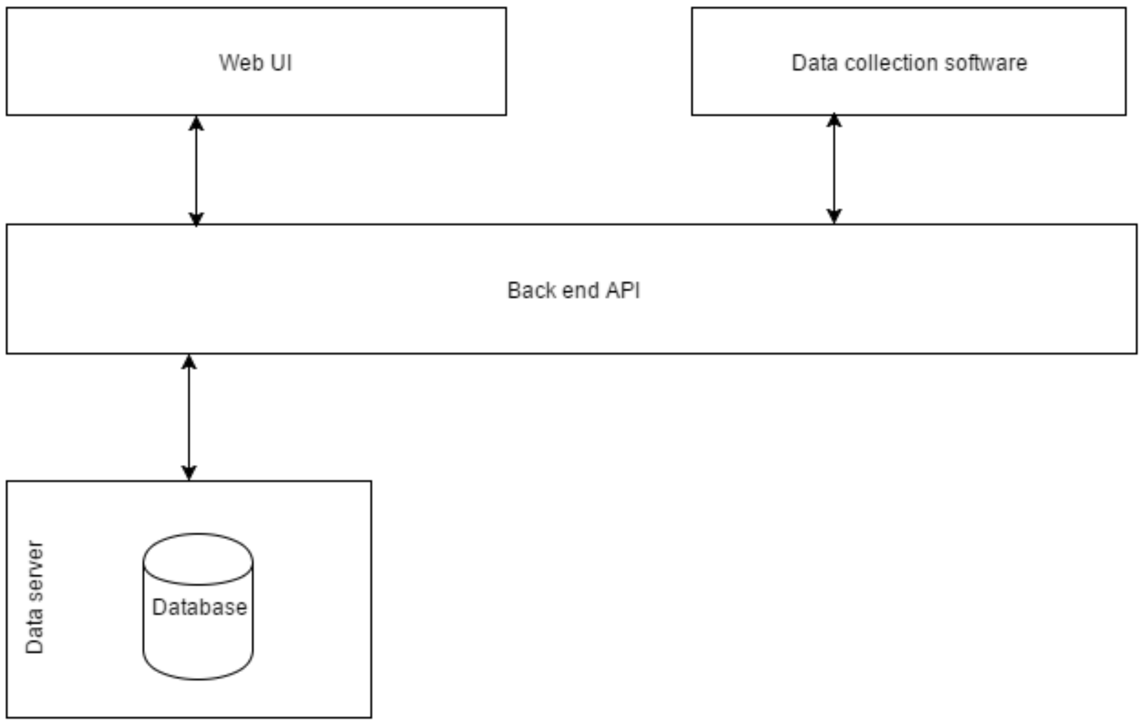


Figure 1-1: System model of Lumi people counting

Data collection software are responsible for collecting and sending counting data. There are 3 local client applications. The first application identifies, counts people, and tracks their directions. Another application running simultaneously oversees sending the collected data to the back-end API. Also, settings of a mini computer can be configured using a web-based application accessible in a local secured network.

The back-end API handles HTTP requests, executes additional business logic and interacts with the database service to add, edit, or get necessary data. Front-end application provides an interface for user to view data regarding their group, account, locations, devices, and people counting data. It also provides the user an interface to add and edit resources.

2 Thesis product

My responsibility in the project is to develop the front-end application for Lumi people counting platform. The main task is to design, develop all functionalities required for the application to interact with users and the back-end API, build, deploy the application, as well as create test cases and document the architecture and features.

2.1 Objectives

For functional requirements, the application includes the following features:

- Authentication: User must log in to access services provided by the application. User authentication is mandatory when making HTTP requests to protected path of the back-end API. Also, authentication should be stored in the browser's window for convenience.
- People counting chart: Authenticated user should be able to view people counting data collected by devices under their control. Those data should be presented in readable format such as chart and can be filtered by several metrics such as time or location, per user's needs.
- Control panel: User should be able to view, add and edit resources such as companies, locations, devices, and users. Resources should only be available within companies and for their parent companies. Resources should be updated and show to user immediately after new information is submitted and validated.
- Bug report: User should be able to submit bug report using a bug report form.

These features should be available for a pilot user based in Helsinki

2.2 Requirements

Per Sommerville et al., functional and performant, maintainable, dependable, and usable are desirable attributes in good software (Sommerville 2011). However, this paper will focus on extensibility, an important quality for a growing project with changing requirements like Lumi.

Extensibility is the ability of a system that allows new functionality to be added with minimal or no effects on its internal structure and data flow. There are three most important qualities of an extensible software system: modifiability, maintainability, and scalability.

- Modifiability as defined by Bass et al. is determined by how functionality is divided architecturally and by how coding techniques are applied within the code. A system is modifiable when a change involves the least number of changes to the least number of possible elements (Bass et al. 2003). According to Johansson, this means striving towards high cohesion and low coupling within the code.
- Maintainability as defined by Sommerville is to design a system to allow for the addition of new requirements without risk of adding new errors (Sommerville 2011). Per Johansson, this also requires the code to have high cohesion and low coupling.
- Scalability as defined by Bondi, is “the ability of a system to expand in a chosen dimension without major modifications to its architecture” (Bondi 2000). A scalable system can handle higher data load with minimal effect on performance. (Johansson & Löfgren 2009)

As seen above, the definitions of modifiability and maintainability imply the importance of the following four software quality attributes: modularization, component based architecture, coupling, and cohesion.

2.2.1 Modularization

Modularization is a technique that emphasizes separating a software system into multiple distinct and independent modules, which can independently execute one or many tasks. These modules may work as basic constructs for the entire software. Modules should be designed with separated concerns so that they can be developed and/or compiled separately and independently.

Advantages of modularization include improved maintainability, functionality based software modules, desired level of abstraction, high cohesion, reusability, and enhanced security (Tutorials Point 2016)

2.2.2 Component based architecture

Component based architecture is a software architecture where the application is separated into many objects called components. A component is a software object which handles a certain functionality or a set of functionalities. It should be created to interacts with other components and to be reused in the application. Components are loosely coupled and independent unit whose implementation are isolated from their interface, which means one implementation can be replaced without affecting other parts of the system. Component based architecture is an important software development approach since software are becoming larger and more complex, thus creating demand for high reusability. (Sommerville 2011)

2.2.3 Coupling

Coupling is how much one component implementation depends on that of another component; in simple words, how much it knows about the execution of the other component. Usually, coupling is in contrast of cohesion, as a software component should aim to achieve loose coupling and high cohesion. The act of making a tightly coupled component looser is called decoupling.

Loose coupling is a method of reducing the dependency between the components in a system while tight coupling is where components depend on one another to the extent that it is obligated when changing one component to change other components depending on it.

A good example of tight coupling is iPods as they are tightly coupled because their battery is soldered fixed to the device that once the battery dies, replacing new battery is very expensive. A loosely coupled player would allow the battery to be changed effortlessly.

When the components are loosely coupled, developer can create separate implementations for different problem areas. With separate implementation, a component can be changed as much as desired without breaking some bigger process that is using it. Furthermore, when components can be independently changed, they can be developed independently and even deployed separately. (Skrchevski 2015)

2.2.4 Cohesion

Cohesion in software engineering is the degree to which the elements of a certain module, class or component connected to each other to perform the task required of their container. A module with high cohesion has its functions and properties strongly related functionally.

Therefore, it reduces the complexity of the module as well as increases its maintainability and reusability.

There are several types of cohesion commonly found and here is the description of some of them:

- Coincidental cohesion: This type indicates low cohesion. Coincidental cohesion is when parts of a module are grouped without a clear meaningful relationship between them. An example of this in Angular could be a Utility service where functions and properties are completely independent of each other.
- Sequential cohesion: Sequential cohesion is when parts of a module are grouped because the output from one part is the input to another part. The source code normally can be followed from top to bottom which shows good level of coupling and maintainability.
- Functional cohesion: This type indicates high level of cohesion. A module achieves functional cohesion when its elements are grouped because they all supplement a single comprehensive task of the module (Skrchevski 2015)

3 Theoretical framework

This section describes the theoretical frameworks applied in this paper. It serves as a guideline to approach the goals and objectives of the project so that a solution can be developed. The process of creating Lumi front-end application includes developing a software product. This creates the need for a theoretical framework focusing on the development process of a software, thus validating the choice of using waterfall model.

The waterfall model is an example of a plan-driven process where plan and schedule of the process activities are created before implemented. The fundamental development activities of the waterfall model are reflected in its main stages:

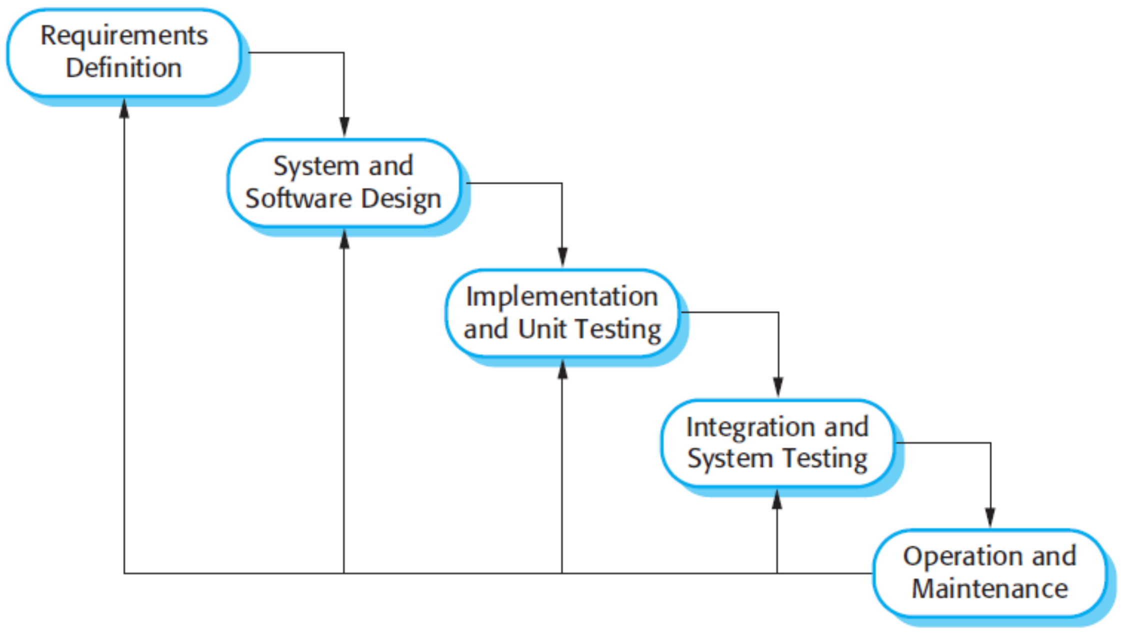

Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

Figure 3-1: Main stages of the waterfall model

- Requirements analysis and definition: Consultation with system users helps establishing the system’s services, constraints, and goals. They are then defined in detail and serve as a system specification.
- System and software design: The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.
- Implementation: During this stage, the software components are implemented based on the design.
- Testing: Unit testing is used to verify that each unit follows its specification. Next, the individual modules and components are integrated and tested as a complete software to ensure that the requirements have been fulfilled. After testing, the software system is delivered to the customer.
- Operation and maintenance: Usually, this is the longest life cycle phase. The operation process is where the system is installed and put into practical use. During maintenance, new errors are discovered and corrected, enhancing the implementation of system units, and improving the system’s services.

In principle, the result of each phase should be approved in one or more documents. The following phase should not start until the previous phase has finished. In practice, as the development follow its phases and the team understand more about the product, changes are rather inevitable. For example, during design, problems with requirements are identified or during coding, design problems are found. Feedback from one phase to another happens frequently, so the software process is not a simple linear model. Changes then must be reflected in documents produced in each phase.

Main benefits of the waterfall model are consistency and familiarity. Many other engineering process and projects follow waterfall model or a variant of it. Since it is easier to use a common management model for the whole project, software processes based on the waterfall model are still commonly used. Also, documentation is produced at each phase making the process visible so managers can monitor progress against the development plan. (Sommerville 2011)

4 Knowledge base

In this section, all concepts, and theories the application is based on are explained. These include overall system model like cloud computing and Software as a Service, client-server model, or high-level architectures like front-end driven application and application framework.

4.1 Cloud computing & Software as a Service

Cloud computing is a model to deliver computing resources conveniently on-demand of the customer. The resources can be continuously developed and released with minimal service provider interaction or management effort.

Software as a Service (SaaS) is a service model for cloud computing where the applications are accessible from various client devices. The consumer is not in charge of managing or controlling the underlying cloud infrastructure including but not limited to network, servers, operating systems, and storage (Mell & Grance 2011). SaaS model has many benefits such as reducing cost and time of prototyping, manageable scaling in spending and resource usage and being more accessible.

4.2 Client-server model

Client – server is a communication method between two remote applications. The client-side application can be called a front-end application which communicates by making requests to the server-side application. The server-side application can use a back-end API to handle its requests. Figure demonstrates how this model works

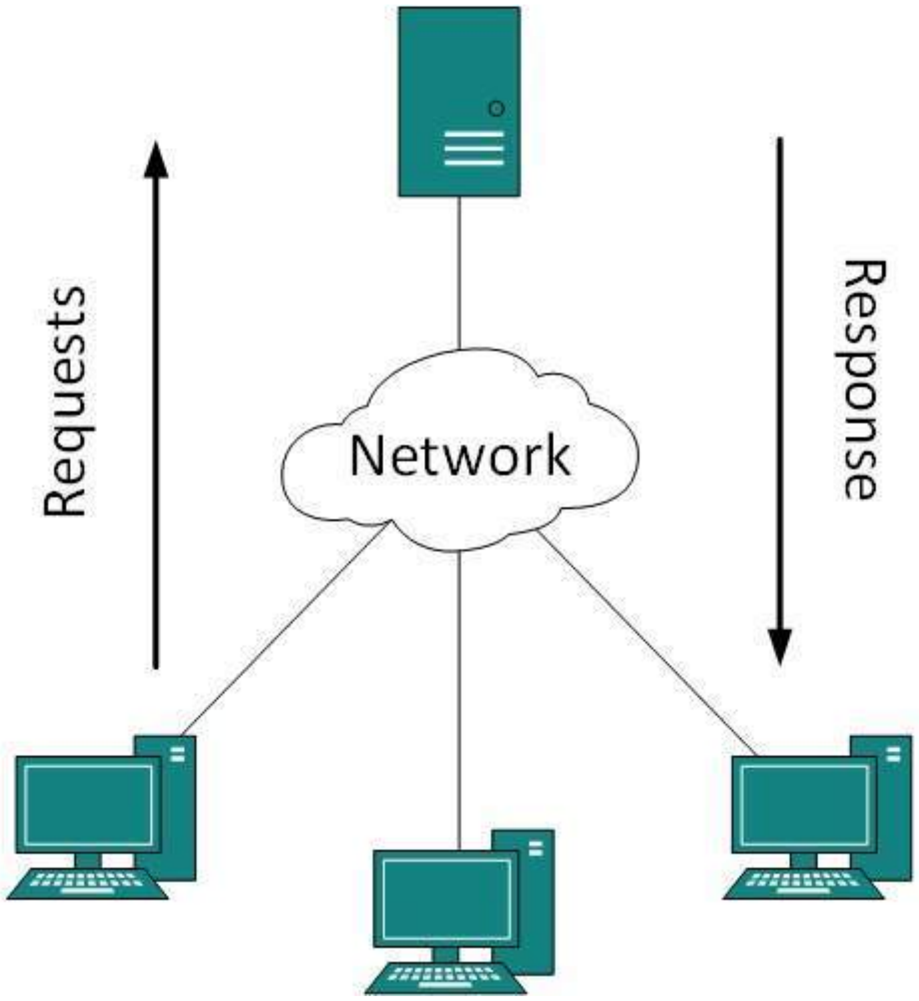


Figure 4-1: Client-server model

The front-end application usually provides the view and interface to the user. It controls what the user can see and how they can interact with the system. The back-end API mainly handles calculations, business logic, database interactions, and performance. (Jewett 2015)

4.3 Front-end driven application

In the past, websites were just static documents created with HTML. However, as the demand for reliable ways to create dynamic content and complex UI increases, application logic in website gets increasingly complex, creating the concept “front-end driven application”. A front-end driven application is understood to have a “thick client” architecture pattern which typically provides rich functionality independent of the back-end of the program (Halpern 2016). Front-end Driven Applications run on the client first from static files such as HTML, CSS, JavaScript and rely on HTTP requests to connect to independent, third party back-end services such as data storage, file storage, search, user management, emailing, and payments that can’t occur on the client alone (Lindley 2014).

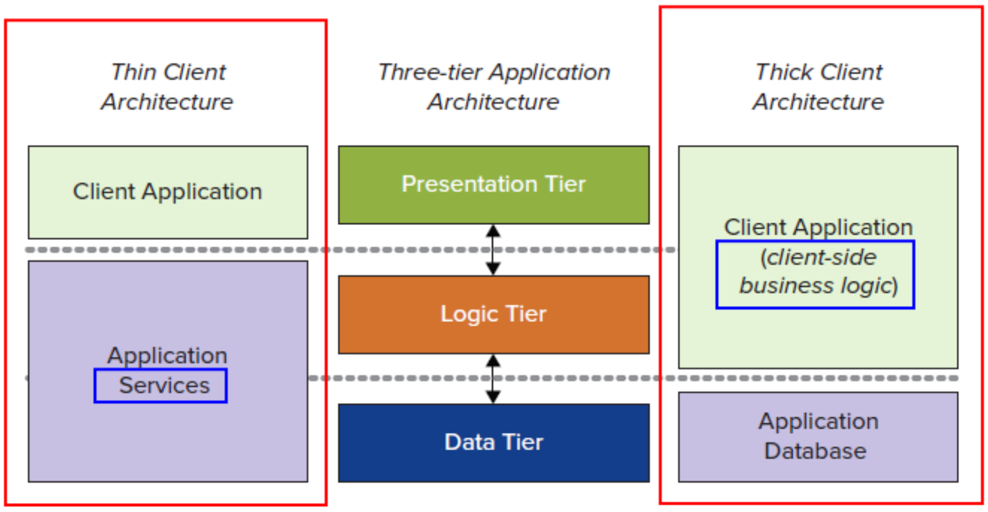


Figure 4-2: Compare thin client and thick client architecture

The primary difference between front-end driven application and traditional web app is that in a traditional web app, the server is accountable for fetching data and compiling it into HTML which is served as the view to users, while in a front-end driven application the browser is responsible for doing so (Staticapps.org 2016).

4.4 Application framework

A framework can be defined as an abstract design comprised of framework objects. It acts as "a skeleton, or scaffolding, that determines how framework objects relate to each other". A framework contains abstract and concrete class implementations that are reusable. The application classes use or inherit the implementation of framework objects and architecture, then customize them with detail implementations. In this way, the application becomes "extensions of the application framework".

Using application frameworks benefits the application in many aspects. Design and code reuse are the primary technical advantages. Additionally, the framework localised key design and implementation decisions, making the system easier to maintain. The primary business advantages of application frameworks are higher developer productivity and shorter development and deployment time. Also, since application frameworks are highly matured, applications tend to be less buggy and tend to have a similar structure, as they share the same framework and therefore has the same architecture and similar implementations. (Riehle 2000)

5 Design

In the design phase of the development process, an abstract representation of how the application works is created. It starts with an overall architecture of the system with all components and their interconnection. Then the detail specification of each component is described. This includes all objects required in the component and how they combined to acquire and transform data resources into corresponding output.

5.1 System modelling

System modelling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. A model serves as a blueprint of a system, ensures that the business functionality is completed and correct, user's need is fulfilled, and program design meets software quality standards before the implementation, therefore reducing changes later in development process. In general, a graphical notation such as UML is selected as a method of representing the system. The UML defines thirteen types of diagrams to support many different types of system model, divided into three categories:

- Structure Diagrams represent static application structure including the Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, and Deployment Diagram.
- Behaviour Diagrams represent general types of behaviour including the Use Case Diagram (used by some methodologies during requirements gathering); Activity Diagram, and State Machine Diagram.
- Interaction Diagrams represent different aspects of interactions including the Sequence Diagram, Communication Diagram, Timing Diagram, and Interaction Overview Diagram. (Object Management Group 2005)

Use case modelling is commonly used to support requirements analysis. A use case model describes different scenarios when user interacts with the system. Each use case represents a discrete task that involves external interaction with a system. Figure below shows the use case model from the Lumi web UI application that represents all main tasks of the system including login, logout, view chart data, add, view and edit services. (Sommerville 2011)

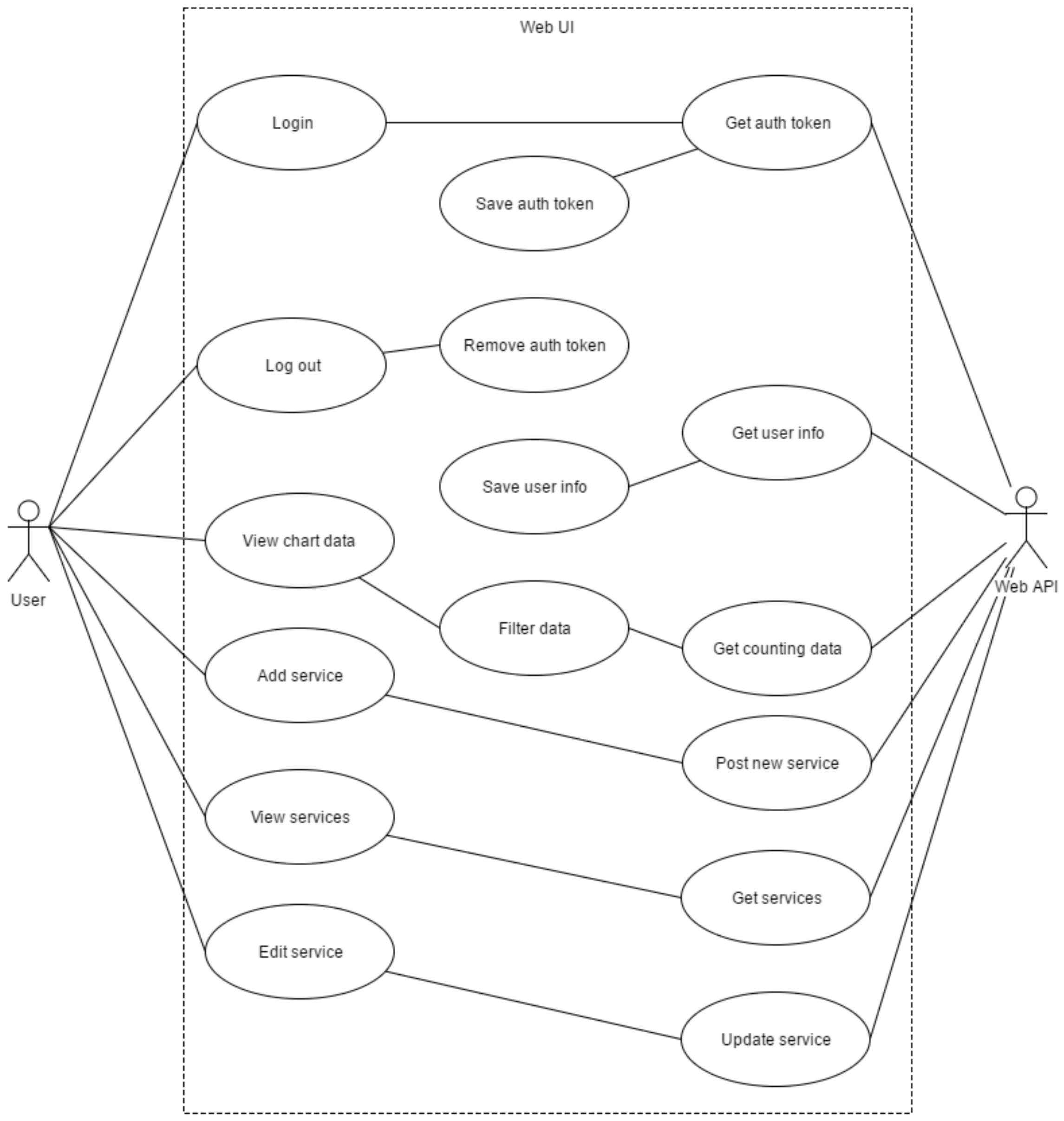


Figure 5-1: Different use cases of Lumi front-end application

There are two actors in the use case: the back-end API that handles data request and the user. From this model, Lumi front-end application should be divided into authentication, chart component, add a service component and view and edit service component. In addition, there should be a component to handle routing. The structure of the project folder can be derived from this use case model.

5.2 Routing modelling

For the navigation system, I use UI-Router, a third party angular component that allows for advanced navigation features of a single page application, which fit Lumi web UI application's potential for scaling. UI-Router organises the navigation system into states, which are setup before the application starts running and being rendered to view as the user navigating. A UI-Router state usually correlates to a view section with its UI and navigation which contains a feature of the application. Every state is an object with properties defining the functionality of the application when that state is active. The main properties of each states are:

- 'name': A name for the state, which can be used to refer to that state
- 'URL': The URL of the browser when the application should navigate to that state
- 'views': How the UI will look and behave
  - 'template': The html file defines how the view looks
  - 'controller': The AngularJS controller file controls behaviour of the view

A UI-Router state can have nested states and nested views. All states of the application and their nested states form a state tree. Nested states views are often rendered inside its parent's view, these are nested views. (AngularUI 2016)

UI-Router's approach of using a state tree splits the application into a hierarchy of functionality. The tree defines the application's functionality structure. The URL and views are presentation and control of the active state. This improves maintainability and scalability for Lumi web UI application. Below is the state tree of the application using UML state diagram

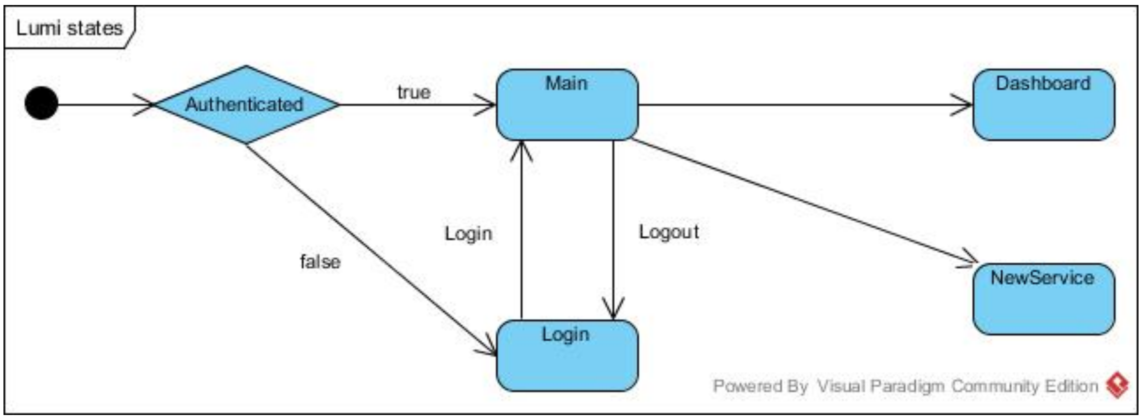


Figure 5-2: State diagram of Lumi front-end application

When user wants to transition to the Main state, the application checks their authentication token and only makes the transition if they are authenticated, otherwise the user is navigated to Login state. Because the user should not be able to navigate to any content pages without authentication, all content pages are child states of the main state which only accessible with user authentication. This ensures any future states added in similar fashion will be protected by the same authentication system.

5.3 Authentication modelling

As authentication in web application is a common feature, there are many examples and tutorials related to the issue. After researching and benchmarking, I decided Lumi front-end application should use a token based authentication. A token is a piece of data that inherently has no meaning, but when used alongside the correct tokenization system, help securing access to an application. Below is a diagram describing the token-based authentication system

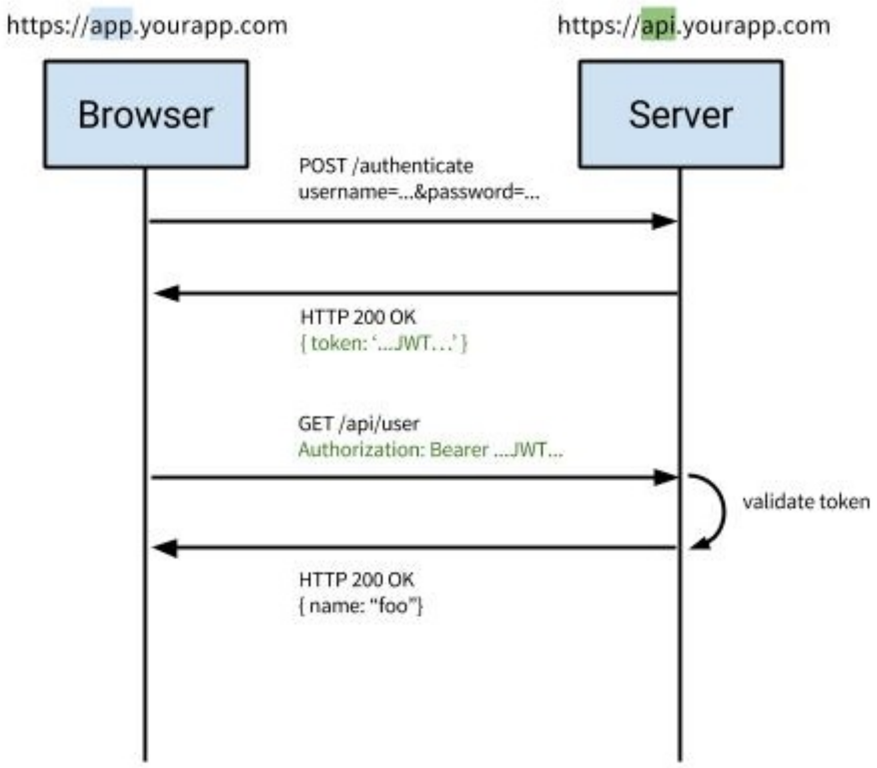


Figure 5-3: Sequence diagram describes token-based authentication

Token based authentication works by ensuring that each request to a server is accompanied by a signed token which the server verifies for authenticity and only then responds to the request. Token is retrieved after the user log in and saved in either cookies, local storage, or session storage of the browser. For Lumi, I chose to save the authentication token in cookies using AngularJS $cookies module. The token then is sent to the back-end API on each request and must either be removed when the user logs out or closes the browser window or expire after a certain period. (Pose 2014)

5.4 Chart modelling

Lumi people counting chart receives an array of raw counting data for a set of devices within a time range and filter them by a time unit. The process of creating a chart is described in the sequence diagram below

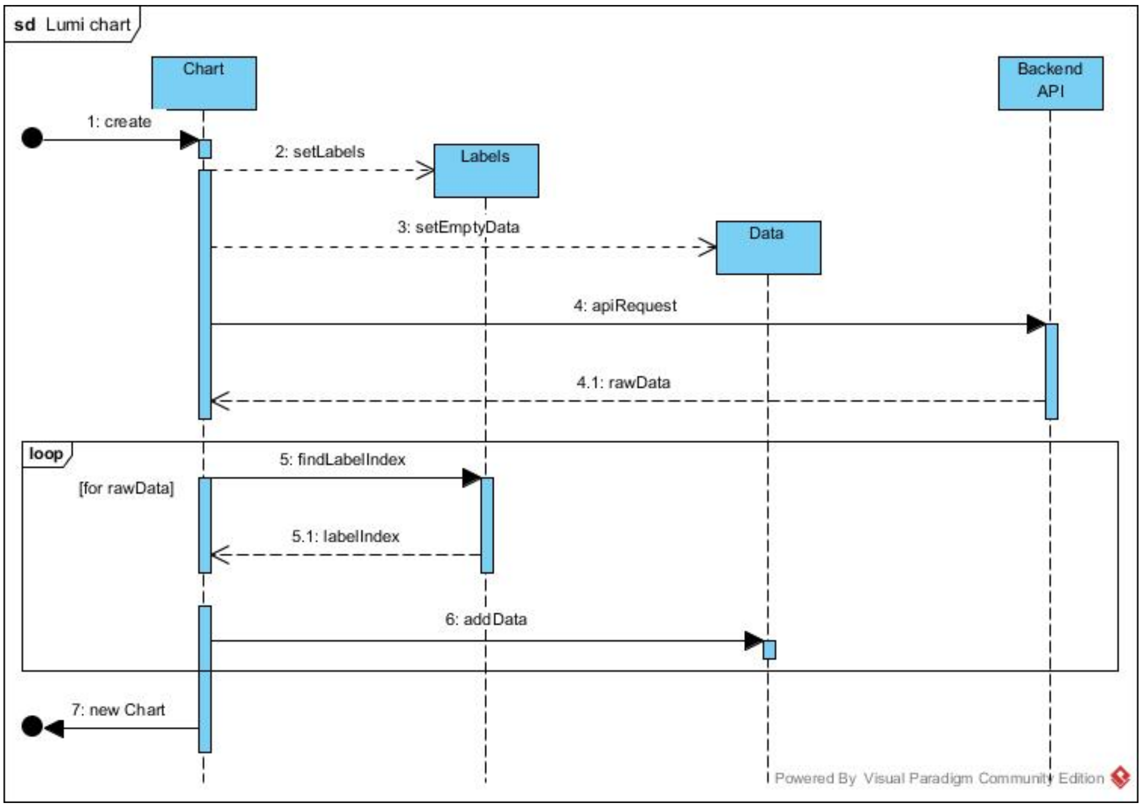


Figure 5-4: Sequence diagram describes the process to create a chart

When a chart is created, first it creates its labels – an array of time interval based on the time unit provided and its empty data – an array of the same length as labels, but empty. Then it makes a request to the back-end API for people counting data. The back-end API returns “rawData” – an array of people counting records. Each record is an object containing id of the device that recorded it, timestamp of when the device detected a person, and other information. The application then finds the label that record resides and adds the record to the data array. After filtering all items in “rawData” array, a new chart is created.

6 Implementation

The implementation phase is the duration where the actual product is created. This process makes use of the models created in the design phase by using optimal languages, libraries, and development tools to realise the design of every components in the software. Individual components are integrated into the completed application, built, and deployed.

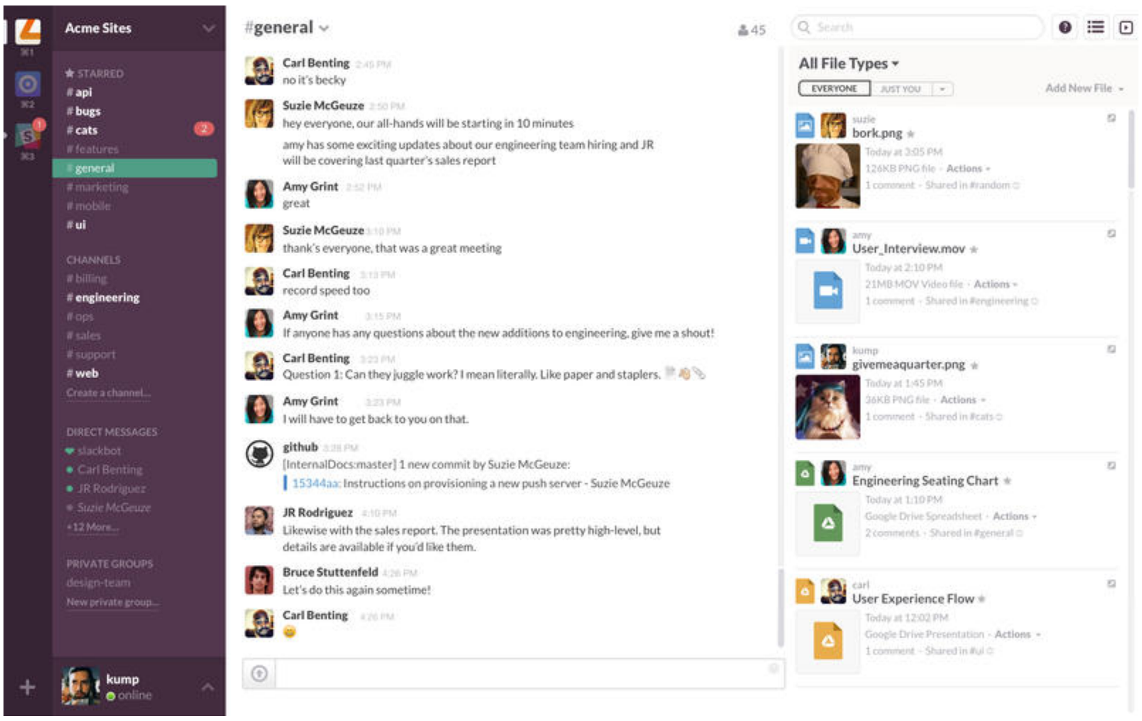
6.1 Development team

The development team consists of 2 back-end developers, a graphic designer and me as front-end developer. All developers have previous experiences working with common technologies, tools, and practices. This allows the team to collaborate with ease. Another advantage the team has is that all developers are familiar with JavaScript and NodeJS, so choosing those technologies for the project is a straight-forward decision.

When it comes to management, communication and progress monitoring are our main focuses. Effective communication plays vital role in the success of a project. It bridges gaps between client and the organization, among the team members as well as other stake holders in the project such as hardware suppliers. Monitoring Lumi web app project in general includes reviewing and tracking project progress. Changes from the original plan are identified and managed to keep the project within scope, on time, and within budget.

6.1.1 Communication

Slack is used for internal communication between the project team and the project manager. Slack is a messaging application for teams with helpful features for collaboration such as private channels and file-sharing. Slack provides a centralized place to communicate internally through instant messages and in chat rooms, which can reduce the time team members must spend on e-mail. Being able to use all features from mobile devices also increases members' availability.



Carl Benting 2:45 PM
no it's becky

Suzie McGeuze 2:50 PM
hey everyone, our all-hands will be starting in 10 minutes
amy has some exciting updates about our engineering team hiring and JR will be covering last quarter's sales report

Amy Grint 2:52 PM
great

Suzie McGeuze 3:10 PM
thank's everyone, that was a great meeting

Carl Benting 3:13 PM
record speed too

Amy Grint 3:15 PM
If anyone has any questions about the new additions to engineering, give me a shout!

Carl Benting 3:23 PM
Question 1: Can they juggle work? I mean literally. Like paper and staplers.

Amy Grint 3:23 PM
I will have to get back to you on that.

github 3:28 PM
[InternalDocs:master] 1 new commit by Suzie McGeuze:
15344aa: Instructions on provisioning a new push server - Suzie McGeuze

JR Rodriguez 4:10 PM
Likewise with the sales report. The presentation was pretty high-level, but details are available if you'd like them.

Bruce Stuttenfeld 4:26 PM
Let's do this again sometime!

Carl Benting 4:26 PM
😀

suzie
bork.png
Today at 3:05 PM
126KB PNG file · Actions
1 comment · Shared in #random

amy
User_Interview.mov
Today at 2:10 PM
21MB MOV Video file · Actions
1 comment · Shared in #engineering

kump
givemeaquarter.png
Today at 1:45 PM
36KB PNG file · Actions
1 comment · Shared in #cats

amy
Engineering Seating Chart
Today at 1:10 PM
Google Drive Spreadsheet · Actions
2 comments · Shared in #general

carl
User Experience Flow
Today at 12:02 PM
Google Drive Presentation · Actions
1 comment · Shared in #ui

Figure 6-1: User interface of Slack

Bitrix24 is a project management tool our team used to follow schedule and report their progress. Bitrix24 improves efficiency both personally and collaboratively by providing comprehensive organizational features. Tasks can be defined to team members, or delegated after being received. Project management features of the product are imbedded in the ‘groups’. Time spent working on any given project task can be reported, and tasks are integrated into the group calendar so managers can monitor the progress on a regular basis. (Bitrix24 2016)

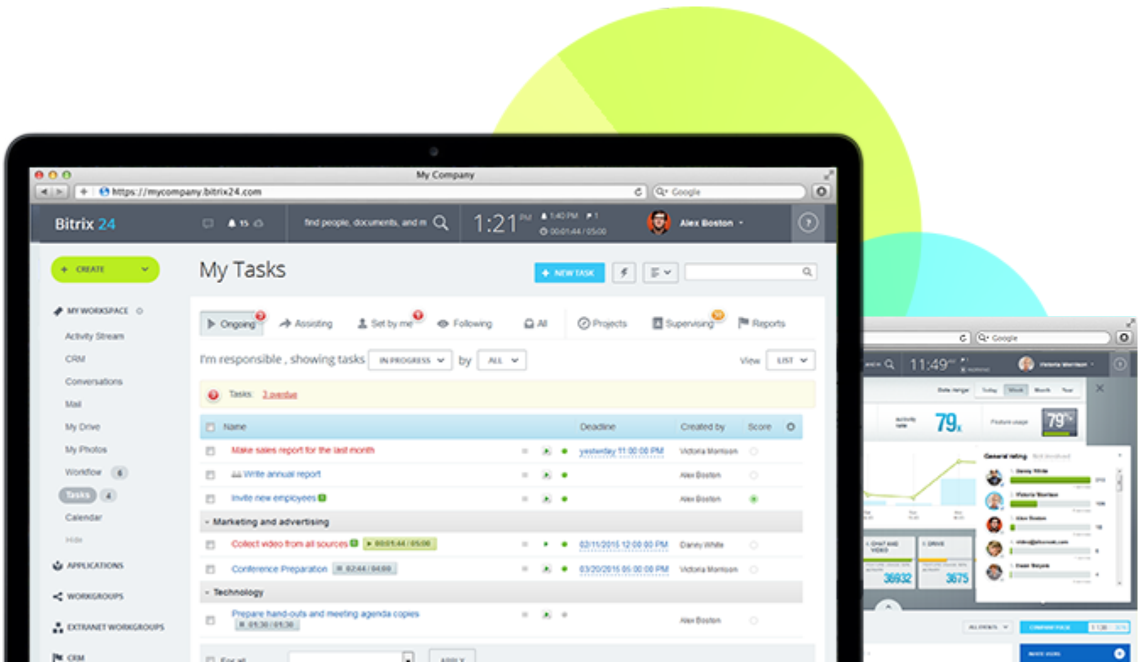


Figure 6-2: Bitrix24 tasks management interface

6.1.2 Progress monitoring

We use several methods of monitoring the progress to check whether tasks are completed per the original schedule and make iterations to the schedule accordingly.

- Activity Monitoring: All activities scheduled within some task can be monitored on day-to-day basis. When all activities in a task are completed, it is considered as complete.
- Status Reports: The reports contain status of activities and tasks completed within a given period, generally a week. Status can be marked as finished, pending or work-in-progress etc.
- Milestones Checklist: Lumi development process is divided into multiple phases where major tasks are performed. A milestone checklist of objectives and team members mark the end of these phases can report their status.

6.1.3 Risk Management

During risk management process, the team identify, analyse, and prepare solutions for possible risks in the project. A successful project normally includes evaluating potential problems in the plan and developing strategies to address them.

Because the project deadline is aggressive, the risk of requirement change or misinterpreting requirement is quite substantial. Among all features, those that are common for several existing applications which implementation can be reused such as user authentication and navigation pose smaller risks. Other features unique for Lumi web application which for the large part must be designed and built by the development team are more vulnerable to changes, therefore more time should be reserved.

There is also a risk of under-estimation of required time and resources. Although I have experiences with the technologies used in the project, I am not used to the scale and complexity of this application, so details have a potential to be left out.

6.2 Technologies

To implement all designs model of the components, many web technologies were used. These includes HTTP to get and post data; HTML, CSS, and JavaScript as standard language for web browsers; AngularJS and ChartJS as frameworks and libraries to help with development; and finally, Node.js to create a server for the application.

6.2.1 HTTP

The HTTP protocol is a request/response protocol for distributed information systems. A client establishes a connection to the server before sending a request in the form of a request method, URL, and protocol version, followed by a message containing request modifiers, client information, and possible body content. The server then sends a respond containing a status line, including the message's protocol version and a success or error code, followed by a message with server information, entity meta-information, and potential entity-body content. The set of common methods for HTTP are: GET, POST, PUT and DELETE. However, the most used methods are GET and POST, which are explained below

The GET method is used to retrieve the information identified by the requested URL. A GET request can include more information about the request in its header like path, query, data type, etc. The POST method is used to send an enclosed data to the origin server at the resource identified by the requested URL. POST is designed to allow a uniform method to

- Post a message to a bulletin board, newsgroup, mailing list, or similar group of articles
- Providing a block of data, such as submitted data from a form, to a data-handling process
- Extending a database through an append operation.

However, the server usually determined the actual function performed by the POST (Fielding 1999)

6.2.2 HTML

HTML is a mark-up language for describing web documents. HTML documents include a tree of elements and text. In the source, a start tag, such as "<body>", and an end tag, such as "</body>" indicates an element. Certain start tags and end tags can in certain cases be omitted and are implied by other tags. Attributes can be added to elements to specify their precise implementation. This mark-up text can be parsed by a browser into a DOM (Document Object Model) tree, which is a representation of the document which is stored in the memory. Each element in the DOM tree is represented by an object, and these objects can be controlled through their API using scripts embedded in the document. Scripts (typically in JavaScript) are small programs that can be embedded using the script element or using event handler content attributes. (Faulkner et al. 2016)

6.2.3 CSS

CSS (Cascading Style Sheets) is the language for describing the rendering of structured documents such as HTML and XML, including colours, layout, and fonts. A CSS document is a series of qualified rules consisting of usually style rules that apply CSS properties to HTML elements, specifying their presentation styles, and at-rules that define special processing rules or values for the CSS document. (Atkins & Sapin 2014)

6.2.4 JavaScript

JavaScript is a high-level programming language commonly used as a Web technology. JavaScript is part of the triad of fundamental technologies for web development: HTML to specify the content of web pages, CSS to specify the presentation of web pages, and JavaScript to specify the behaviour of web pages. With JavaScript being able to run on all modern browsers without the need for plug-ins or compilation, it is used in an overwhelming majority of modern websites, and all modern web browsers such as on desktops, game consoles, tablets, and smart phones. (Flanagan 2011)

6.2.5 AngularJS

AngularJS is a JavaScript MVC framework for making front-end web applications. MVC (Model – View – Controller) is an architectural design pattern that aims to improve application organization by enforcing the isolation of business data (Models) from user interfaces (Views), with a third component (Controllers) traditionally managing logic and user-input. By separating application logic from the user interface, MVC frameworks increase thee reusability of their data and logic for other interfaces in the application.

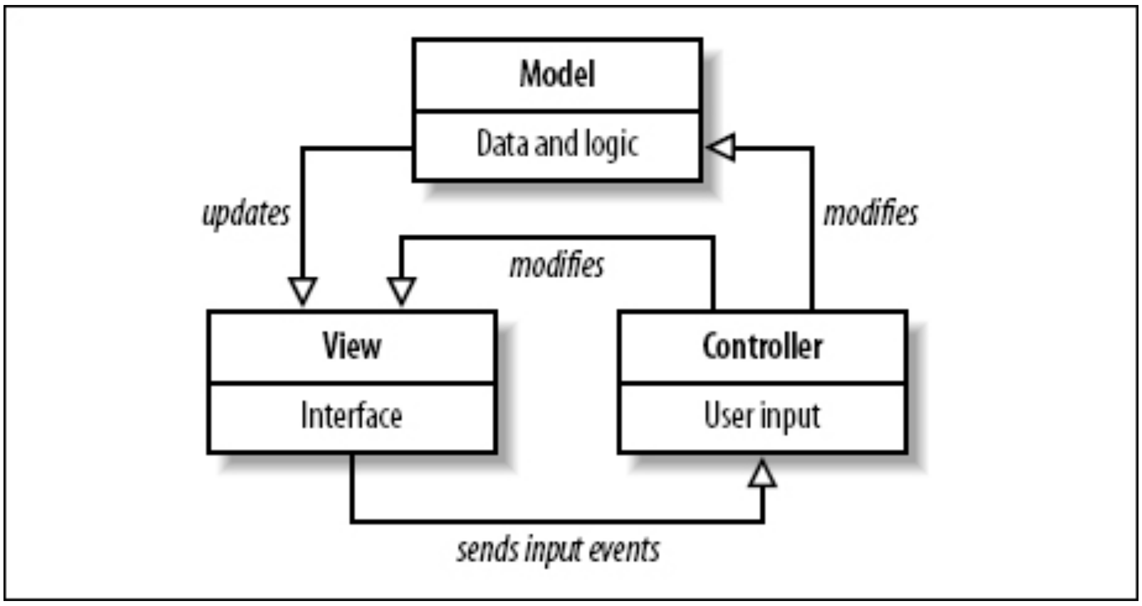


Figure 6-3: Model - View - Controller pattern

- Model: A Model manages data and logic of the application. An MVC application can affect its model using Controllers. When a model changed, it would inform its observers.
- View: A View is a visual representation of its Model that represented the current state of a Model. The Observer pattern allows the View to know whenever the Model was updated or modified. AngularJS applications contains HTML templates which can be loaded and transformed, and rendered as the view. AngularJS directives extend HTML's syntax to express application's components clearly and succinctly.
- Controller: Controllers role is handling user interaction (such as key-presses and mouse clicks), making decisions for the View. Controllers are responsible for two tasks: update the view when the model changes and update the model when the user manipulates the view.
- Directive: In general, directives are markers on a DOM element (such as an attribute, element name, comment, or CSS class) that attaches a behaviour, often defined in a controller, to that DOM element using AngularJS's HTML compiler. Directives can also be used to handle DOM transformation of the element and its children.
- Service: In AngularJS world, the services are singleton objects or functions that carry out specific tasks. It holds some business logic that can be shared across the application. A controller or directive can be injected with any services. AngularJS manages these service objects. (Osmani 2015)

According to (Lau 2013), Angular offers an application several benefits including:

- Using directives, HTML can include new elements and syntax (for example <tab></tab>, <calendar></calendar>, <dropdown></dropdown>) without the developers manually manipulating the DOM. All the application needs to do is to assign attributes to elements to implement new functionalities. DOM manipulation is delegated to directives, which in turn allows the application to only concern with updating the view with new data. The directives will handle the subsequent behaviour logic of the view.
- With Angular taking care of high level architecture such as MVC model, developer can focus on the core logic of the application. The view is defined using HTML, which is more concise. Angular data-binding takes care of adding data to the view so it does not need to be done manually. Since directives are separate from the application code, multiple teams can work on them simultaneously with minimal integration issues.
- In traditional web applications, the view modifies the DOM to present data and manipulates the DOM to add behaviour. Using Angular, DOM manipulation code is navigated into directives. Angular sees the view as just another HTML page with placeholders for data. This way of looking at the view help user interface designers tremendously. User interface designers can focus on the view without distractions from DOM manipulations and application logic. Development process is streamlined by making application purely about presenting business data into views.
- Angular is unit testing ready. Angular links dependencies from different components together by Dependency Injection (DI). Because all controllers depend on DI to pass it information, Angular application can carry out unit testing by injecting mock data into the controller and measuring the output and behaviour. In addition, fake server responses into controllers can be handled using an Angular mock HTTP.

For those reasons, AngularJS has gained significant popularity in the web developing community and is one of the most popular front-end frameworks, creating an active community where I can find solutions for issues I faced during the development.

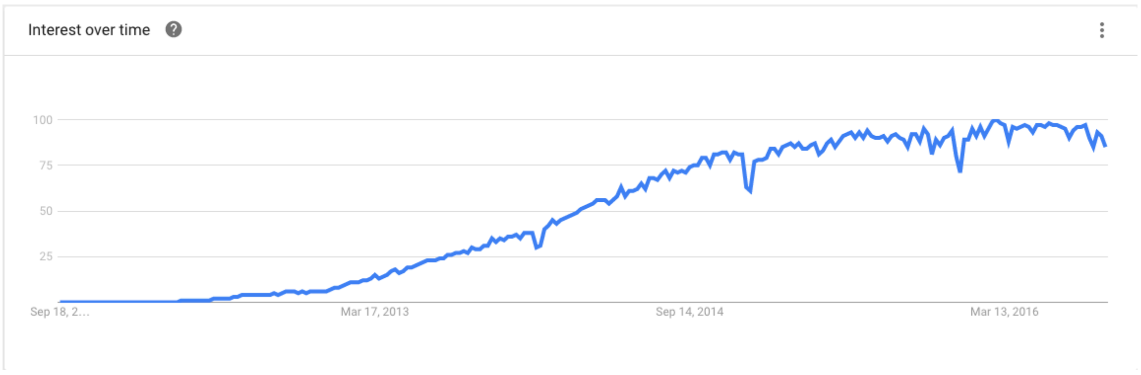


Figure 6-4: Interest in search term "AngularJS" over time

6.2.6 ChartJS

To help user gaining meaningful insight into the people counting data collected, I decided to display the data in charts. Since creating charts and graphs for web pages and front-end applications is a common requirement, many open source third party libraries are developed and actively maintained. After making sufficient research on JavaScript chart libraries, I chose Chart.js as my chart library for the following reasons

- Chart.js supports creating many types of chart including line chart, bar chart and pie chart.
- Chart.js uses HTML5 canvas which is highly performant across all modern browsers.
- Charts created are responsive and change size when the browser resizes.
- The API is very versatile, well documented, and highly customisable.
- Angular-chart is an Angular component using Chart.js making it easier for Angular developer to implement Chart.js in their app.

6.2.7 Node.js

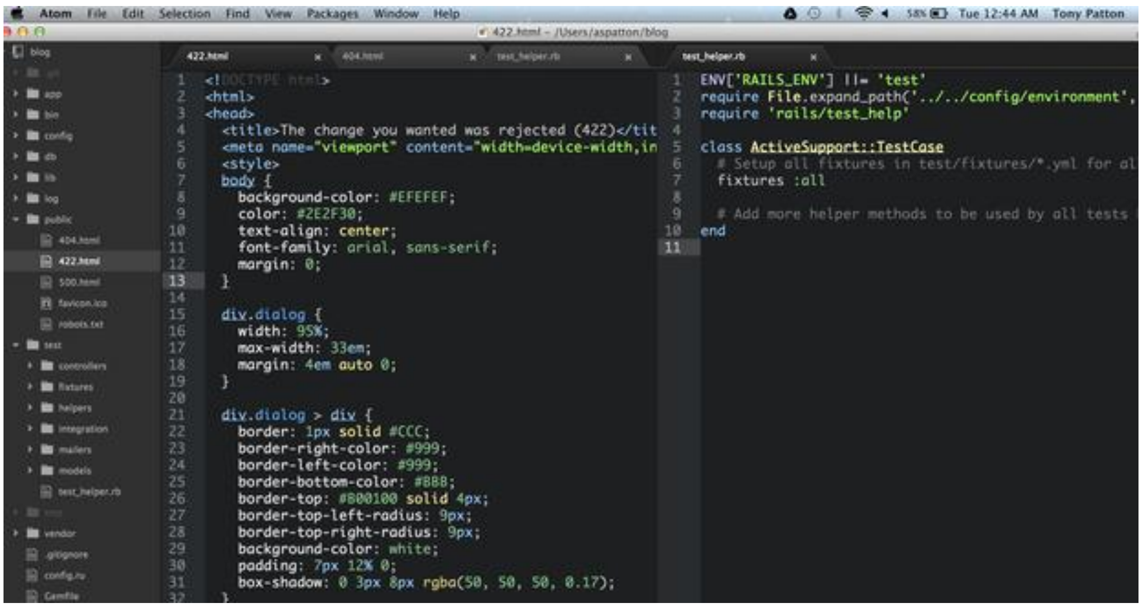
Node.js is a JavaScript runtime environment that is used to develop and run various types of applications (Node.js Foundation 2016). Node.js is commonly used to create Web servers and networking tools with the support of many modules. Lumi front-end application uses Node.js to create a web server since deploying with Node.js using Azure Web App Service is our choice for deployment. Further information about Azure Web App Service and how it is used is provided in “Development tools” section.

6.3 Development tools

In this section, all programs, tools, and services that were used in development process of Lumi front-end application are described. Atom is used as our preferred text editor; Git is used as our version control system; Grunt and Yeoman are our choice for automatic task runner and scaffolder tools; ngDoc was used to generate documentation of the source code; and Azure Web App Service was selected as deployment hosting service.

6.3.1 Atom

Atom is a text editor fitting for web development. It has several benefits such as being free, customisable with built in package manager, multiple panes, and a file system browser. (GitHub 2016)



```
<!DOCTYPE html>
<html>
<head>
  <title>The change you wanted was rejected (422)</tit
  <meta name="viewport" content="width=device-width,in
  <style>
  body {
    background-color: #EFEFEF;
    color: #2E2F30;
    text-align: center;
    font-family: arial, sans-serif;
    margin: 0;
  }

  div.dialog {
    width: 95%;
    max-width: 33em;
    margin: 4em auto 0;
  }

  div.dialog > div {
    border: 1px solid #CCC;
    border-right-color: #999;
    border-left-color: #999;
    border-bottom-color: #BBB;
    border-top: #B00100 solid 4px;
    border-top-left-radius: 9px;
    border-top-right-radius: 9px;
    background-color: white;
    padding: 7px 12% 0;
    box-shadow: 0 3px 8px rgba(50, 50, 50, 0.17);
  }
```

```
ENV['RAILS_ENV'] ||= 'test'
require File.expand_path('../../config/environment',
require 'rails/test_help'

class ActiveSupport::TestCase
  # Setup all fixtures in test/fixtures/*.yml for al
  fixtures :all

  # Add more helper methods to be used by all tests
end
```

Figure 6-5: Atom user interface

6.3.2 Git

Git is a widely-used version control system, which helps a software team manage changes to source code over time. Git keeps track of every modification to the code by each contributor in a special kind of database. In Git, every developer's working copy of the code is also a repository that can contain the full history of all changes. (Atlassian 2016)

Developing software without using Git is risky, because it provides the developer with backups of different development versions. Git can also enable developers to move faster and it allows software teams to preserve efficiency and agility as the team scales to include more developers. Git has many features to help with development process:

- A complete long-term change history of every file: Having the complete history enables going back to previous versions to help in root cause analysis for bugs and it is crucial when needing to fix problems in older versions of software.
- Branching and merging: Having team members working concurrently is the main benefit. Creating a “branch” in Git keeps multiple streams of work independent from each other while also providing the facility to merge that work back together, enabling developers to verify that the changes on each branch do not conflict.
- Traceability: Being able to trace each change made to the software and being able to annotate each change with a message describing the purpose and intent of the change can help with reading the code and understanding the code and therefore enable developers to make correct changes.
- Faster release cycle: The ultimate result is a faster release cycle. These capabilities facilitate an agile workflow where developers are encouraged to share smaller changes more frequently. In turn, changes can get pushed down the deployment pipeline which makes Git work very well with continuous integration and continuous delivery environments.
- Community: In many circles, Git has come to be the expected version control system for new projects. If the team is using Git, odds are new developers do not have to be trained on workflow, because they’ll already be familiar with distributed development. (Atlassian 2016)

6.3.3 Yeoman & Grunt

Yeoman is a scaffolding tool which helps kick-starting new projects, prescribing best practices and tools to configure an effective build process and pull relevant dependencies needed. Yeoman can be used by running a generator, which is basically a plugin, in the terminal with the ‘yo’ command to scaffold complete projects or useful parts. The Yeoman workflow includes three types of tools for enhancing convenience and effectiveness of a web application development process: ‘yo’ is the tool for scaffolding; Gulp or Grunt can be chosen as building tool and ‘npm’ or ‘bower’ as package manager.

The Package Manager is used for dependency management, so that developers can use third-party code to streamline the development process. npm and Bower are two popular options.

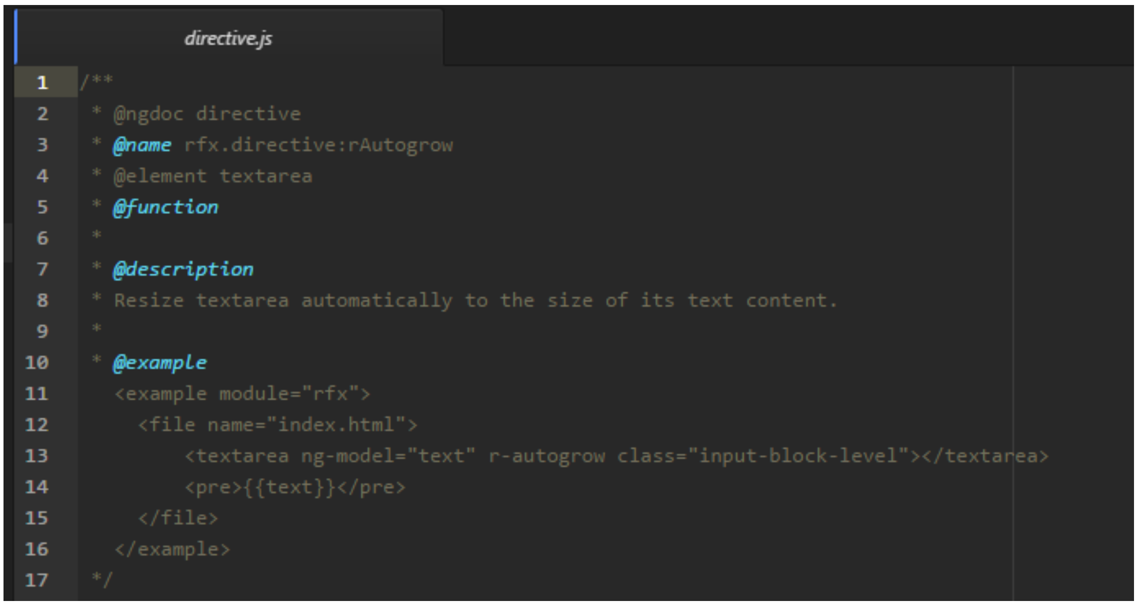
The Build System is used to build, preview and test the project. Grunt, alongside Gulp, is a popular option. Grunt ecosystem consists of many plugins to automate different tasks needed to preview the code during the development process, test the code with various situations as well as build a project from development to production.

With its generators, Yeoman creates a “client-side stack” which comprises of “tools and frameworks that can help developers quickly build web applications”. For example, with the yeoman angular generator, instead of manually installing every plugins and tools and setting up at the start of every project, ‘yo angular’ command can be run and Yeoman will take care of providing everything needed for developing, building and testing applications. Yeoman angular also includes support for linting, minification and much more, so applications can be well documented and optimized. (The Yeoman Team 2016)

In particular, the build tool Grunt automates many common tasks that are a part of many projects. The less attention must be spent on performing repetitive tasks like minification, compilation or unit testing, the easier it is for me to focus on developing the product. After a Grunt file is configured, a task runner can do most of such mundane works with minimal effort. (2016)

6.3.4 ngDoc

ngDoc is an API documentation generator for AngularJS, based on another documentation generator for JavaScript called JsDoC. ngDoc has a purpose of documenting the API of a JavaScript application or library. ngDoc supports standards tags for generic JavaScript such as name, parameter, returns, description, example as well as some other tags specific for Angular such as directive, service, and scope (GitHub 2014). Documentation comments are added directly to the source code, right alongside the code itself. Below is an example of the comment block:


directive.js

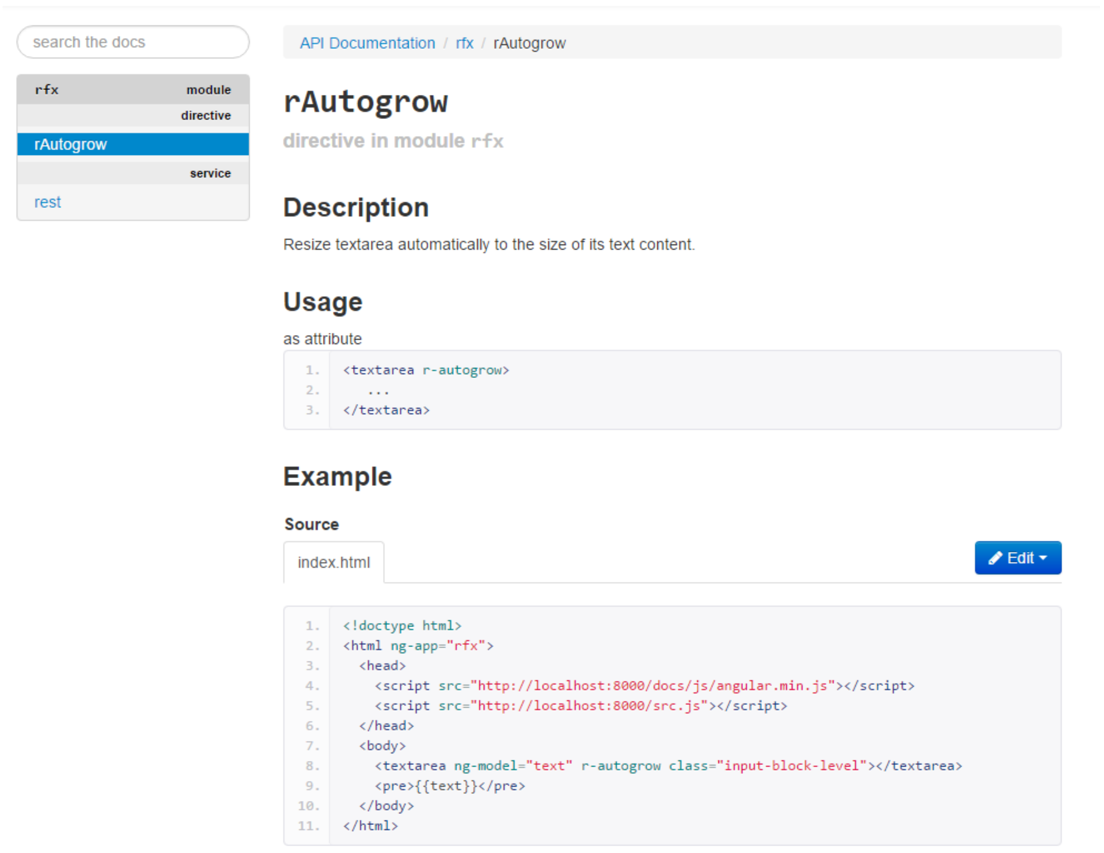
```
/**
 * @ngdoc directive
 * @name rfx.directive:rAutogrow
 * @element textarea
 * @function
 *
 * @description
 * Resize textarea automatically to the size of its text content.
 *
 * @example
  <example module="rfx">
    <file name="index.html">
        <textarea ng-model="text" r-autogrow class="input-block-level"></textarea>
        <pre>{{text}}</pre>
    </file>
  </example>
 */
```

Figure 6-6: Comment block to generate ngDoc documentation

There are several benefits of generating documentation from source code comments

- All the documentations are kept coordinated as the code changes.
- Version-specific documentation can be available by simply checking out a version of the application and running the build.

The ngDoc tool then scans the source code and generates a complete HTML documentation website.



rAutogrow

directive in module rfx

Description

Resize textarea automatically to the size of its text content.

Usage

as attribute

```
<textarea r-autogrow>
   ...
</textarea>
```

Example

Source

index.html

```
<!doctype html>
<html ng-app="rfx">
  <head>
    <script src="http://localhost:8000/docs/js/angular.min.js"></script>
    <script src="http://localhost:8000/src.js"></script>
  </head>
  <body>
    <textarea ng-model="text" r-autogrow class="input-block-level"></textarea>
    <pre>{{text}}</pre>
  </body>
</html>
```

Figure 6-7: The resulting documentation website

6.3.5 Azure Web App Service

App Service Web Apps is a compute platform that is developed for hosting websites and web applications. This platform-as-a-service (PaaS) offering of Microsoft Azure manages the infra-structure to run and scale applications and lets developers focus on the business logic of their applications.

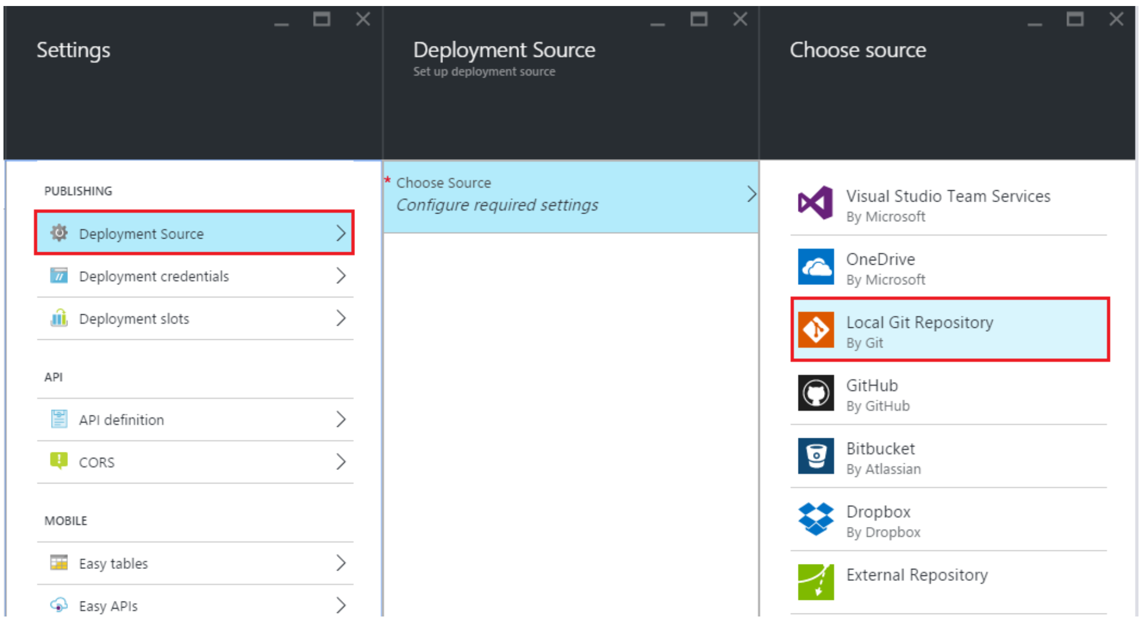
In App Service, Azure provides web apps as the compute resources for hosting a website or web application. The web apps can run on shared or dedicated virtual machines managed by

Azure and isolated from other customers, depending on the pricing tier chosen. Software code can be in any language or framework that is supported by Azure App Service, such as ASP.NET, Node.js, Java, PHP, or Python.

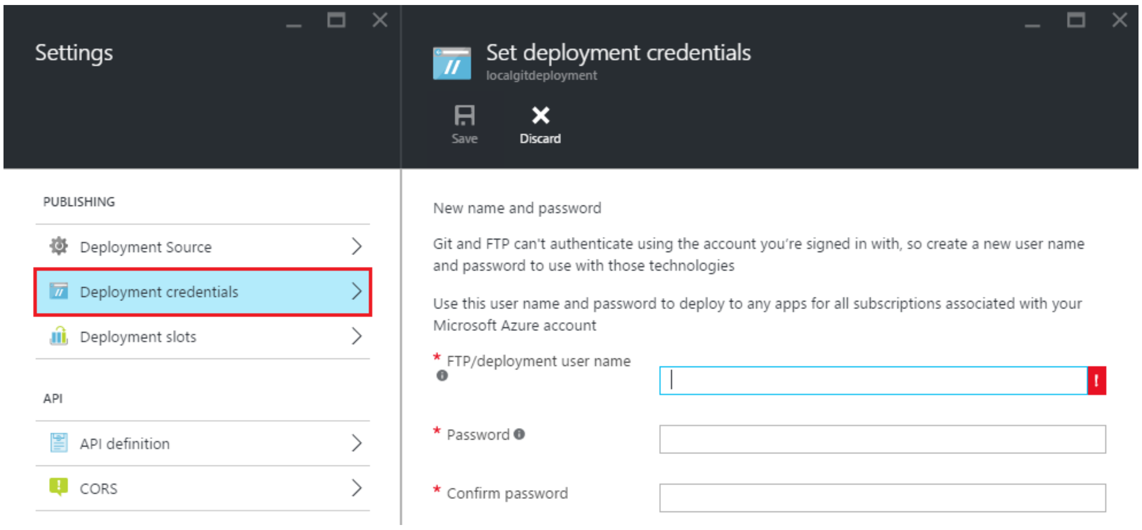
Azure App Service maintains the application framework for Lumi web front-end app, which is Node.js. In addition, application framework can be customized such as choosing version for Node. With web server and application framework being managed by Azure App Service, deploying process is simplified to only deploying code, binaries, content files, and their respective directory structure, to the /site/wwwroot directory in Azure. App Service supports many deployment options, but since Lumi web UI app is already managed by a local Git repository, the most convenient option is repository-based deployment with manual sync from local Git (Lin 2016).

To deploy an existed local Git repository to Azure App Service, I follow the instruction provided by Microsoft Azure:

1. Log in to the Azure Portal.
2. Choose Local Git Repository as the deployment source



3. Set deployment credentials



New name and password

Git and FTP can't authenticate using the account you're signed in with, so create a new user name and password to use with those technologies

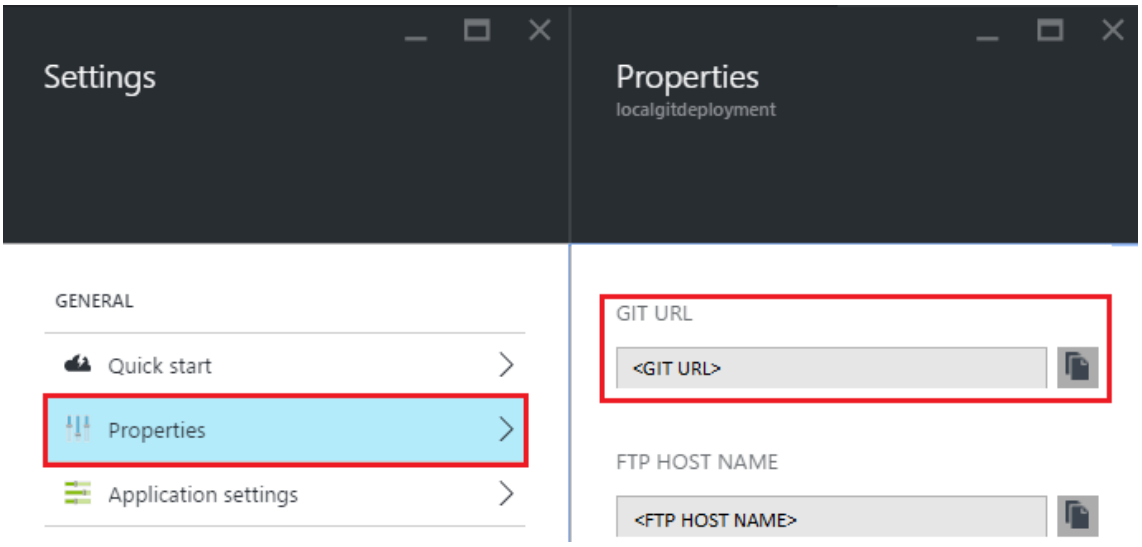
Use this user name and password to deploy to any apps for all subscriptions associated with your Microsoft Azure account

* FTP/deployment user name

* Password

* Confirm password

4. Get remote Git repository UR



GIT URL

<GIT URL>

FTP HOST NAME

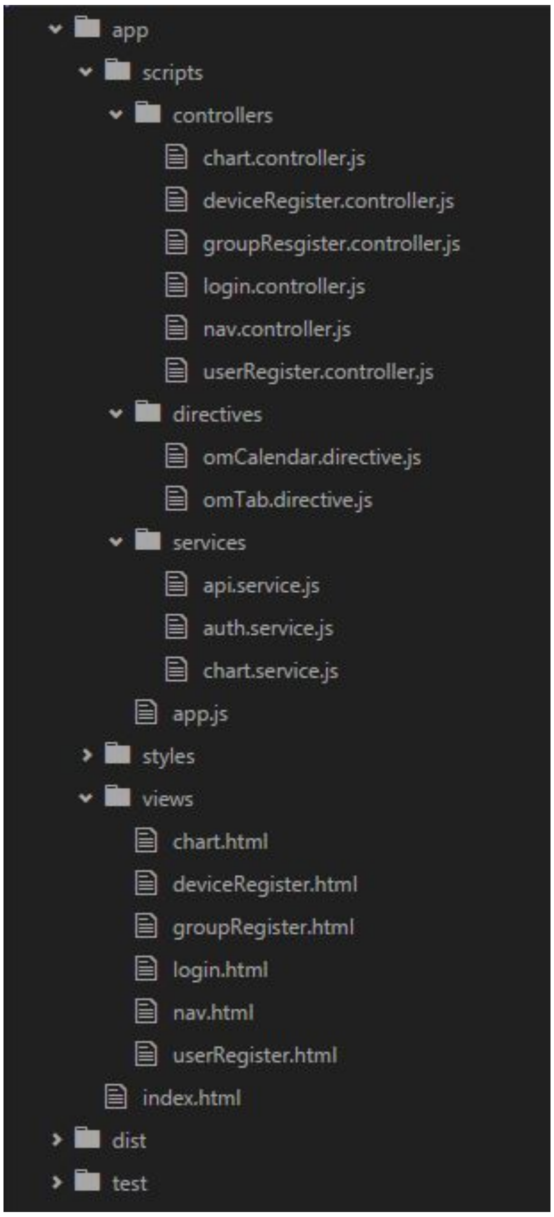
<FTP HOST NAME>

5. Add the remote Git repository and push the content of the local Git to Azure App Service (Grigoriu 2016)

6.4 Directory structure

Directory structure is the way folders and files located in a project folder. A good directory structure should allow developers to open and work with all the related files for a feature, as well as identify the purpose and function of a file with minimal efforts. This makes an application easily maintainable and extensible.

There are some commonly used directory structures for an AngularJS application. Sort by type: controllers have their own folder, views have their own folder, external libraries have their own folder. This structure is preferable when writing a small application since its separation of concerns makes it easier for the reader to follow and have a conceptual idea of what the application is trying to do.

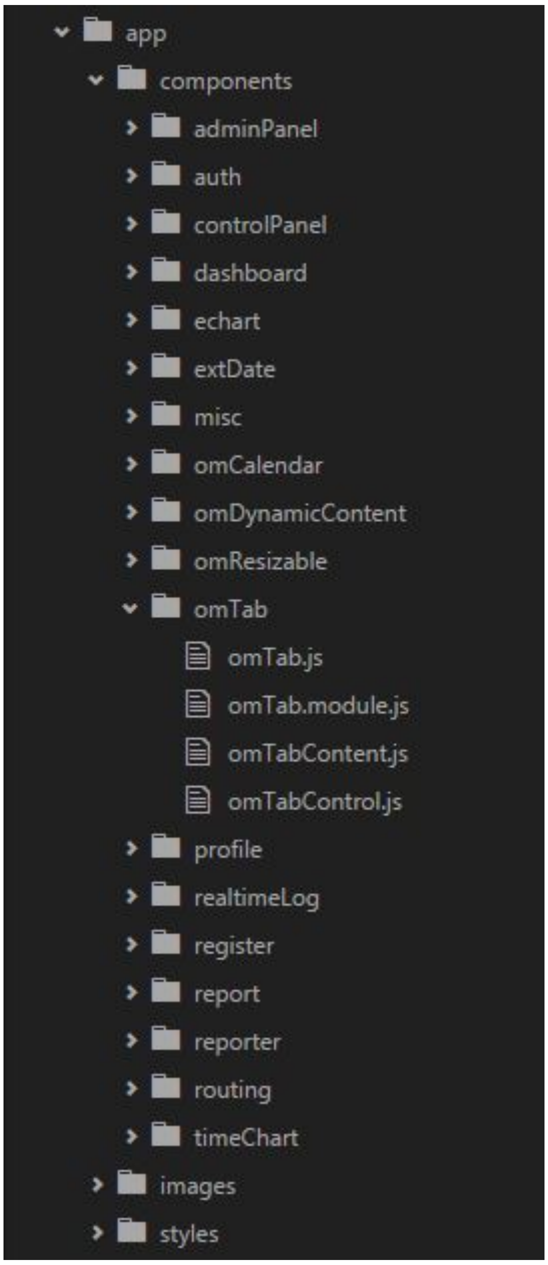


```
app
  scripts
    controllers
      chart.controller.js
      deviceRegister.controller.js
      groupResgister.controller.js
      login.controller.js
      nav.controller.js
      userRegister.controller.js
    directives
      omCalendar.directive.js
      omTab.directive.js
    services
      api.service.js
      auth.service.js
      chart.service.js
    app.js
  styles
  views
    chart.html
    deviceRegister.html
    groupRegister.html
    login.html
    nav.html
    userRegister.html
  index.html
dist
test
```

Figure 6-8: An example of a sort-by-type directory structure

Files are located based on their type, so all controller files are group together, similarly for directives, services, and views. Although easy to understand at the beginning, as the application grows, sort by type becomes less useful since all files that related to a feature are scattered in several folders and to work with a feature, developers must go through several folders.

A better directory structure is sort by components. It is based on a principle that an ideal AngularJS app structure should be combined with single purposed reusable components. (Kukic 2014)



```
app
  components
    adminPanel
    auth
    controlPanel
    dashboard
    echart
    extDate
    misc
    omCalendar
    omDynamicContent
    omResizable
    omTab
      omTab.js
      omTab.module.js
      omTabContent.js
      omTabControl.js
    profile
    realtimeLog
    register
    report
    reporter
    routing
    timeChart
  images
  styles
```

Figure 6-9: An example of a sort-by-feature directory structure

The components folder contains all components of the application. In each component folder, there are all files needed for the component to work including views, controllers, services, and directives. Specifically, Lumi web UI application includes main components such as routing, register, chart, authentication, and control panel. This way, each component resembles a mini-MVC application. If the component has multiple related views, these files will be further separated into ‘views’, ‘controllers’, ‘services’ subfolders.

The benefits of this modularized approach include:

- Maintainability: The approach above logically divides the application into smaller components and I am easily able to locate all files necessary for any component that I am working with.
- Extendibility: The code will be much easier to extend. Adding new directives and pages will not affect existing folders. Additionally, with this approach, components can be added or remove from the application with relative ease.

6.5 Routing component

Routing component consists of a config file (routing.conf.js) where all UI-Router states are defined. The navigation component contains a template file (nav.html) with a left navigation bar where user can find the button lists of different views and navigate where they want to go, and a controller (nav.controller.js) which handles behaviour logic of the navigation bar including showing the add new service popup as well as logging the user out.

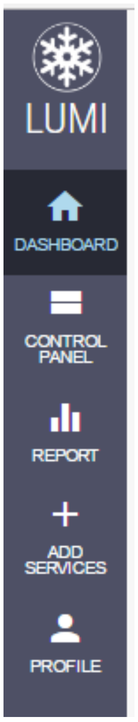


Figure 6-10: Lumi front-end application navigation bar

6.6 Authentication component

The authentication component contains a template file (login.html) with an HTML form that allows the user to input their username and password and a button to submit them. It also has a controller file (login.controller.js) to make HTTP request to the back-end API to authenticate the user and listen to the response. The user will be navigated to the dashboard if the response is successful, otherwise a message will be displayed on screen announcing that the authentication has failed.

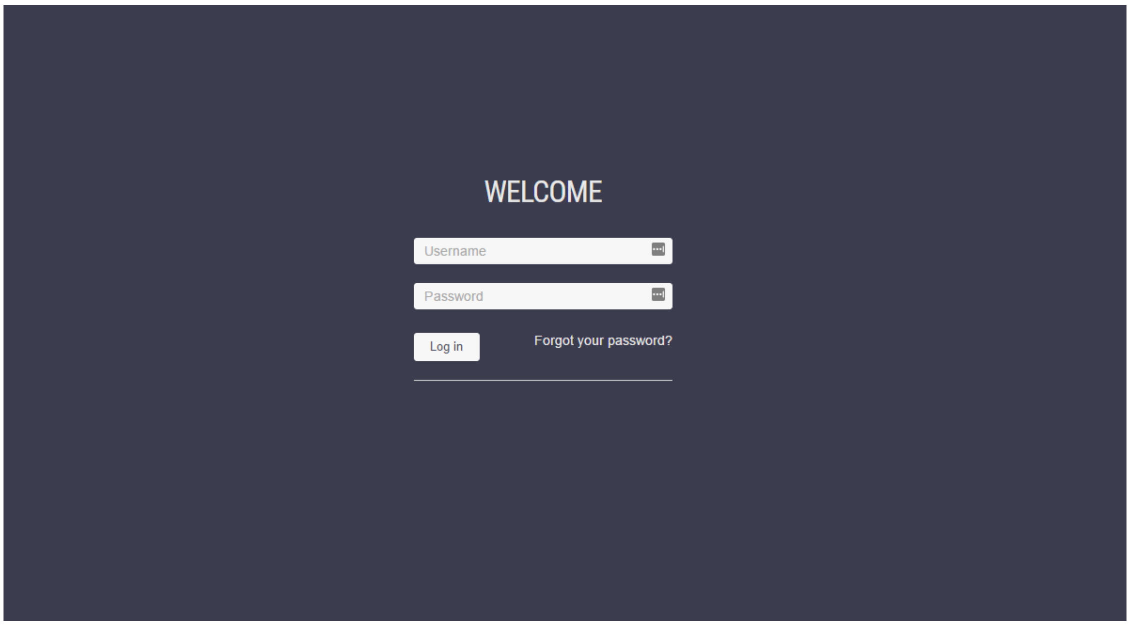


Figure 6-11: Log in page

6.7 Chart component

Since chart data can be filtered in several types, namely by day, by week and month, the chart component includes a template service (chartTemplate.js) to choose a type of chart to create. There is also a service file (chartCreator.js) that receives a template, get raw data and filter them to chart, and a directive (chart.js) that renders the chart as HTML canvas. The figure below shows the resulting chart component being served to the user

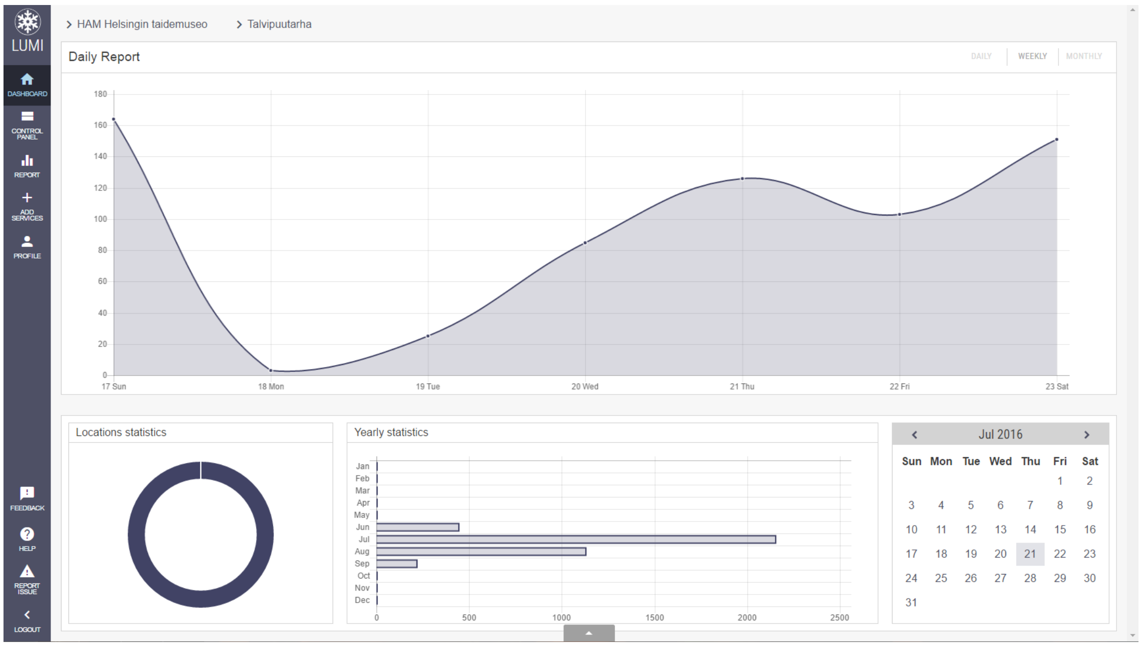


Figure 6-12: Dashboard view with chart components

The benefit of separating chart logic and chart rendering is that the chart data can be reused to render in different ways such as table of a different chart type. A separate template service is easier to document and make changes.

6.8 Build and deployment process

The purpose of the build process is to reduce the size of the code, remove development specific code, and link all backend request to the web API instead of the local one. To achieve this with minimal work and following industry standards, I use a Grunt process generated by Yeoman called 'build', with some modifications for this project.

Yeoman creates a default Grunt build task which does the following:

- Clean 'dist' folder: clear all files and folders in 'dist' directory to prepare for the new build.
- Link third party dependencies to index: include third party libraries and components such as Angular, angular-chart or UI-Router. This is a automation process so developer must not forget third party dependencies in the build.
- Concatenate and uglify JavaScript files: all script files are combined into one big file, and uglify removes spaces and line-brakes from the one big script file. These two processes reduce the size of the build.
- Pre-process Sass files into CSS files: read ".scss" files and create ".css" files for application styles.

- Copy other files including images and fonts to the ‘dist’ folder: automatic process to save time on manual copying of images and fonts.
- Minify JavaScript, CSS, and HTML files: Minification process removes unnecessary or redundant data without affecting how the browser processes the resource. This process further reduces the size of the build, thus improves performance of the application.

Modify Yeoman build task for Lumi front-end application

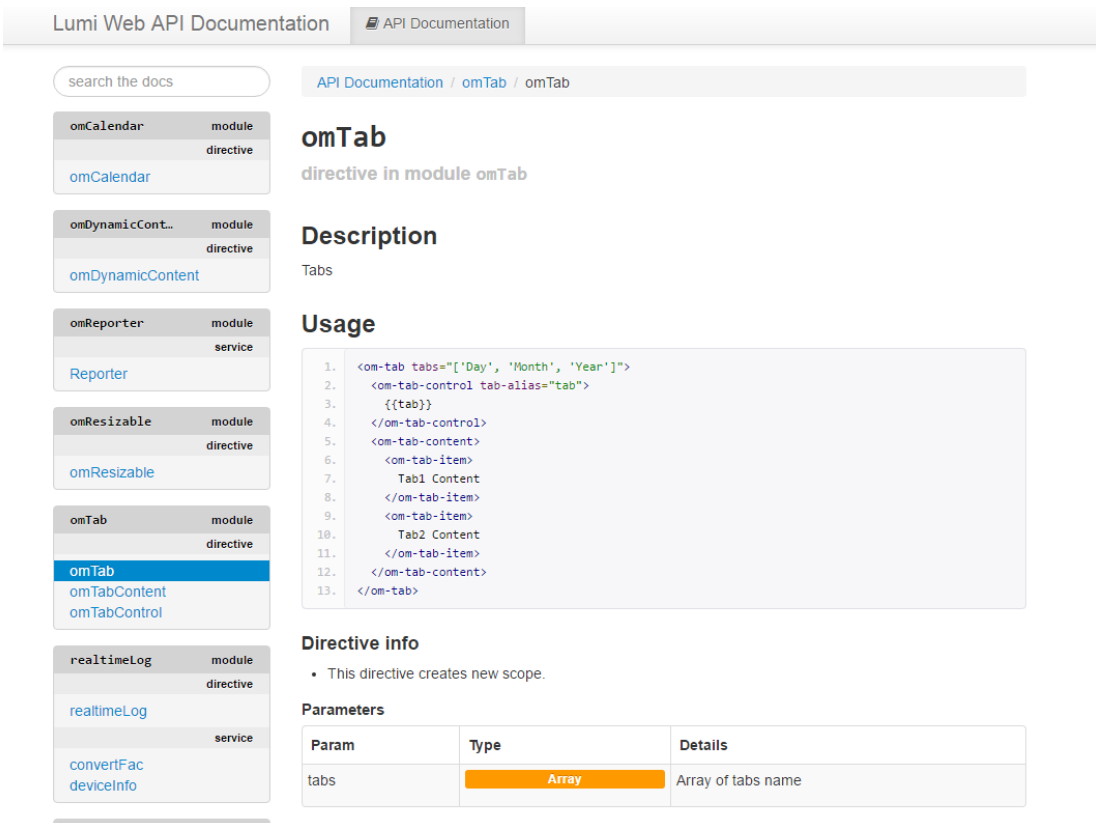
- Update tasks with new directory structure
- Remove logging statements
- Replace URL of development API to production API URL

For the server to serve the content to browsers on request, I wrote a Node.js script to. With AngularJS handle all the routing, logic and rendering for the client, this run script only need to redirect all request to a ’dist’ folder where all files needed for the application to work are located.

As mentioned above, Lumi front-end app uses git to deploy to azure web services. To separate development build with production build, I use two separate app services. With the build tasks being automatic, deployment to development app services can be done daily in a few commands. The development build then can be evaluated by the manager and if approved, can be pushed to production.

6.9 Documentation

As mentioned above, the documentation is generated following ngDoc. Because the project is already using Grunt, I integrated the documentation process into the workflow with a third-party grunt task called grunt-ngdocs. The task is modified and registered in a ‘grunt docs’ task which will serve the output documentation web page and update whenever any changes is made in the source code documentation. The final documentation website can be seen below



omTab

directive in module omTab

Description

Tabs

Usage

```
<om-tab tabs="['Day', 'Month', 'Year']">
  <om-tab-control tab-alias="tab">
    {{tab}}
  </om-tab-control>
  <om-tab-content>
    <om-tab-item>
      Tab1 Content
    </om-tab-item>
    <om-tab-item>
      Tab2 Content
    </om-tab-item>
  </om-tab-content>
</om-tab>
```

Directive info

- This directive creates new scope.

Parameters

Param	Type	Details
tabs	Array	Array of tabs name

Figure 6-13: Lumi front-end application documentation website

Since the documentation is only used internally, it is not hosted on the server, but is managed alongside the souce code. The ‘grunt docs’ command can be used later to access the documentation.

7 Testing

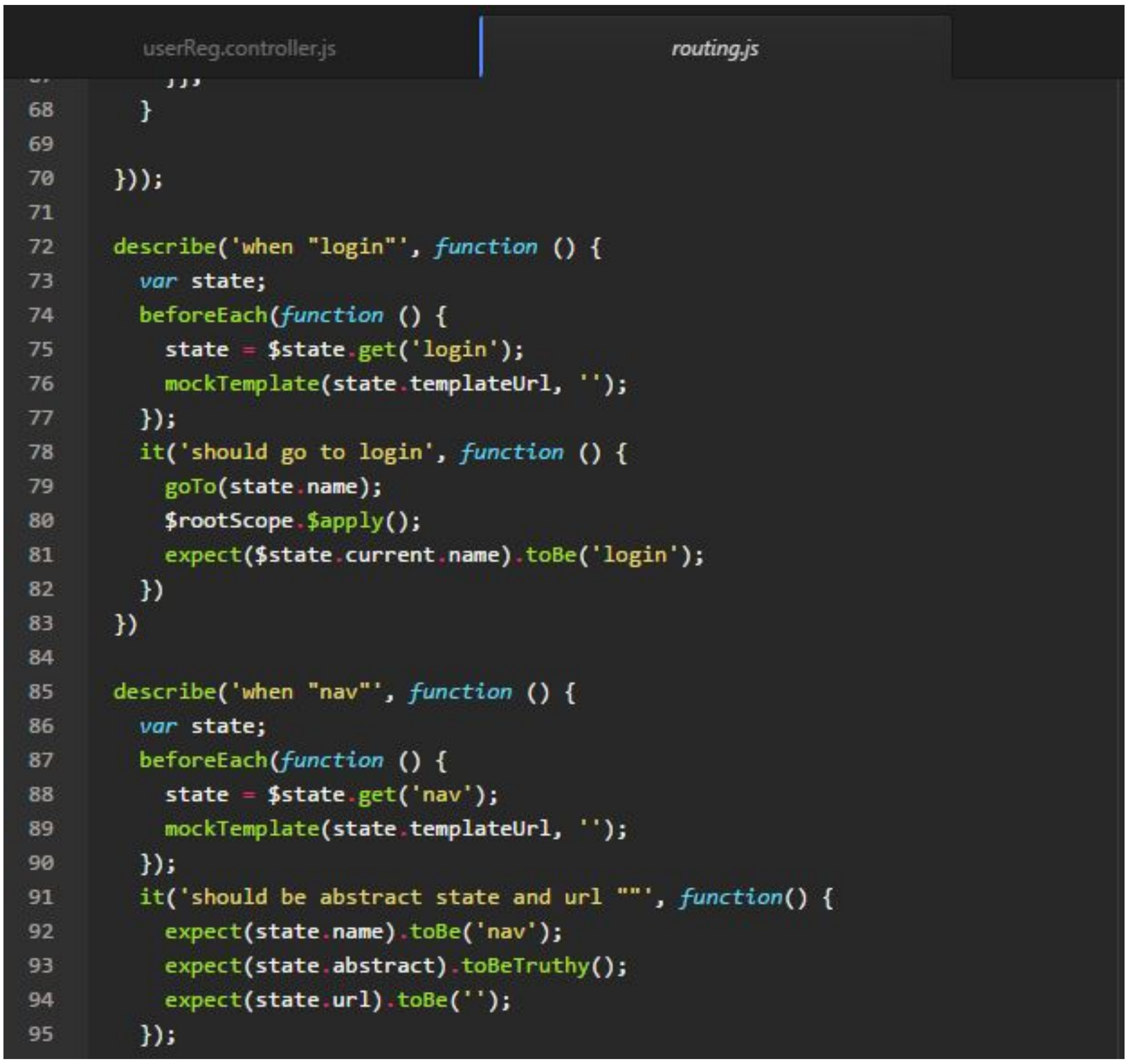
Testing is the process of verifying that the program works as intended and discover bugs before it is released. The results of the test consist of errors, anomalies, or information about the program’s quality attributes. Typically, a commercial software system must go through three stages of testing:

1. Development testing, where the system is tested during development to discover bugs and defects.
2. Release testing, where a separate testing team tests a complete version of the system before it is released to users.
3. User testing, where users or potential users of a system test the system in their own environment. (Sommerville 2011)

In Lumi front-end application project, three specific testing methods were conducted. Unit testing was used for development testing. Release testing was carried out by the project manager one week before the release date, and user testing was done in cooperation with a pilot test customer in Helsinki region.

7.1 Development testing

Unit test scenarios were created for major components such as chart services, routing services and log in service. For example: Routing should allow user to go to states they have access to, meaning unauthenticated user can only access login page, regular authenticated user cannot add new resources.



```
    }

  }));

  describe('when "login"', function () {
    var state;
    beforeEach(function () {
      state = $state.get('login');
      mockTemplate(state.templateUrl, '');
    });
    it('should go to login', function () {
      goTo(state.name);
      $rootScope.$apply();
      expect($state.current.name).toBe('login');
    })
  })

  describe('when "nav"', function () {
    var state;
    beforeEach(function () {
      state = $state.get('nav');
      mockTemplate(state.templateUrl, '');
    });
    it('should be abstract state and url ""', function() {
      expect(state.name).toBe('nav');
      expect(state.abstract).toBeTruthy();
      expect(state.url).toBe('');
    });
```

Figure 7-1: An example unit test case

7.2 User testing

The user testing phase was conducted after the app was deployed to Azure Web App Service by a pilot customer located in Helsinki city. Test period started in June 2016 and finished in August 2016, during which a set of cameras and a computer was installed at the tested location to gather people counting data. Data was sent to the back-end API to be saved in the database in real time. The front-end app was accessible to view data and interact with the system.

There are many functionalities tested by the users during the pilot period.

First 1UP Media admin user logged in to add a new company, user and device for the customer company and the test user. The login was successful, and so was adding new resources. To test saving authentication feature, 1UP admin user refresh the browser page. The authentication token was successfully saved and the user did not have to log in again.

After the installation team installs the camera and device and configure them, the regular user can log in to see the real-time chart for that day. He/she can also select a filter so the chart data can cover different devices or time range. All tasks were completed successfully.

To finish the tasks for the installation day, the regular user logged out of the application, and tried to access the service without authentication. The attempt was unsuccessful means the log out function was functional.

After the installation day, the regular user can access the application on a network connected computer at any time. The chart fulfilled the requirements of being functional and performant. There were some minor issues during the pilot test, but they were fixed quickly and the user was satisfied.

7.3 User feedback

Towards the end of the pilot period there was an extra feature requested that the customer can print the chart data as PDF file for report purposes. The development team responded quickly and the feature was developed, tested, and released before the end of the pilot. Overall, the customer was pleased that Lumi people counting was usable and provided insightful data.

8 Maintenance

The maintenance process starts after the application is released in August 2016. The development team follows bug reports and feature requests from users, and from that a new development cycle starts with requirement analysis, design, implementation, testing and documenting. Since the maintenance process is outside the scope of this paper, it will not be discussed further.

9 Conclusion

Overall, Lumi front-end application has succeeded in providing user-friendly interface for its customers to view critical user behaviour data and manage their counting data services as well as gaining business value for 1UP Media. With strong emphasis on extendibility and maintainability, the application can be expanded to include several services beside people counting data. The development process can be used as an example of creating front-end web application, and its practices can be reuse for later projects.

References

AngularUI. 2016. About States. Accessed 2016. https://ui-router.github.io/guide/states

Atkins, T. & Sapin, S. 2014. CSS Syntax Module Level 3. [Document] W3C. CSS Working Group (3) Accessed 2016. https://www.w3.org/TR/css-syntax-3/

Atlassian. 2016. What is Git. Getting Git Right. Accessed 2016. https://www.atlassian.com/git/tutorials/what-is-git

Atlassian. 2016. Why Git for your organization. Getting Git Right. Accessed 2016. https://www.atlassian.com/git/tutorials/why-git

Bass, L., Clements, P. & Kazman, R. 2003. Software Architecture in Practice. 2nd ed. Boston: Addison Wesley.

Bitrix24. 2016. Project Management and Tasks. Accessed 2016. https://www.bitrix24.com/features/tasks.php

Bondi, A. 2000. Characteristics of Scalability and Their Impact on Performance. In: Proceedings of the 2nd international workshop on Software and performance, Ottawa, 2000. ACM New York.

Faulkner, S., Eicholz, A., Leithead, T. & Danilo, A. 2016. HTML 5.1. [Document] W3C. (5.1) Accessed 2016. https://www.w3.org/TR/html/introduction.html#a-quick-introduction-to-html

Fielding, R. 1999. Hypertext Transfer Protocol -- HTTP/ 1.1. [Document] Internet Engineering Task Force. (1.1) Accessed 2016. https://tools.ietf.org/html/rfc2616#section-1

Flanagan, D. 2011. JavaScript: The Definitive Guide. 6th ed. Sebastopol: O'Reilly.

GitHub. 2014. Writing AngularJS Documentation. [Wiki] Accessed 2016. https://github.com/angular/angular.js/wiki/Writing-AngularJS-Documentation

GitHub. 2016. Atom. Accessed 2016. https://atom.io/

Grigoriu, D. 2016. Local Git Deployment to Azure App Service. Microsoft Azure. Accessed 2016. https://docs.microsoft.com/en-us/azure/app-service-web/app-service-deploy-local-git

Grunt. 2016. Accessed 2016. http://gruntjs.com/

Halpern, B. 2016. The Fat Client - Thin Client Debate. Dev. https://dev.to/ben/the-fat-client---thin-client-debate

Jewett, M. 2015. A Comparison of Frontend and Backend Web Development. Bloc. https://blog.bloc.io/frontend-vs-backend-web-development/

Johansson, N. & Löfgren, A. 2009. Designing for extensibility. Bachelor Thesis. Gothenburg: University of Gothenburg 2016.

Kukic, A. 2014. AngularJS Best Practices: Directory Structure. Scotch.io. Accessed 2016. https://scotch.io/tutorials/angularjs-best-practices-directory-structure

Lau, D. 2013. 10 Reasons Why You Should Use AngularJS. Site Point. Accessed 2016. https://www.sitepoint.com/10-reasons-use-angularjs/

Lin, C. 2016. Deploy your app to Azure App Service. [Article] Microsoft Azure. Accessed 2016. https://docs.microsoft.com/en-us/azure/app-service-web/web-sites-deploy

Lindley, C. 2014. Front-end Driven Applications – A New Approach to Applications. Telerik. http://developer.telerik.com/featured/front-end-driven-applications-new-approach-applications/

Mell, P. & Grance, T. 2011. The NIST Definition of Cloud Computing. Special Publication. Gaithersburg: NIST NIST.

Node.js Foundation. 2016. Node.js. Accessed 2016. https://nodejs.org/en/

Object Management Group. 2005. Introduction to OMG's Unified Modelling Language (UML). UML. http://www.uml.org/what-is-uml.htm

Object Management Group. 2005. What is UML. Unified Modelling Language. Accessed 2016. http://www.uml.org/what-is-uml.htm

Osmani, A. 2015. Learning JavaScript Design Patterns. O'Reilly.

Pose, A. 2014. Cookies vs Tokens. Getting auth right with Angular.JS. Auth0. Accessed 2016. https://auth0.com/blog/angularjs-authentication-with-cookies-vs-token/

Riehle, D. 2000. Framework design, a role modelling approach. PhD Thesis. Zurich: Swiss Federal Institute of Technology Zurich.

Skrchevski, B. 2015. High cohesion and loose coupling. Accessed 2016. https://thebojan.ninja/2015/04/08/high-cohesion-loose-coupling/

Sommerville, I. 2011. Software Engineering. 9th ed. Boston: Addison-Wesley.

Staticapps.org. 2016. Defining Static Web Apps. Accessed 2016. https://staticapps.org/articles/defining-static-web-apps/

The Yeoman Team. 2016. Yeoman. Accessed 2016. http://yeoman.io/

Tutorials Point. 2016. Software Design Basics. Software Engineering Tutorial Accessed 2016. http://www.tutorialspoint.com/software_engineering/software_design_basics.htm

Figures