MEAN Software Stack

MEAN-stack and its relative strengths and weaknesses relative to its competitors

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulun opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

Visamäki, kevät 2018

Toni Nevalainen

Visamäki unit
Degree programme in business information technology
ICT

Author Toni Nevalainen **Year** 2018

Subject of Bachelor's thesis MEAN Web-development Software Stack

ABSTRACT

The purpose of this thesis was to research the features and the relative strengths and weaknesses of MEAN software stack compared to its competitors. The thesis didn't have a commissioner and instead was done in intent to grow the expertise and knowledge base of the author.

The thesis showcases the different components of MEAN-stack and their individual functions and showcases the function of the software stack in form of a simple documented application.

MEAN-stack is a highly popular single-page web-application software stack that provides fast prototyping and development speed. Its use of Node.js JavaScript runtime environment enables it to serve large number of concurrent clients efficiently. Compared to its competitors like MERN-stack, which uses a distinct client-side framework compared to MEAN, MEAN offers more complete and faster to operate set of tools for single-page web-application development at the cost of longer time required to gain proficiency in them. MEAN is also more processing and network transfer intensive to the web-application clients because of its more complete, bulkier initial toolset compared to MERN stack´s more modular one.

Keywords Web-development, MEAN, JavaScript, Framework, Software Stack

Pages 23 p.

Vocabulary

Software stack – Collection of software designed to support an application or a software solution.

Software framework – Software framework for supporting an application.

Application Programming Interface – Interface through which a software serves another software with data.

CONTENTS

1 INTRODUCTION

This thesis introduces MEAN-stack, a single-page web-applications development software stack that offers rapid prototyping and development speed at cost of higher time to gain proficiency. This thesis compares the relative advantages and weaknesses of MEAN-stack to its competitors like MERN-stack, which similarly focuses in JavaScript single-page application development. This thesis does not have a commissioner and was written in intent of gaining vital expertise in web-development programming in preparation of entering the workforce.

MEAN-stack is a single-page web-application software stack programmed throughout using JavaScript. MEAN-stack is highly popular tool in modern single-page application web-development market.

The thesis introduces the different elements of MEAN-stack and their individual histories and functions and the practical section of the thesis showcases its use in practice in form of a documented, simple MEAN-stack web-application. The example application follows the flow of information through MEAN-stack from the user interface of the application through the workings of the client-side framework to the server back-end and ultimately into the database used in MEAN-stack, MongoDB. The example application showcases basic functions of each of the components of MEAN-stack.

2 SOFTWARE STACK

Software stacks are sets of software brought together to act as a platform for an application. The different software of a software stack are often developed independently and brought together because they can together fill some specific purpose such as support a web-application. MEAN-stack for example consists of a server-side software framework, client-side software framework, server-runtime environment and a server database, forming a package that is fit for creation of single-page web-applications programmed throughout using JavaScript. Software stacks are often modular and can swap the software and frameworks they use as the project requires. MEAN-stack for example could swap its client-side framework Angular to another front-end framework or library, such as React without disrupting rest of the stack, effectively turning it into the MERN-stack. Software stack can also be expanded with addition of new software and most projects use a number of other software in addition to what the initially picked software stack provides. (Wodehouse 2018.)

Some relevant software stacks include LAMP, .NET and ruby on rails. Linux based web software stack, Windows based general-purpose software stack and a cross platform Ruby programming language based web application rapid development software stack respectively. (Wodehouse 2018.)

Software stacks vary in the level of structure they impose upon software projects developed using them. MEAN-stack for example, due to its use of the Angular front-end software framework, imposes far more significant control over the functions of the application as a whole compared to React JavaScript library using MERN-stack. (Wodehouse 2018.)

The standardization offered by software stacks also focuses of programming expertise in the software development market by providing central standards in which developers can become experts in. This eases acquisition of new programmers into software development projects using popular software stacks as existing expertise on the practices and tools used in popular stacks are naturally more prevalent in the developer workforce. (Wodehouse 2018.)

Software stacks are formed of variety of different software types, including run-time environments, databases, frameworks, libraries, tools, and whatever else software included in a set of software required to support some specific purpose or type of application. (Wodehouse 2018.)

3 MEAN-STACK

MEAN-stack is a single page web-application software stack. The abbreviation MEAN (MongoDB, Express, Angular, Node.js) was originally coined by Valeri Karpov in year 2013. MEAN-stack is a highly popular web-development software stack and is programmed throughout using JavaScript, a very popular web-development programming language. Using JavaScript throughout the stack facilitates code and API sharing between the different elements of the stack. For example, Express.js and Angular, the back- and front-end frameworks of MEAN could both utilize the same service that communicates using JSON (JavaScript Object Notation) file format without needing any additional middleware because JSON is interpreted natively by JavaScript runtimes. Due to JavaScript´s historical popularity and ease of learning, the available workforce skilled in it is large, which makes software projects programmed with it cheaper to staff. Large developer community also produces many community resources, libraries and frameworks for development use. (Williams 2017.)

MEAN-stack consists of MongoDB, Express.js, Angular and Node.js. A database, a back-end framework, a front-end framework and a server runtime respectively. Together they form a stack that is efficient at operation of single-page web-applications. Due to using Node.js as its server runtime, MEAN-stack is efficient at hosting web services that need to serve a great number of concurrent clients due to its low processing overhead. (Williams 2017.)

MongoDB is a NoSQL document database. It stores its data into individual documents that can contain many variables stored in different datatypes. Saving data in documents instead of rigid data tables like in a more traditional SQL database gives MongoDB versatility with the number of stored variables and can significantly speed up the development and prototyping of new applications. MongoDB developer MongoDB.inc has also created Mongoose, Express.js plugin that allows for data modeling on MongoDB, giving it data storage and querying functionality similar to a SQL database. (Mongodb.com 2018.)

Express is a minimalist back-end framework meant for creation of web-applications and web API:s. Express offers simplified routing, error handling and template engines out of box and is designed to be extendable using plugins. (Expressjs.com 2018.)

Angular is a client-side software framework specializing in single-page web-applications. Angular provides functions such as nested user interface components that can be combined to build user interfaces, view routing and HTTP services. With the toolset provided by angular, it is possible to create client-side single-page web-applications that contain nearly all the functionality required in a web-application, leaving the server mostly for shuttling data between the database and the client application. Moving majority of the code client-side reduces the processor workload incurred on the server and thus makes hosting such applications cheaper on the server. (Holmes 2016, p. 26.)

Node.js is a cross-platform JavaScript server runtime that utilizes a non-blocking input/output architecture to enable it to potentially serve tens of thousands of simultaneous connections. Node offloads tasks to be processed by the system kernel and engages with them only when the requests have completed or have failed. Node has low per connection overhead but is inefficient in tasks with high processing load due to its architectural limitations. (Nodejs.org 2018.)

MEAN-stack projects often utilize other frameworks and libraries as well, such as Grunt.js, which helps with automating many aspects of a project, such as minifying and linting used JavaScript files. (Tsuneo 2015.)

3.1 Node.js

Node.js is an open source cross-platform JavaScript server run-time environment. It operates on an event loop system that it utilizes for unloading requests by clients to be executed by the system kernel, only interacting with them once they are done or have failed. This enables Node.js to serve tens of thousands of simultaneous connections, enabling it to serve a great number of concurrent clients. Node can run on Linux, Mac OSX and Microsoft Windows systems after 2008R2/Windows 2012. Node was originally released by Ryan Dahl in 2009 and is a distributed development project under the stewardship of the Node.js foundation. (Nemeth 2018.)

Node runs on a single thread and uses V8 JavaScript engine, originally developed for Google Chrome web browser as its engine for parsing and executing JavaScript. Node can run any language that compiles into JavaScript before execution, giving a developer options on the language used. For example, TypeScript, a superset of JavaScript, is strongly typed. Strong typing makes testing easier and reduces changes of accidental type conversion during compilation and execution by requiring the data types of variables to be strictly defined in the code. Other examples include CoffeeScript which is designed to be more readable JavaScript and Dart, which is a general-purpose Google developed object-oriented language that can optionally be transcompiled into JavaScript. (Patel 2018.)

Node.js serves best as a platform for fast, scalable network applications. Node runs on a single thread and collects all its client connections into an event loop whereas a more traditional PHP server would allocate a thread for each connection and give them a dedicated chunk of RAM memory, usually 8 megabytes. In case of a rush hour, a great number of simultaneous connections can aggregate huge amounts of ram usage, potentially filling the server to capacity. Running out of memory causes the server to slow down and can crash the server, potentially making the website less appealing or unusable. Node.js manages this by setting the requests into an event loop and executing them asynchronously in-between receiving new requests, scheduling additional operations and responding to clients. (Capan 2013.)

Due to using JavaScript Node.js facilitates creation of software stacks programmed throughout using JavaScript. This makes it possible to share code and application programming interfaces between the different parts of the application. Software stack programmed using a single language reduces the amount of required language expertise in a project and eases communication between employees working on different parts of the stack. (Heller 2017.)

A software stack programmed using solely JavaScript can also use JSON, JavaScript Object Notation for moving data between its components without needing additional parsing as JSON can store JavaScript objects and variables and be directly parsed back into JavaScript variables and objects by modern JavaScript engines. (Heller 2017.)

Node.js has a great number of community created libraries, frameworks and development resources. NPM (Node Package Manager) is the default package manager of Node.js. The NPM command line client can be used to download packages from an online database called npm registry. The packages can be public or private. Public packages are available to everyone whereas private packages may require the developer to pay the creator to gain access to the package. (Holmes, 2016, p.8.)

3.1.1 Event loop

Node.js consists of Node.js run-time and the V8 JavaScript engine. The run-time handles oncoming connections, collecting them into an event loop from which it sends requests to be fulfilled by the system kernel and sends responses back to the client. Once requests sent to be executed have finished or have failed, they send a call-back into the event loop, signaling their readiness to be sent back or to proceed into their next stage. (Node.js.org, 2018.)

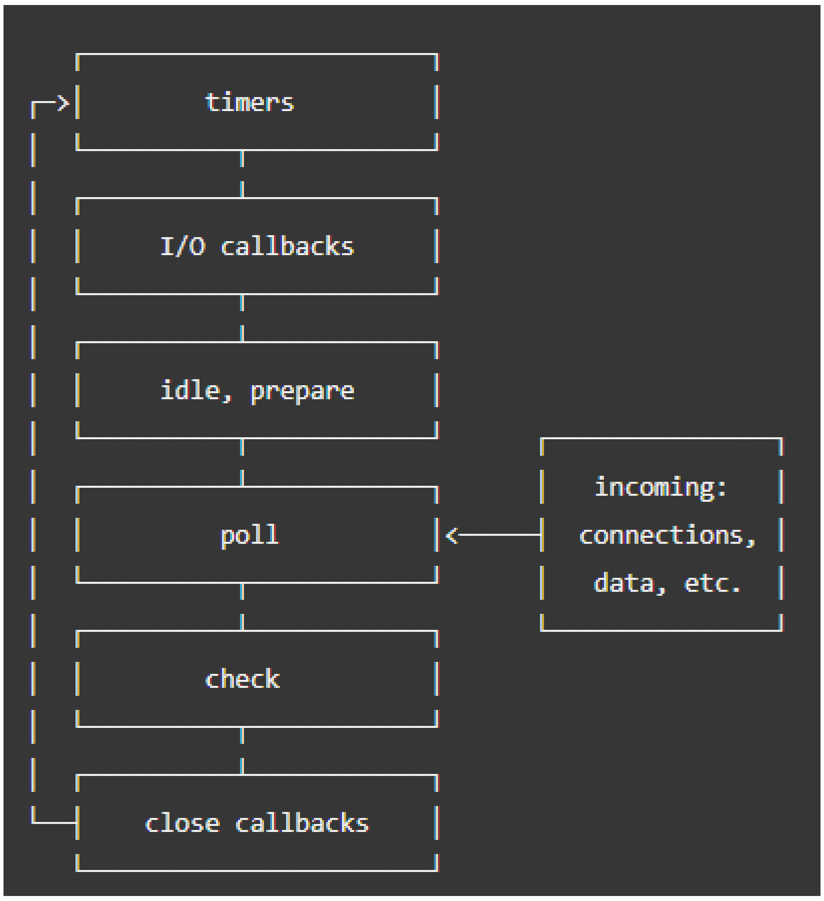


Figure 1. Simplified model of the event loop

The event loop is split into phases, showcased in a simplified form in figure 1. Timers phase executes call-backs scheduled by setTimeout() function. SetTimeout() specifies an amount of time the program must wait before executing a task. The poll phase executes input and output callbacks. More polling phase operations can be scheduled during and outside the polling phase. This can lead to the polling phase lasting longer than the timers phase call-back´s specified delay. As such the time specified in the SetTimeout() is the minimum time it takes for the task to start executing. Between each cycle of the loop Node.js checks if it has any requests, responses or timers waiting and stops running the cycle if there are none. (Node.js.org, 2018.)

3.1.2 Threading

Node.js runtime runs on a single thread by default, reducing its ability to scale vertically as its locked into a single CPU without the use of libraries such as cluster. Node.js collects all its client operations to a single event loop, removing the overhead that would be caused by context switching between different clients, relaying the processing required for the requests to the system kernel. System kernel then decides on how the requests are processed, usually spreading them across the multiple CPU cores found in most modern home and server hardware environments. (Ramirez 2017.)

Using the cluster library enables Node.js to scale across multiple cores and systems, enabling it to serve greater number of concurrent clients but using cluster requires the application to be programmed to use cluster. For a single thread Node.js application to use cluster, it must be reprogrammed to use it from bottom up. (Nodejs.org 2018.)

3.2 Express

Express is a minimalist Node.js back-end framework that offers request routing out of box and is designed to be easily extended to fit the needs of the software development project using plugins. (express.js.com, 2017.)

Express was originally released by TJ Holowaychuk under MIT license in 2010. Express was bought by Strongloop in 2014, which was in turn bought by IBM in 2015. IBM gave the stewardship of Express to the Node.js foundation in 2016. (Tzur 2016.)

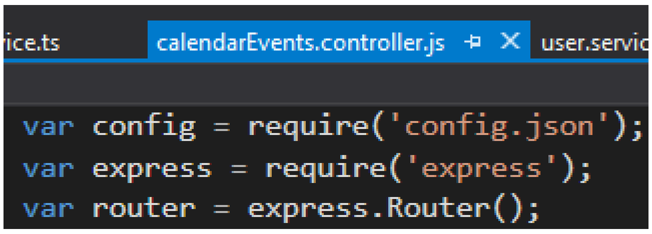
Express functions by providing a middleware object that the developer initializes and uses for the various features such as routing or rendering templates. Express routing intercepts incoming HTTP requests and routes them to the parts of the application code defined by the developer, based on the URL address and the type of HTTP request used. The Express middleware can also manipulate the request and response objects, streamlining the development process. (Expressjs.com 2017.)

Express is a versatile and lightweight framework that lends itself well for rapid development and prototyping with its simplistic syntax and wealth of community resources available for developer use. (Hallam 2018.)

Due to the minimalism and simplicity of Express, there are multiple other back-end frameworks built upon it, such as Feathers, a framework designed for very fast application development and prototyping. Other relevant frameworks include frameworks such as Express Gateway, a microservices API framework and Kraken, a Framework that extends upon Express and gives it extra functions and security features. (Expressjs.com 2017.)

3.2.1 Routing

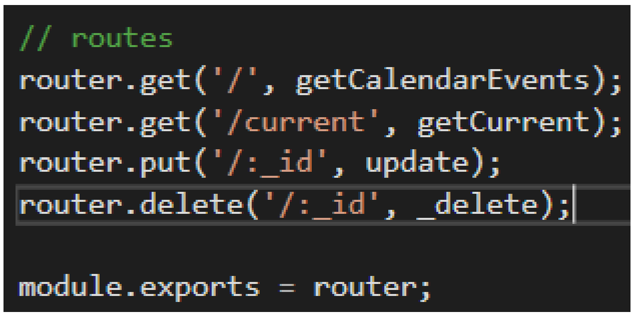
Express provides simplified routing for the server environment by providing a middleware object (visible in figure 2) that intercepts oncoming HTTP requests and routes them to specific parts of the code based on the request URL and the used HTTP request. The router middleware object can also perform various functions related to responding to the client such as sending error codes. (expressjs.com 2017.)



```
var config = require('config.json');
var express = require('express');
var router = express.Router();
```

Figure 2. Importing express into a component

Express streamlines routing by structuring it into functions that trigger based on the request URL and HTTP request, eliminating the need for complex selection trees that parse requests, showcased in figure 3. (expressjs.com 2017.)

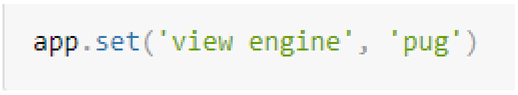


```
// routes
router.get('/', getCalendarEvents);
router.get('/current', getCurrent);
router.put('/:_id', update);
router.delete('/:_id', _delete);

module.exports = router;
```

Figure 3. Adding routes into the router object

3.2.2 Template engines

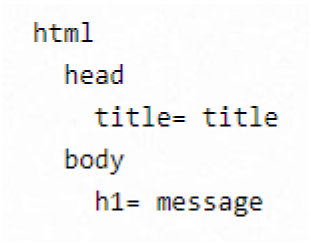
Express can use variety of different template engines to serve clients with web pages with content customized for each specific client. The express application generator installs the Jade template engine by default. Other relevant template engines include engines such as Ejs, Pug and Mustache. To utilize a template engine the developer must first load the template engine module internally into the express middleware, showcased in figure 4. (expressjs.com, 2017.)



```
app.set('view engine', 'pug')
```

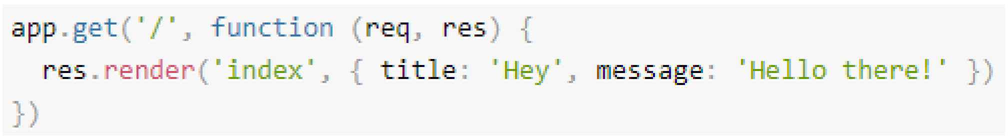
Figure 4. Pug template engine is loaded into Express.

Template engines utilize template files that render out into html and JavaScript, varying based on the parameters given with the Express middleware route leading to the template file, visible in figure 7. A simple pug template file is visible in figure 5. (expressjs.com, 2017.)



```
html
  head
    title= title
  body
    h1= message
```

Figure 5. A simple pug template engine template.



```
app.get('/', function (req, res) {
  res.render('index', { title: 'Hey', message: 'Hello there!' })
})
```

Figure 6. A template engine is used to render a simple hello web-page as a response.

Template engines can be used to render complex web pages in the backend with JavaScript included, potentially speeding up the browsing experience of users by pre-rendering web pages on the server and sending them to the client, saving the user´s web-browser from rendering the web-page locally. (expressjs.com, 2017.)

3.2.3 Database integration

The developer can integrate a variety of different databases into the Express application by installing one of the many database drivers available for Express. Many of the database drivers can be found from Node Package Manager. In general, the process of installing a database driver is a matter of loading it from the Node Package Manager and importing and initializing the driver´s middleware object with the right connection information to connect into one of the databases compatible with the said driver. (expressjs.com 2017.)

Some database drivers enable additional features performed in Express, such as the Mongoose, which can perform SQL-like data modeling and querying on MongoDB, a noSQL database. (expressjs.com 2017.)

3.3 MongoDB

MongoDB is an open source NoSQL database. NoSQL databases store their data into collections of documents unlike more traditional SQL databases that store their data into data tables. NoSQL databases are more versatile in prototyping new applications as a NoSQL document can contain varying number of multiple kinds of variables, whereas a SQL data table would need to be modified to accept different kinds or number of variables from the application. SQL tables, due to their structural uniformity are simpler to query and thus more complex queries into the database can be performed easier. SQL language may be preferred if the software development project will require complex queries. MongoDB can be granted a virtual rigid data structure similar to a SQL database using Mongoose, an Express MongoDB database driver. Mongoose enables the developer to define the format in which the documents must be submitted to the database, emulating a SQL database. Mongoose enables the application to query the database in a more powerful, SQL-like format. MongoDB is best used for purposes like big data and content management and delivery, mobile and social infrastructure, user data management and data hub functions. (Tutorialspoint.com 2018.)

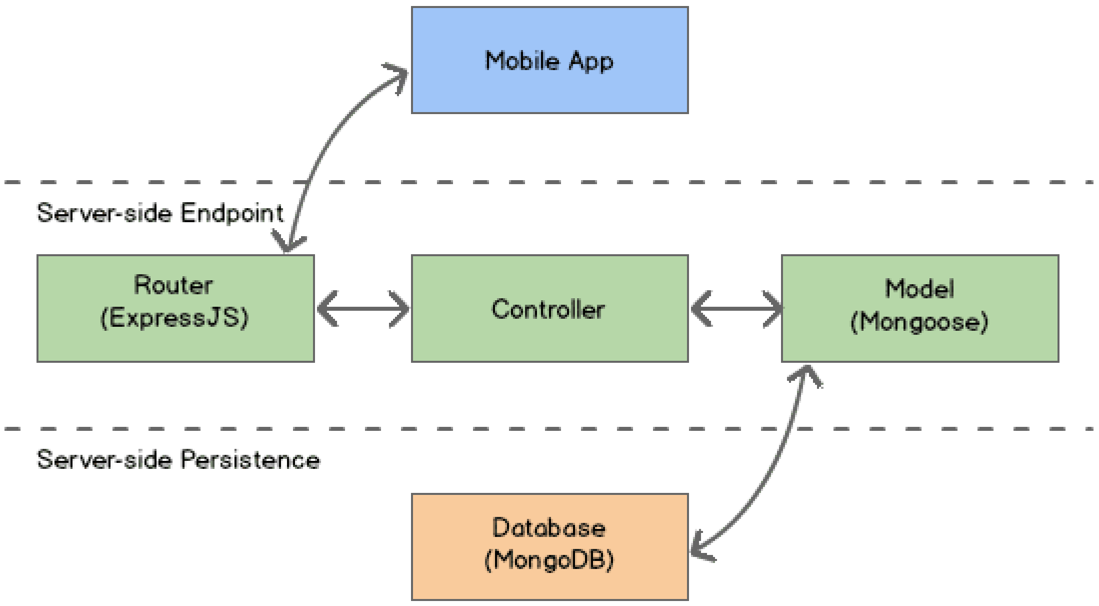


Figure 7. Simplified model of the data flow between a mobile app, express.js back-end and the MongoDB database.

MongoDB was originally released February 2009 as a open source project by the company 10gen. 10gen changed its name into MongoDB Inc. in August 2013. (bytescout.com 2018.)

3.3.1 File storage and querying

MongoDB stores its data into collections of documents. Documents are collections of variables of various datatypes allowed in MongoDB. Collections exist to organize documents into groups and insulate them from other documents to avoid naming overlap. MongoDB is able to accept various datatypes from multiple different languages, provided a driver capable of translating the datatypes is available to the language. The database driver translates the objects found in the language of the language of the runtime environment into BSON, Binary JavaScript Object Notation, which the database stores into its documents. Similarly, the database driver translates the BSON it receives from the database back into variables and objects usable by the language the driver the supports. (Mongodb.com 2018.)

When querying data, the developer specifies the collection to be used and queries the documents based on the values of the variables found inside the documents. For example, if the developer wants to find a certain document they can search for it with its identification code that MongoDB generates automatically when creating new documents or querying for some other variable found in the documents in the collection. Using an empty find() method on a MongoDB driver such as mongoose returns the whole collection that is being queried in the response received from the database. (Mongodb.com 2018.)

MongoDB stores its data in form of BSON, Binary JavaScript Object notation, a superset of JSON that is designed to be more efficient with data storage and faster to scan when doing queries. (Mongodb.com 2018.)

3.3.2 Data redundancy, scalability and sharding

MongoDB document collections can be sharded across multiple different databases to provide scalability and data redundancy. Individual documents are not dependent on one another and thus a single collection could be stored across multiple different machines if required. Sharding enables MongoDB to scale in a streamlined and simple manner as the developer only needs to add and setup more hardware for MongoDB to use to acquire increased throughput and data storage. (Mongodb.com 2018.)

MongoDB can have multiple copies of the same document across multiple different machines, enabling robust data redundancy. MongoDB automatically handles version control for redundant documents and solves any version conflicts by saving and spreading changes across the documents and delaying queries as necessary to avoid conflicting documents saves and queries into yet to be updated documents. By having duplicates across many servers, a robust redundancy and increased throughput can be achieved as multiple servers can host the same document and supply it to the clients to together. (Mongodb.com 2018.)

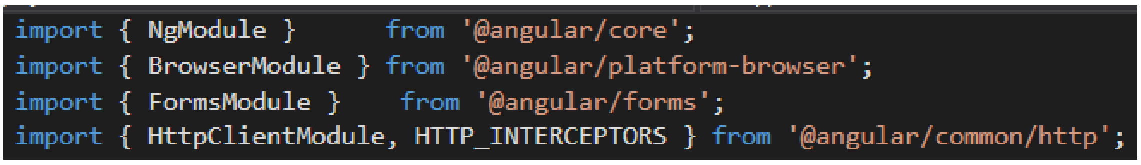
3.4 Angular

Angular is a client-side single-page web-application creation framework that offers a variety of tools that simplify and speed up single-page web-application and dynamic web-content creation. Using angular the developer is able to avoid writing most of the boilerplate code that would be included in a web-application, such as direct DOM manipulation. Angular offers tools such as data-binding, view routing, view trees, dependency injection and routing. (angular.io 2018.)

Angular was originally released by Miško Hevery in 2009 as Angular.JS but was completely rewritten in 2016 by the original developer team and named as Angular 2, eliminating most of the backwards compatibility with the earlier Angular.js. This required most programs written using Angular.js to be rewritten for Angular 2 or continued developing using Angular.js. Angular is developed and maintained by Google and a number of individual developers and companies. (Brandrick 2017.)

The toolset provided by Angular makes it possible for most of the business logic of a web-application to be moved to the client-side, leaving the back-end mostly for shuttling data between the client and the database. This serves to reduce the hosting cost of the application by moving most of the processor workload to the client. This works especially well with Node.js as Node.js is at its strength fulfilling massive amounts requests concurrently but suffers from CPU intensive tasks. (angular.io 2018.)

Angular projects consist of NgModules, packages of angular code files. NgModules are usually intended to serve some specific function, such as a specific workflow or a use case, such as a shopping cart in a webshop. All Angular projects possess at least on NgModule, a root NgModule that bootstraps the Angular framework, augmenting the runtime environment to with the ability to parse and process Angular mark-up. NgModules contain components, services and other code relevant to the function to the NgModule and are usually built to work independently, possessing all the code they need to fulfill their intended purpose. NgModules are enabled in the application by importing them into the root module. NgModules can import and export features from and to one another in a similar way to JavaScript. (angular.io 2018.)



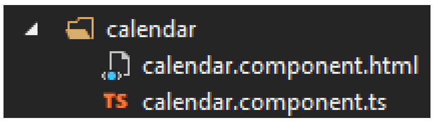
```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule }    from '@angular/forms';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
```

Figure 8. NgModules imported into the root NgModule used in the showcase application of this thesis.

NgModules mainly consist of component and service files. Components handle business logic of the application. In combination with template files they create user interface elements called views. Services are packages of code that contain reusable and common code. Components can subscribe into these services to gain their functionality. Services exist to reduce the amount of rewritten code in a application, serving the same, commonly used code to multiple different components. (angular.io 2018.)

NgModules are split into five general categories: domain feature modules, routed feature modules, routing modules, service feature modules and widget feature modules. Domain feature modules for example serve user experiences dedicated towards a particular application domain, such as placing an order in a web store. (angular.io 2018.)

Components consist of a HTML template file with Angular directives and a typescript controller file that handles the business logic used in servicing the HTML file with UI data. The component and template file of view can be seen in figure 9. The rendered product of a component is a UI element called a view. (angular.io 2018.)



```
calendar
    calendar.component.html
    calendar.component.ts
```

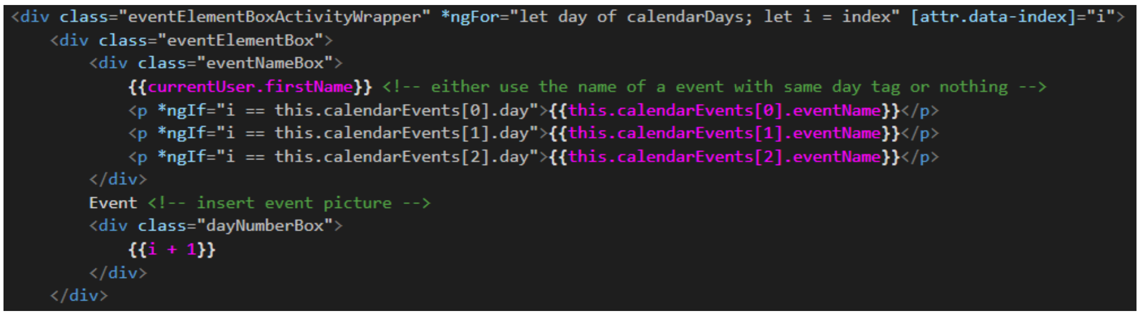
Figure 9. The component and template files of a calendar view.

Angular uses TypeScript, a strongly typed superset of JavaScript. Strong typing reduces the risk of accidental data type conversion in the code and eases testability as any attempts at assigning a value of wrong data type to a differently typed variable causes an error. (Typescriptlang.org 2018.)

Angular competes with react and vue in the single page application market. React is a user interface library with smaller initial library size compared to Angular but lacks many of the tools Angular provides. React can be programmed in JSX, which is a superset of JavaScript that uses HTML-like mark-up to make React user interface element creation code more readable. React was released in March 2013. Vue is the newcomer to the field and is the fastest growing of the three in terms of users, released in 2014 by a ex google employee, vue is described as a “intuitive, fast and composable MVVM” and is used for example by Alibaba and gitlab. (Neuhaus 2017.)

3.4.1 Angular directives and metadata

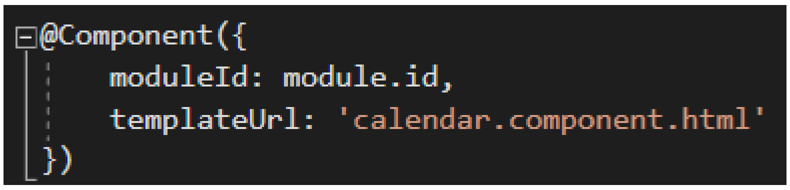
Angular provides the developer with a set of Angular mark-up that is in the compilation process of the Angular application. Directives are code that are used in the template file of a view to repeat, show, hide or remove DOM elements based on the data model found in the component file. Angular directives can be seen used in figure 10. (angular.io 2018.)



```
<div class="eventElementBoxActivityWrapper" *ngFor="let day of calendarDays; let i = index" [attr.data-index]="i">
    <div class="eventElementBox">
        <div class="eventNameBox">
            {{currentUser.firstName}} <!-- either use the name of a event with same day tag or nothing -->
            <p *ngIf="i == this.calendarEvents[0].day">{{this.calendarEvents[0].eventName}}</p>
            <p *ngIf="i == this.calendarEvents[1].day">{{this.calendarEvents[1].eventName}}</p>
            <p *ngIf="i == this.calendarEvents[2].day">{{this.calendarEvents[2].eventName}}</p>
        </div>
        Event <!-- insert event picture -->
        <div class="dayNumberBox">
            {{i + 1}}
        </div>
    </div>
```

Figure 10. A ngFor directive being used to iterate through an array of calendar day objects.

Angular metadata decorators are int turn used communicate with the Angular framework the role and various other relevant information of a individual class of TypeScript in the Angular project. Such as marking a component class as a component by using the @Component decorator, seen in figure 11. (angular.io 2018.)



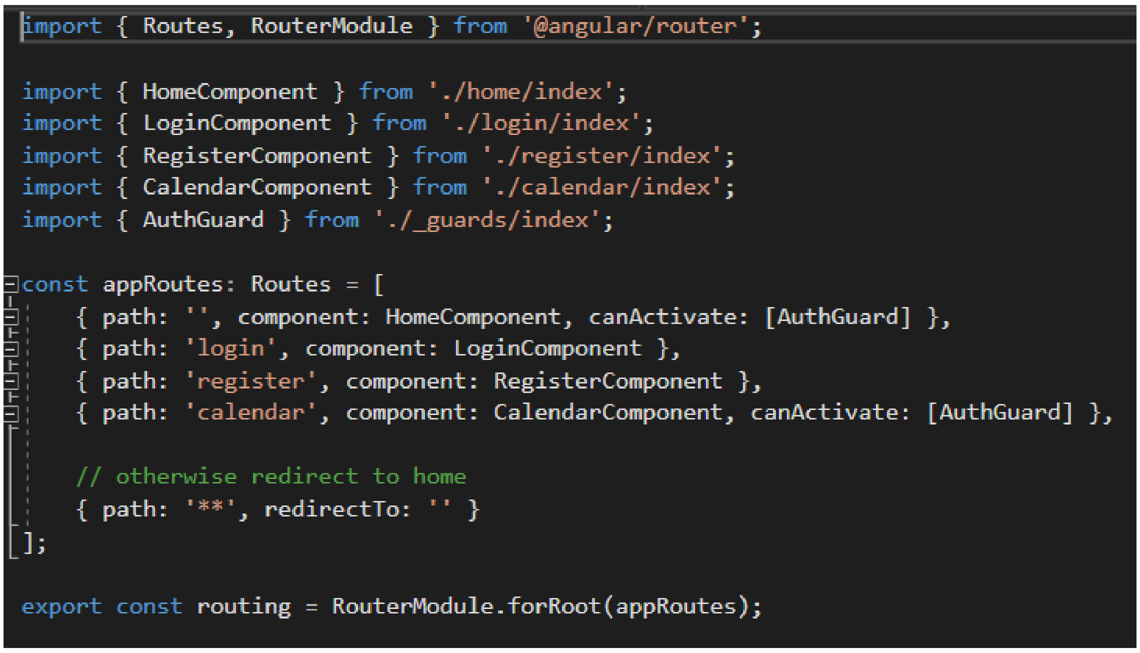
```
@Component({
    moduleId: module.id,
    templateUrl: 'calendar.component.html'
})
```

Figure 11. Component metadata marked with a decorator.

With the use of Angular decorators, Angular is able to connect template files and component files and form a common compilation context between them, enabling the template file to utilize objects and variables found in the component file. (angular.io 2018.)

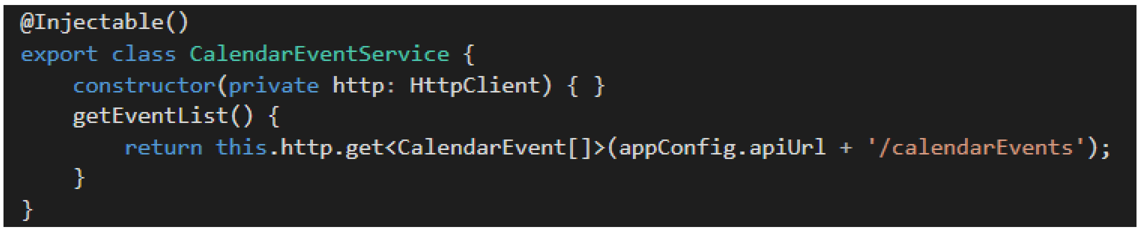
3.4.2 Routing

Angular core libraries provide a router NgModule that the user can import into their own project, enabling navigation from one view into another using a mechanism that is similar to that found in modern internet browsers. The router transitions are saved into the browser page history, giving the application a fast and user-friendly method for navigating from view to view. The router is enabled by importing it into a NgModule and supplying it with an array of URL sub-addresses and components, into which the URL sub-address will route the user. An Angular route list can be seen in figure 12. (angular.io 2018.)

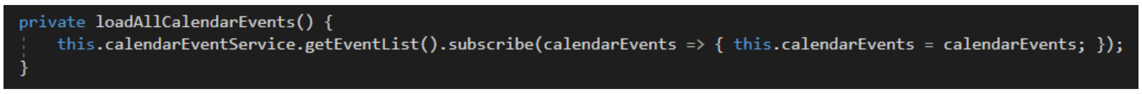

```
import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/index';
import { LoginComponent } from './login/index';
import { RegisterComponent } from './register/index';
import { CalendarComponent } from './calendar/index';
import { AuthGuard } from './_guards/index';

const appRoutes: Routes = [
    { path: '', component: HomeComponent, canActivate: [AuthGuard] },
    { path: 'login', component: LoginComponent },
    { path: 'register', component: RegisterComponent },
    { path: 'calendar', component: CalendarComponent, canActivate: [AuthGuard] },

    // otherwise redirect to home
    { path: '**', redirectTo: '' }
];

export const routing = RouterModule.forRoot(appRoutes);
```
Figure 12. Importing the router NgModule and supplying it with routes.

3.4.3 Observables

Observables is a type of event listening mechanism where the developer assigns a function to be observable and creates an observer object for handling the events the observable function releases. The observable starts sending the observer updates about itself after the observer initiates the observation using a subscribe function built into the Observable class. An example of an Observable http request can be seen in the figure 13. The arrow brackets around the CalendarEvent array identify the getEventList method as an observable to Angular. (angular.io 2018.)


```
@Injectable()
export class CalendarEventService {
    constructor(private http: HttpClient) { }
    getEventList() {
        return this.http.get<CalendarEvent[]>(appConfig.apiUrl + '/calendarEvents');
    }
}
```
Figure 13. HTTPclient request that uses a Observable request, distinguishable by the < > brackets used with the get method.

The observer method is a method that executes when observables sends an update. The observer method receives the data from observable function and executes its own internal code, using the received data it’s the data model of the component of the observer method. Example of this can be seen in Figure 14, where the observer function subscribes into the service seen in figure 13. (angular.io 2018.)

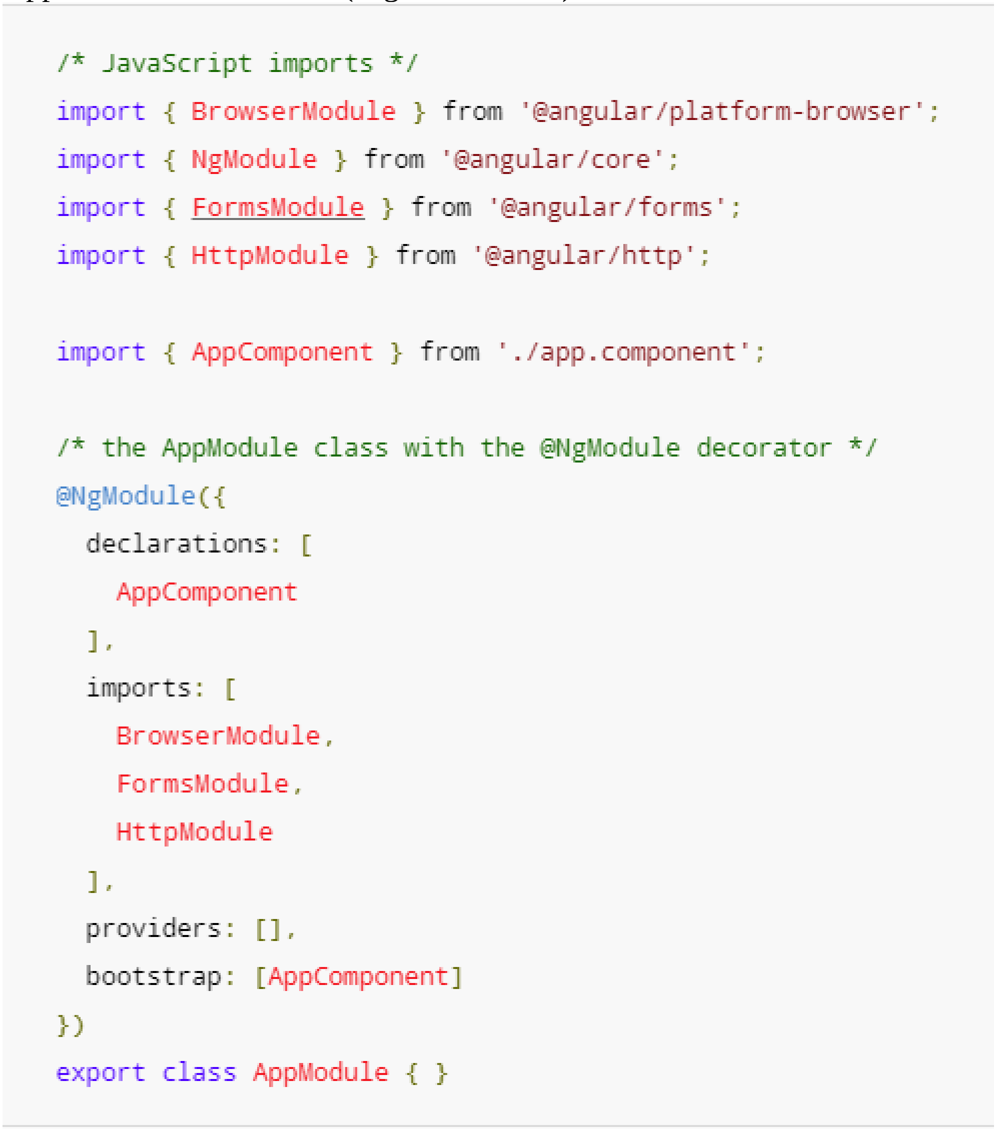


```
private loadAllCalendarEvents() {
    this.calendarEventService.getEventList().subscribe(calendarEvents => { this.calendarEvents = calendarEvents; });
}
```

Figure 14. Observer function subscribing into a observable function.

3.4.4 Installing and bootstrapping Angular

All Angular applications contain a NgModule for bootstrapping Angular to enable its functionality. By convention, this NgModule is usually called AppModule. If you use the Angular CLI to generate an app, the default AppModule is as follows: (angular.io 2018.)



```
/* JavaScript imports */
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';

/* the AppModule class with the @NgModule decorator */
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Figure 15. Angular CLI generated root NgModule that bootstraps Angular.

4 CALENDAR APP

The practical section of this thesis is a simple showcase application that showcases the important functions of the different components of MEAN-stack. The application is a simple web-application with a user interface that draws event data from the stack´s MongoDB database and locates it into a grid built using Angular directives. Due to the time constraints the practical example had to be minimized and some features of the different components of MEAN-stack had to be left out.

4.1 Used libraries and software

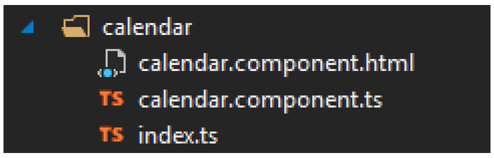
I programmed the application using Microsoft Visual Studio Community 2017 version 15.3.5 and the different used frameworks of MEAN-stack were their newest versions as of 8.10.2018.

4.2 Internal workings of the calendar app

The application I made is a simple technology showcase stack, containing an account creation system, login, and a calendar grid rendered using Angular directives and populated with events requested from the Express back-end running one a Node.js server runtime, which in turn queries the data from the MongoDB database.

4.2.1 Angular

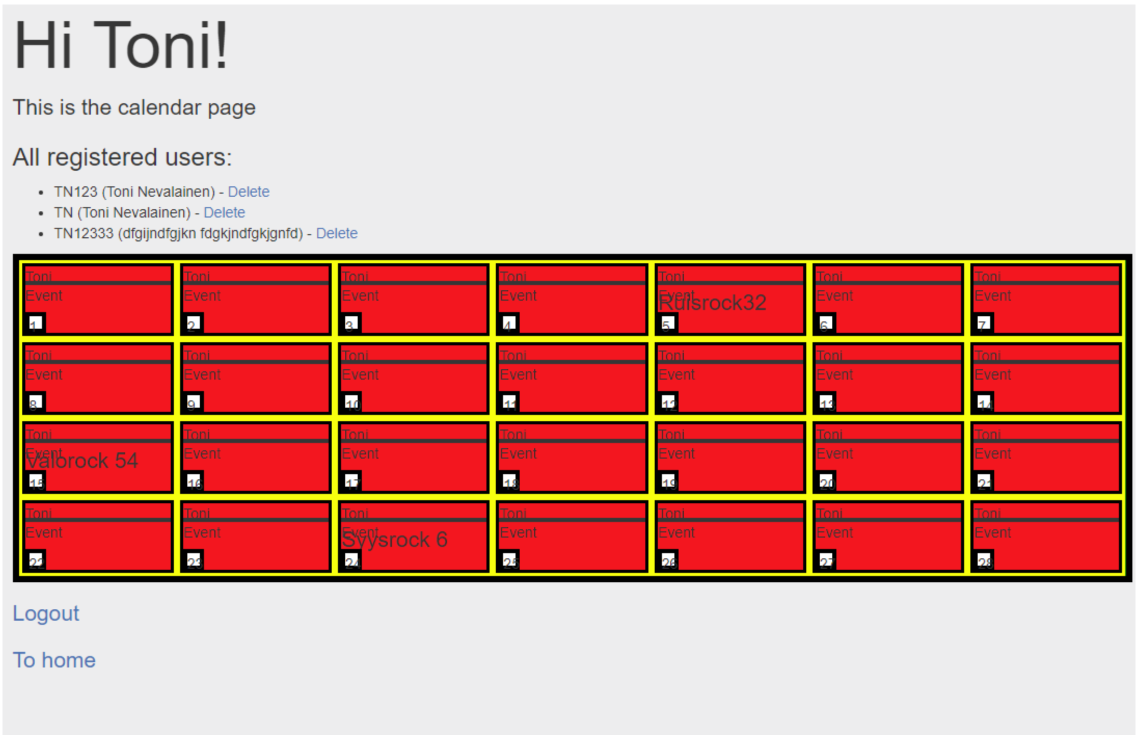
With Angular I created a login, a register and a calendar system. The application is split to views, which are further divided into component and template files, visible in figure 15 as the .ts and .html files respectively.



```
calendar
    calendar.component.html
    calendar.component.ts
    index.ts
```

Figure 16. Model and template files of the calendar view, the html and ts file respectively.

The main application screen consists of a list of all registered users, a calendar grid that is rendered using Angular directives and couple of Angular routing links, visible in figure 16.


Hi Toni!

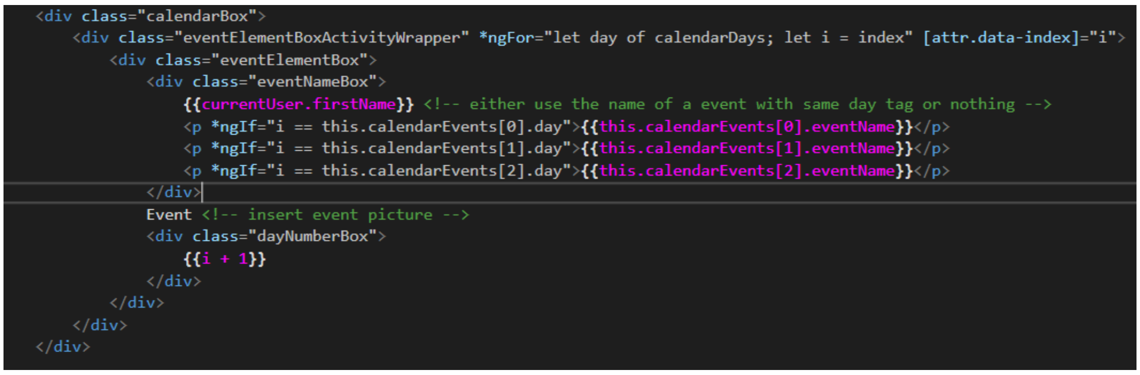
This is the calendar page

All registered users:

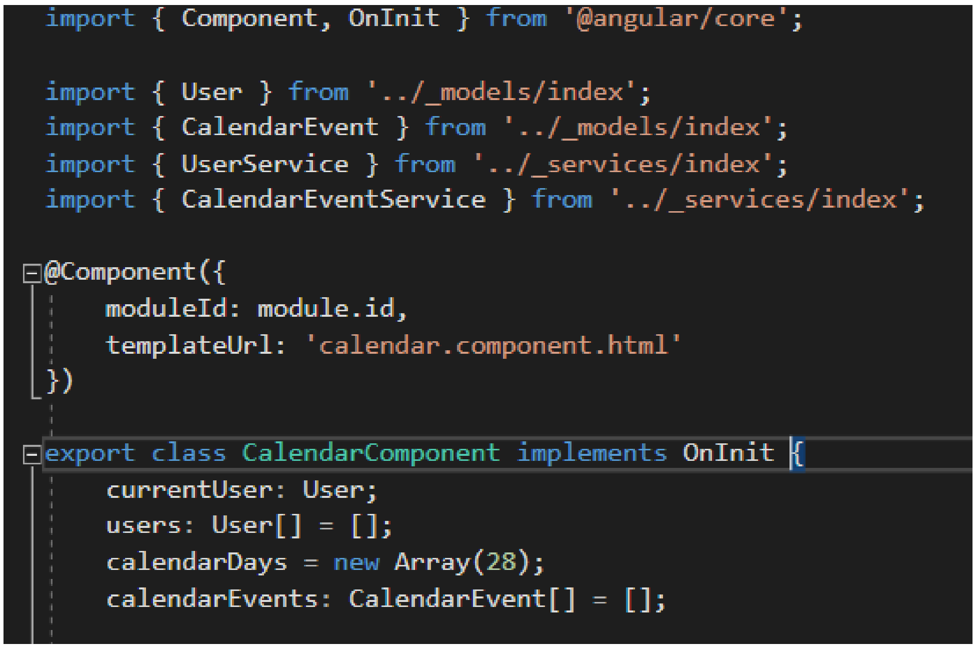
- TN123 (Toni Nevalainen) - Delete
- TN (Toni Nevalainen) - Delete
- TN12333 (dfgijndfgjkn fdgkjndfgkjgnfd) - Delete

Figure 17. The user interface of the application.

The grid is rendered using Angular directives, using the *NgFor directive to repeat a ready-made HTML element 28 times in a divider, ordered into a colored grid using CSS mark-up, visible in figure 17. Class attribute of the different elements are used to assign CSS directives to the different elements.


```
<div class="calendarBox">
    <div class="eventElementBoxActivityWrapper" *ngFor="let day of calendarDays; let i = index" [attr.data-index]="i">
        <div class="eventElementBox">
            <div class="eventNameBox">
                {{currentUser.firstName}} <!-- either use the name of a event with same day tag or nothing -->
                <p *ngIf="i == this.calendarEvents[0].day">{{this.calendarEvents[0].eventName}}</p>
                <p *ngIf="i == this.calendarEvents[1].day">{{this.calendarEvents[1].eventName}}</p>
                <p *ngIf="i == this.calendarEvents[2].day">{{this.calendarEvents[2].eventName}}</p>
            </div>
            Event <!-- insert event picture -->
            <div class="dayNumberBox">
                {{i + 1}}
            </div>
        </div>
    </div>
</div>
```
Figure 18. The HTML code with ngFor Angular mark-up used to render the calendar view in the front page.

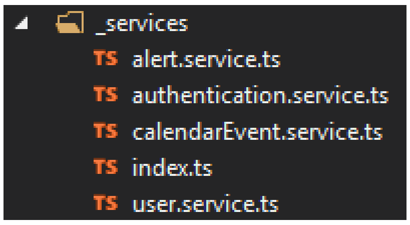
The business logic of the view is contained in a separate component file that contains the variables used for template HTML file. One such variable is the calendarDays variable that the template´s ngFor directive uses for repeating the HTML and Angular directives under it for 28 times, visible in figure 17. The component file subscribes into services and calls the functions necessary for aggregating the data used in the template, visible in figure 18.



```
import { Component, OnInit } from '@angular/core';

import { User } from '../_models/index';
import { CalendarEvent } from '../_models/index';
import { UserService } from '../_services/index';
import { CalendarEventService } from '../_services/index';

@Component({
    moduleId: module.id,
    templateUrl: 'calendar.component.html'
})

export class CalendarComponent implements OnInit {
    currentUser: User;
    users: User[] = [];
    calendarDays = new Array(28);
    calendarEvents: CalendarEvent[] = [];
```

Figure 19. Angular component file.

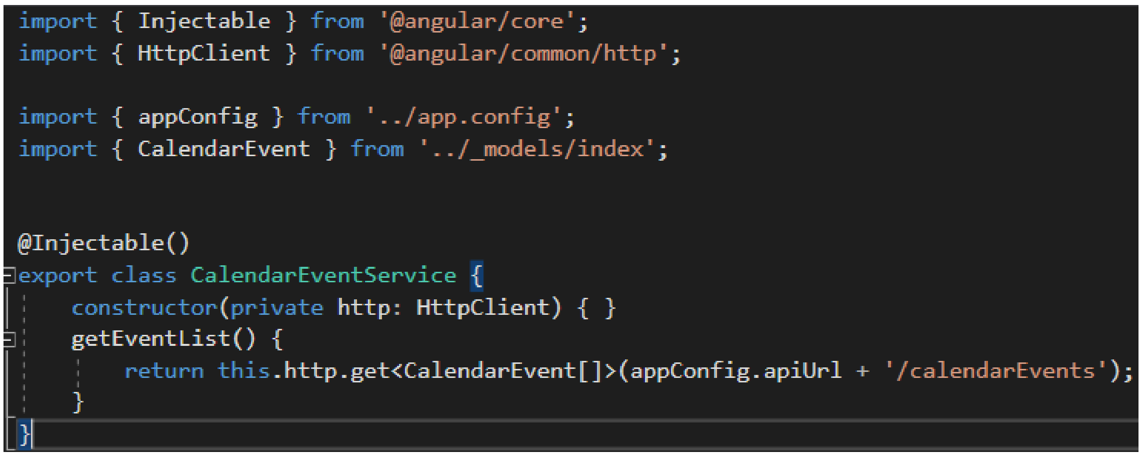
Angular has a service provider system where service files are collected under service providers into which components can subscribe into. Subscribed components gain access to all the service files under the provider without needing to import them individually, service files used in the showcase application can be seen figure 19.



```
_services
    alert.service.ts
    authentication.service.ts
    calendarEvent.service.ts
    index.ts
    user.service.ts
```

Figure 20. Angular service files.

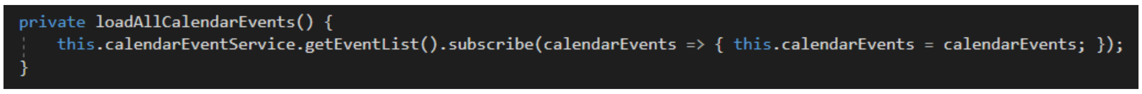
The calendarEventService service visible in figure 19 is responsible for requesting event data from the server. The service sends out an observable request, distinguishable < > brackets visible in the getEventList() function visible in the figure 20.



```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { appConfig } from '../app.config';
import { CalendarEvent } from '../_models/index';


@Injectable()
export class CalendarEventService {
    constructor(private http: HttpClient) { }
    getEventList() {
        return this.http.get<CalendarEvent[]>(appConfig.apiUrl + '/calendarEvents');
    }
}
```

Figure 20. The getEventList function requests an array of event objects from the Express back-end.

Once it receives the response from the server, it releases an event that trigger a different method, visible in figure 21 that saves the received calendar event data into the calendarEvents variable found in the calendar view component file.

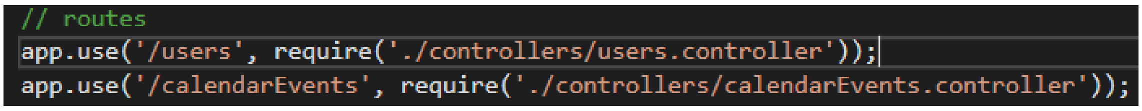


```
private loadAllCalendarEvents() {
    this.calendarEventService.getEventList().subscribe(calendarEvents => { this.calendarEvents = calendarEvents; });
}
```

Figure 21. Observer function subscribing into a observable function.

4.2.2 Express

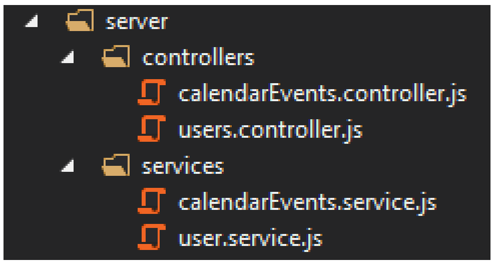
The request arriving in the Express back-end is routed into a controller file responsible for handling data related calendar events based on the request´s URL address and HTTP method, visible in figure 21.



```
// routes
app.use('/users', require('./controllers/users.controller'));
app.use('/calendarEvents', require('./controllers/calendarEvents.controller'));
```

Figure 22. Application level routing.

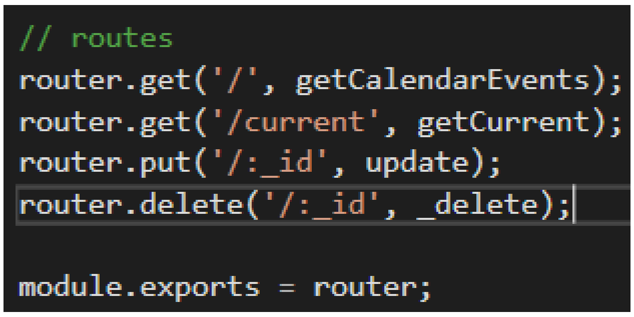
In the calendar event request´s case the used routing actives on any HTTP method and with /calendarEvents sub-address, seen in figure 21. Using app.use method instead of some other method such as app.get makes the route HTTP request type agnostic. The request gets routed into calendarEvents.controller controller, visible in figure 22, based on the request URL.



```
server
    controllers
        calendarEvents.controller.js
        users.controller.js
    services
        calendarEvents.service.js
        user.service.js
```

Figure 23. Express controllers and services.

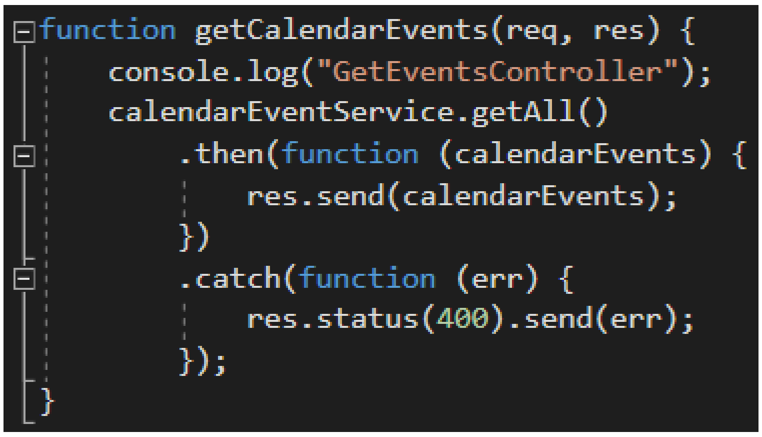
Express back-end is split into controllers and services, services provide functions to the controllers and controllers serve client requests primarily. The calendarEventsController further routes the request onto functions inside itself based on further sub-addresses on the request. The getCalendarEvents function is called as the request used in the example lacks any additional beyond the initial /calendarEvents sub-address, visible in figure 23. The the route activated in the showcase is the topmost visible in figure 23. Unlike the routing before calendarEventsController, the routing inside requires a get HTTP request to be used in order for the request to be routed to the function used in the showcase.



```
// routes
router.get('/', getCalendarEvents);
router.get('/current', getCurrent);
router.put('/:_id', update);
router.delete('/:_id', _delete);

module.exports = router;
```

Figure 24. Router level routing.

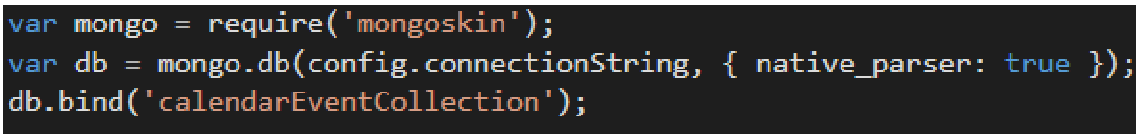
The getCalendarEvents function calls calendarEventService:s getAll function to get all events in the document collection used in the application, visible in figure 24.



```
function getCalendarEvents(req, res) {
    console.log("GetEventsController");
    calendarEventService.getAll()
        .then(function (calendarEvents) {
            res.send(calendarEvents);
        })
        .catch(function (err) {
            res.status(400).send(err);
        });
}
```

Figure 25. getCalendarEvents function in the calendarEventsController.

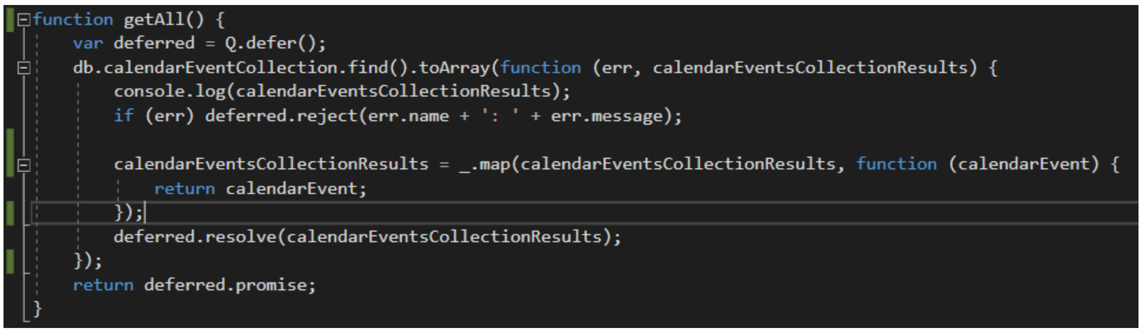
A MongoDB driver is imported for interaction with the MongoDB database and a connection string is supplied to it from the configuration files of the application project, visible in figure 25. A .bind() method is used to establish a connection into the database. The connection string (‘calendarEventCollection’) is used to pick the document collection to be targeted in the queries to the database.



```
var mongo = require('mongoskin');
var db = mongo.db(config.connectionString, { native_parser: true });
db.bind('calendarEventCollection');
```

Figure 26. Initializing the mongoskin MongoDB database driver.

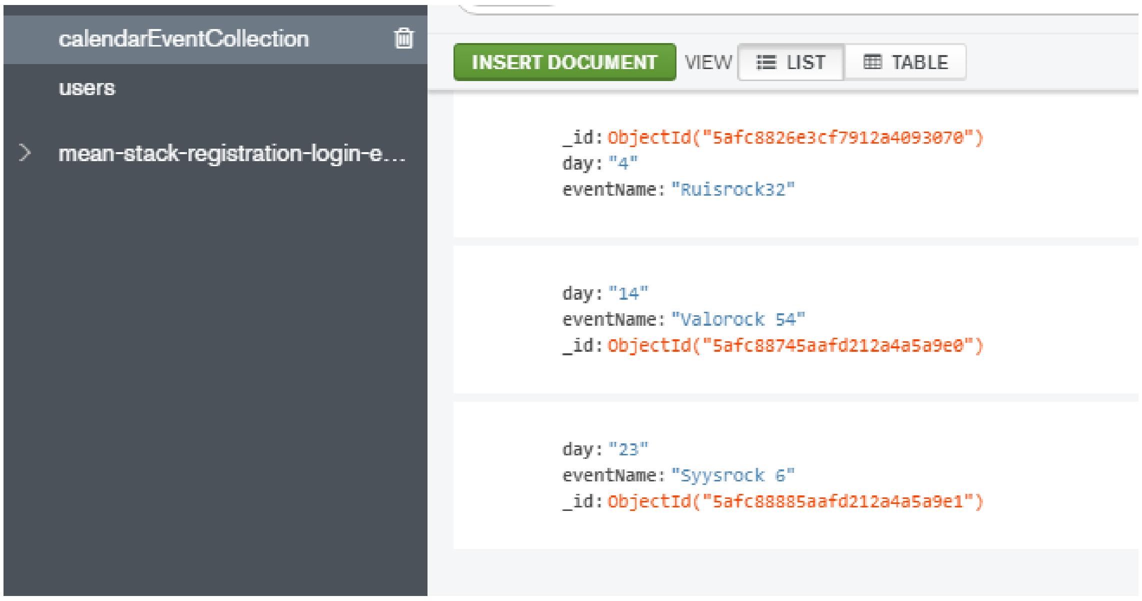
The calendarEventService´s getAll function, visible in figure 25, uses the database driver to request all the documents under the calendarEventCollection collection and returns them as an array for the calendarEventsController controller, which in turn returns it to the front-end Angular observable request. From the observable the data moves to the component file of the calendar view and the user interface is rendered using the template file based on the data found in the component file.



```
function getAll() {
    var deferred = Q.defer();
    db.calendarEventCollection.find().toArray(function (err, calendarEventsCollectionResults) {
        console.log(calendarEventsCollectionResults);
        if (err) deferred.reject(err.name + ': ' + err.message);

        calendarEventsCollectionResults = _.map(calendarEventsCollectionResults, function (calendarEvent) {
            return calendarEvent;
        });
        deferred.resolve(calendarEventsCollectionResults);
    });
    return deferred.promise;
}
```

Figure 27. getAll function of the calendarEventService; queries event data from the database.

The calendar events seen in the Angular application user interface can be seen in figure 27. The used software is MongoDB compass, a MongoDB GUI client.



```
_id: ObjectId("5afc8826e3cf7912a4093070")
day: "4"
eventName: "Ruisrock32"

day: "14"
eventName: "Valorock 54"
_id: ObjectId("5afc88745aafd212a4a5a9e0")

day: "23"
eventName: "Syysrock 6"
_id: ObjectId("5afc88885aafd212a4a5a9e1")
```

Figure 28. View of MongoDB compass MongoDB GUI client, shows the event data documents used in the thesis application.

4.3 Programming issues and solutions during making of the thesis

Unexpected depth and variety of the stack combined with bad project management led to a need to a shrink the scale of the programming included in the showcase application significantly. The reduced time available for the application was made up my simplifying the application significantly while keeping onto showcasing MEAN-stack´s basic functions.

4.4 Potential future improvements for the app

The application could be expanded to allow for saving user specific events and later completely rewritten to serve open-data geo-located events to mobile users.

5 SYNOPSIS

MEAN-stack is a popular and versatile software stack adept at supporting single-page web-applications. It offers a versatile albeit challenging toolset for the developer but once the developer acquires proficiency using MEAN-stack, the development is faster than using other comparable web-development software stacks. MEAN-stack is best used in projects that require rapid development and prototyping, having a concise syntax with minimum bootstrap code and having utility features such as being programmed throughout using JavaScript.

Being programmed throughout with JavaScript lowers the total required expertise in a software development project, and therefore reduces the total hiring costs for the project. Being single language also enables easier communication between different developers working on different parts of a MEAN-stack software development project and enables the different components to utilize the same JSON serving application programming interfaces, reducing the amount of required middleware for transforming and transmitting data between the different components of the stack. Due to using JavaScript, a popular programming language, hiring into a MEAN-stack software development project is easier because of the number of JavaScript skilled developers available in the workforce.

The thesis suffered from time constraints and bad project management. The showcase application was relatively simple to create using the knowledge acquired during the research and writing phases. The more advanced features of MEAN took significantly longer to research as the scope of the underlying theory was larger than initially considered. This also caused the scope of the thesis to be shrunk to its final form, which showcases MEAN-stack in an introductory format.

REFERENCES

C. Wodehouse, 2018, upwork.com, What is a Framework,
https://www.upwork.com/hiring/development/understanding-software-frameworks/
Read 17.5.2018

C. Wodehouse, 2018, upwork.com, Choosing the Right Software Stack,
https://www.upwork.com/hiring/development/choosing-the-right-software-stack-for-your-website/
Read 17.5.2018

G. Nemeth, 2018, risingstack.com, History of Node.js on a Timeline
https://blog.risingstack.com/history-of-node-js/
Read 17.5.2018

P. Patel, 2018, freecodecamp.org, What exactly is Node.js?)
https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5
Read 17.5.2018

T. Capan, 2013, toptal.com, Why The Hell Would I Use Node.js? A Case-by-Case Tutorial,
https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js
Read 17.5.2018

M. Heller, 2017, infoworld.com, What is Node.js? The JavaScript runtime explained,
https://www.infoworld.com/article/3210589/node-js/what-is-nodejs-javascript-runtime-explained.html
Read 17.5.2018

S. Holmes, 2016, Getting MEAN with Mongo, Express, Angular and Node, p.8

Node.js.org, 2018, nodejs.org, The Node.js Event Loop, Timers, and process.nextTick(),
https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/
Read 17.5.2018

C. Ramirez, 2017, codeburst.io, Multi Core NodeJS App, is it possible in a single thread framework?,
https://codeburst.io/multi-core-nodejs-app-is-it-possible-in-a-single-thread-framework-fe4feaf67f90
Read 17.5.2018

M. Hablich, 2017, github.com, What is V8?,
https://github.com/v8/v8/wiki
Read 17.5.2018

npmjs.com, 2018, npmjs.com, Node Package Manager,
https://www.npmjs.com/

Read 17.5.2018

express.js.com, 2017, Fast, unopinionated, minimalist web framework for Node.js,
https://expressjs.com/
Read 17.5.2018

D. Tzur, 2016, thefullstack.xyz, The Unbelievable History of the Express JavaScript Framework,
https://thefullstack.xyz/history-express-javascript-framework/
Read 17.5.2018

T. Hallam, 2018, mubaloo.com, Best Practices: Deploying Node.js Applications,
https://mubaloo.com/best-practices-deploying-node-js-applications/
Read 17.5.2018

Tutorialspoint.com, 2018, tutorialspoint.com, MongoDB – Advantages,
https://www.tutorialspoint.com/mongodb/mongodb_advantages.htm
Read 17.5.2018

Mongod.com, 2018, mongodb.com, BSON Types,
https://docs.mongodb.com/manual/reference/bson-types/
Read 17.5.2018

Mongod.com, 2018, mongodb.com, What is MongoDB?,
https://www.mongodb.com/mongodb-architecture
Read 17.5.2018

Mongod.com, 2018, mongodb.com, Sharding,
https://docs.mongodb.com/manual/sharding/
Read 17.5.2018

Mongod.com, 2018, mongodb.com, Data Modeling Introduction,
https://docs.mongodb.com/manual/core/data-modeling-introduction/
Read 17.5.2018

Mongod.com, 2018, mongodb.com, Query Documents,
https://docs.mongodb.com/manual/tutorial/query-documents/
Read 17.5.2018

Bytescout.com, 2018, bytescout.com, MongoDB History and Advantages,
https://bytescout.com/blog/2014/09/mongodb-history-and-advantages.html
Read 17.5.2018

C. Brandrick, 2017, medium.com, Angular 1.0 Turns Five Years Old,
https://medium.com/dailyjs/angular-1-0-turns-five-years-old-4d7108a5e412
Read 17.5.2018

W. Smith, 2014, University of Michigan, The C Programming Language,

http://groups.engin.umd.umich.edu/CIS/course.des/cis400/c/hworld.html
Read 17.5.2018

Typescriptlang.org, 2018, typescriptlang.org, Documentation,
https://www.typescriptlang.org/docs/home.html
Read 17.5.2018

Angular.io, 2018, angular.io, Introduction to modules,
https://angular.io/guide/architecture-modules
Read 17.5.2018

Angular.io, 2018, angular.io, Introduction to modules,
https://angular.io/guide/testing
Read 17.5.2018

J. Neuhaus, 2017, medium.com, Angular vs. React vs. Vue: A 2017 comparison,
https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176
Read 17.5.2018

J. Williams, 2017, medium.com, What is The MEAN STACK?,
https://medium.com/@jeremyvsjeremy/what-is-the-mean-stack-9d11ae2cd384
Read 17.5.2018

Y. Tsuneo, 2015, paiza.io, Building full-stack web service - MEAN stack development(1),
http://engineering.paiza.io/entry/2015/07/08/153011
Read 17.5.2018