Portfolio Subproject 4

And

Reflective Synopsis

Group Nr. 7

Hunor Vadasz-Perhat

Jonas Michelsen

Casper Bertel Rye Hintze

Yohannes Kidane Kifle

Github URL: https://github.com/JonasMGit/RAWDATA_Portfolio_Final

Character count total: 47,758

Character count for subproject 4: 17,688

Character count for final report: 30,070

A. Application Design

A.1

The frontend architecture for our single page application is build on the Model-View-ViewModel pattern. This is achieved by making the three following layers: Data Access, The Presentation and Business logic or better known as model, view and view model (MVVM). The model makes callbacks to certain urls paths which calls the backend api, that returns a json answer. This results in a single page application because the browser does not get redirect to a new url but the model handles the api and gets the necessary information.

The view model changes the data inside of the model when an action is made inside view and functions like a middle man for model and view by binding the data between them. The action made inside the view databind register it and starts the process between the layers which in the the end changes the view (MDSN Microsoft, 2013).

In our project we have a main view called index.html that contains the main layout and a piece of code which connects it to the components views and depending on the components view fills the view with different content. This code piece can be seen below:

```
<div class="col-12" data-bind="component: { name: selectedComponent, params: selectedParams}"></div>
```

Our view models can be seen in components folder with the .js extension, furthermore we have a main view model in the app folder called questions.js (Appendix 1) . The model from the MVVM is the ds.js file in the which can be seen in the folder Services. This model gets the data from the backend.

To further explain this pattern for a better understanding we have include this quote:

> "The view classes have no idea that the model classes exist, while the ViewModel and model are unaware of the view. In fact, the model doesn't have any idea about ViewModel and view exists." (MDSN Microsoft, 2013)

A.2.b

In this project we have used the Mostly Fluid which is a multi-device pattern. Mostly Fluid pattern works by dropping elements down below the others in the same order as if you read left to right when the page is resized. (Stackoverflow, 2016). In the bootstrap it works by the grid systems breakpoints, when the grid gets to a certain width it breaks and in this case drop the columns (Developers Google, 2018). We used Mostly Fluid pattern instead of Column Drop and Layout Shifter. The difference between Column Drop and Mostly Fluid is that Mostly Fluid drops the column in the same order as seen from left to right, where the Column Drop will change the order to show content or navigation on the top of the page.

The layout shifter changes the layout of the page depending on the size of the screen, it could be that the navigation menu is on the side of the page when its a medium or large screen, but if the screen becomes small the navigation menu will be on the top. This makes it harder to create because you have to make it work with two layouts but you also have to maintain both layouts, which each have their limitations. We decide go with Mostly Fluid because it is easier to implement but also the content of the application wouldn't necessarily benefit from a certain order and therefore there weren't much reason to implement a different pattern then Mostly Fluid.

A.3

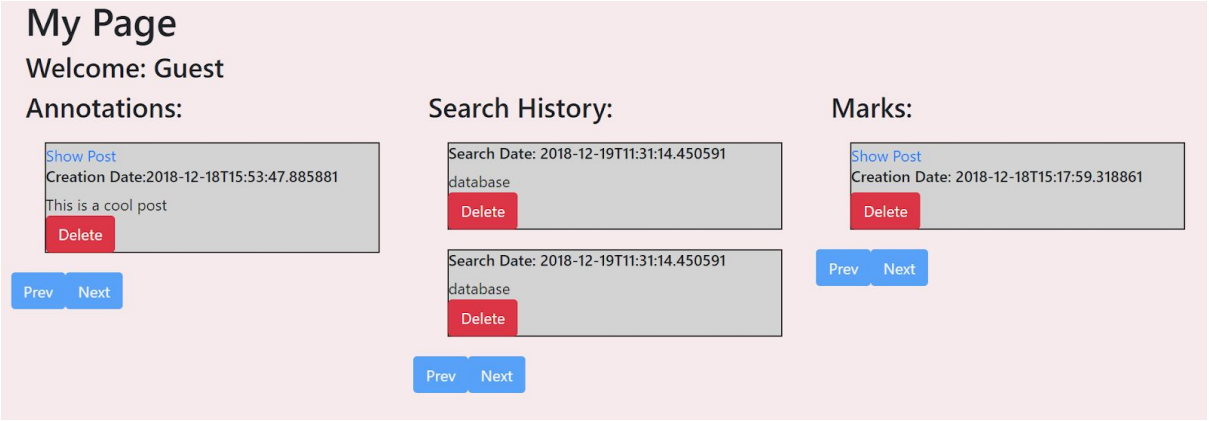
Initial design can be seen in Appendix 2, there have been changes to the overall design mainly because everything wasn't planned down to every little detail in the first place but used as a tool to get better common understanding in the group. The use of bootstraps grid system and Mostly Fluid pattern can be seen in view questionView.html and the userView.html. In the questionView.html the score for the

question, comments and answers uses this pattern. A snippet of the code can be viewed below:

```
<div class="row">
    <div class="col-10">
        <h3>Date: <span data-bind="text: creationDate"></span></h3>
    </div>
    <div class="align-content-end; col-md-2 col-sm-4">
        <h3>Score: <span data-bind="text: score"></span></h3>
    </div>
</div>
```

In the third , we can see that the column change size depending on the width of the browser. The browser width is defined via the md(medium) and sm(small). On a medium size screen the score is in the right side of the question. For comments and answers, if the screen gets smaller, the score drops below the creationDate because of the score column increase in size to 4 and the creationDate is 10 columns. This is a part of the bootstrap grid system and the grid system is only 12 columns wide by default. Therefore if the column size exceed the column size of the grid system it will drop down.

In the userView.html this component's view can the user see his/hers annotations, marks and search history as seen below:



Here we have implemented the Mostly Fluid pattern again. The code snippet below is applied to annotations, marks and search history.

```
<div class="col-sm-12 col-md-6 col-lg-4">
```

In the previous picture the view was viewed as a large(lg) window so their column were 4 columns wide. If the window reaches the md breakpoint the column

Marks will get drop down below Search History and Annotations and if it reaches the sm breakpoint Search History will drop below Annotations but above Marks.

B. The Data Access

In this section we will explain the data access layer in the frontend which is a layer between the backend and business logic of the frontend and it is used to access data from the Web Service by exposing a set of functions. Revealing module design pattern is used and it is one of the design patterns in javascript.

Revealing module pattern is used to preserve encapsulation and reveal certain variables and methods returned at the end of the module. This helps to see which variables and methods are accessible publicly and this makes the code more readable and consistent (Addy Osmani, 2017). We created two javascript files under a services folder in the project, named as dataservice and postman.

Dataservice is used to access data from the backend by exposing a number of functions like getPosts, getPost, searchPosts, getUser, getAnnotation and getMark. We used $.getJson() and $.ajax() jquery AJAX functions to get Api data from the backend. AJAX stands for asynchronous javascript and xml and is used to interact with the backend which helps to update a web pages without reloading. $.getJSON() loads json data from the backend using the get method of http request and it has two parameters the url (the url in which the request is sent) and callback function. Callback function is a function that is passed as a parameter from one function to other javascript function and it is executed inside the other function. When callback function is passed as an argument, the definition of the function is passed as a parameter without execution. (We are not executing the function in the parameter the execution will happen in the other function) (Richard Bovell, 2013)

Example: As it is shown in the screenshot below the callback function is passed as a parameter to the other function and executed in the second function.

Callback function:

```
$.getJSON("api/questions/name/" + terms, callback)
```

Called function by callback function:

```
ds.searchPosts(terms, function (data)
```

The $.ajax() sends asynchronous http request similar to $.getJSON but it provides us option parameters. Data type specifies the type of response data, in our case we use JSON, as type of http method we use and more. The axax method can also specify the type of request like a Get, or a Post, which we use to create and delete elements from the database.All the functions can be seen in appendix 3.

Postman is a javascript file in our project which contains the subscribe and publish patterns which is a way to create a loosely coupled system where components have a single responsibility which means components will only care about their responsibility not other components. Subscribe function has two parameters the event and callback. Callback function calls when a notification is sent and the event receives a notification from a publisher. Publish function has also two parameters the event which loops through subscribers to find the same event and callback function that sends data to the callback function in subscriber. For example in our project we used subscribe and publish within different components. To have clear view on subscribe and publish we added a screenshot which helps to see what a subscribe and publish is in javascript.

In general, as it is shown in the two first screenshots below, a publisher sends an event to notify a subscriber with data which is originally a string but in this case converted to json by JSON.stringfy() (a function which converts a string to json). The foreach loop in publish loops through all the events to find the same name of the event in subscribe, when the event matches with the event in the subscribe then notification will send to the event in the subscribe to notify the subscriber and send json data.

Subscribe

```
var subscribers = [];
var debug = true;
var subscribe = function (event, callback) {
    var subscriber = { event, callback };
    if (debug) console.log("subscribe: " + JSON.stringify(subscriber));

    subscribers.push(subscriber);
```

Publish

```
var publish = function (event, data) {
    if (debug) console.log("publish: " + JSON.stringify({ event, data }));
    subscribers.forEach(function (s) {
        if (event === s.event) s.callback(data);
    });
};
```

Unsubscribe

If a subscriber no longer wants to be notified by a publisher then a subscriber can be unsubscribed and the screenshot below shows unsubscription.

```
return function () {
    subscribers = subscribers.filter(function (e) {
        return e !== subscriber;
    });
};
```

C. Business Logic

C.1

The view model is a part of the MVVC design pattern, that handles the business logic of the application. It is the job of the view model to update the view when changes happen to the model data, and it also handles changes to the model when events occur in the application(e.g saving, deleting and navigation). The view model also makes it possible to separate different responsibility areas of the logic behind an application so it simpler to work on these different parts (MDSN Microsoft,; 2013) .

We have four separate view models to handle the different “pages”, which are questionList, question, user and word cloud; of which each have a distinct role. The questionList handles searching for posts and listing them, as well as supplying the view with clickable links to post that can be updated via search input through the view. Question handles displaying individual posts, with its comments and answers, as well as creating annotations and marks to the post, assigned to a specific user. User contains information about a specific user, such as a user name, and the marks, search history and annotations. It was also intended that the user could update these items, but due to time constraints was not implemented in full. Finally, the word cloud which generates a word cloud from an user defined search term.

C.2

Components further organise the UI code and viewModel code by dividing it into smaller chunks and allowing the parts to be loaded asynchronously, which improves runtime performance (Components and Custom Elements - Overview; unknown date). Components are usually comprised of one view model and a corresponding view, as is the case in our application. We implement components for the view models mentioned in C1 as well as their corresponding views. You can see the file layout for the components in appendix (Appendix:1). In the javascript file we switch components by setting the component to another module whenever a button is pressed that changes view. An example being to view a question when clicking on a link in questionList.

showPost in questionList:

```
var showPost = function (post) {

    postman.publish("selectedComponent", {
        item: "question", params: {
            link: post.link,
            back: searchVal(),
            userId: id,
            postId: post.id
        }
    });
};
```

The postman service sets the selected component to question, which will load the question component and send relevant information to that view model

C.3

RequireJS is a module loader for javascript files that enables files to be loaded asynchronously, which means that the application only loads the modules that are needed for the user. To set up RequireJS we configured our various modules by supplying the folder paths of where they are located:

Require config:

```
require.config({
    baseUrl: "js",
    paths: {
        jquery: "lib/jQuery/dist/jquery.min",
        knockout: "lib/knockout/dist/knockout.debug",
        dataService: "services/ds",
        jqcloud: 'lib/jqcloud2/dist/jqcloud',
        text: "lib/text/text",
        postman: 'services/postman',
        bootstrap: "lib/bootstrap/dist/js/bootstrap.bundle"

    },
    shim: {
        // set default deps
        'jqcloud': ['jquery'],
        'bootstrap': ['jquery']
    }
});
```

So, the configure loads all of the libraries and services needed by the application (jquery, knockout, bootstrap...), so we don't need to load them using scripts.

We then use define() to define a module, which essentially is a function with dependencies to different libraries or services that handles the logic of the application(RequireJS - Quick Guide; unknown date). Each of our components are encapsulated within these define methods and return various functions created within the module. For example our question list component has dependencies to knockout, dataService and postman, meaning they use these libraries/services.

```
define(['knockout', 'dataService', 'postman'], function (ko, ds, postman) {
```

Once all the modules are defined, we use the require() function to load the modules used by the application. We use require to load the components we defined, where we also use knockout to register the components and use require to fetch the viewmodel and view template.

Load Components:

```
// load components
require(['knockout'], function (ko) {

    ko.components.register("question-list",
        {
            viewModel: { require: 'components/QuestionList/questionList' },
            template: { require: 'text!components/QuestionList/questionListView.html' }
        });

    ko.components.register("question",
        {
            viewModel: { require: 'components/Question/question' },
            template: { require: 'text!components/Question/questionView.html' }
        });
```

In connection to knockout we use one more require to apply all of the data-bindings, as HTML doesn't know what a data-bind is so knockout needs to be activated for a view model to take effect. In our app we activate knockout for the main, or parent, view model app/questions, which in turn activates it for all the components under this view model(Observables: Apply Bindings; unknown date).

C.4

We use various data-bindings in our application, but the most important one is the component data-binding. The component binding injects a specified component into an element, and can also pass parameters to it (The "component" binding; unknown date). We use both of these in our index.html file, where we can change the component by using the postman service to set the selected component to another when certain events are triggered.

```
<div class="col-12" data-bind="component: { name: selectedComponent, params: selectedParams}"></div>
```

We also use data-binding on button tags and textInput, which are then bound to a function and observable respectively. Take this example from questionList: questionListView:

```
<div class="col-sm-2 offset-2" style="margin-top: 20px; align-items-end;">
    <input data-bind="textInput : searchVal" />
    <button type="button" class="btn btn-success" data-bind="click: searchPost">Search</button>
</div>
```

Here the input tag is data-bound to a variable called searchVal, which is defined in the model, and will take the string that is input in the textbox and populate the value. The button is bound to the searchPost function that, on button click will make a $.getJson request to the server using the input parameter searchVal() as the search term.

SearchVal:

```
var searchVal = ko.observable(params.back);
```

searchPost:

```
var searchPost = function () {

    $.ajax({
        type: 'POST',
        url: 'api/searchhistory/add/',
        // The key needs to match your method's input parameter (case-sensitive).
        data: JSON.stringify({ search: searchVal(), userId: id }),
        contentType: 'application/json',

    });

    getSearch(searchVal())

};
```

UI:

C.5

The navigation bar uses data-binding to change the component to display it on the application. The navigation has various menu items which are iterated through and displayed. These menu items are defined in the questions javascript file and these navbar items are bound to the a function called changeMenu and it is

activated when one of these links are clicked. There is also a data-bind to a span tag within this changeMenu that populate the function with the name of whatever is clicked.

Navigation bar:

```
<ul class="navbar-nav mr-auto" data-bind="foreach: menuItems">
    <li class="nav-item" data-bind="css: $parent.isActive($data)">
        <a class="nav-link" href="#" data-bind="click: $parent.changeMenu">
            <span data-bind="text: name"></span>
        </a>
    </li>
</ul>
```

changeMenu in js:

```
var changeMenu = function (menu) {
    selectedMenu(menu);
    selectedComponent(menu.component);
};
```

Here the changeMenu function has its menu parameter populated with whichever name value clicked on and sets the selectedComponent() to the component value of the menuItem selected, which will change the view to the particular component. For example if the name is ‘My Page’ it will set the component to ‘userPage’.

NavBar:

menuItems:

```
var menuItems = [
    { name: 'Home', component: 'question-list' },
    { name: 'Cloud', component: 'cloud' },
    {name: 'My Page', component: 'userPage'}
];
```

D. Presentation

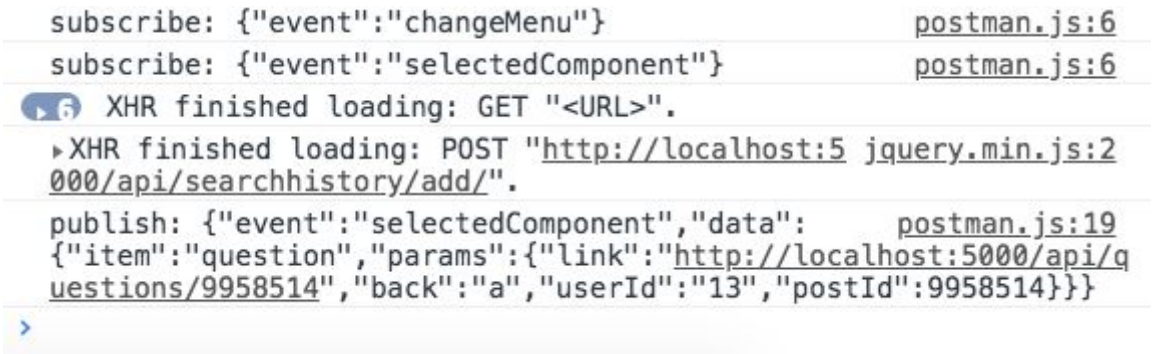
D.1

The single-page application as a solution for a web application is a practical one since it interacts with the user by dynamically rewriting the current page(Paul Sherman, 2018). The page will not be reloaded at any point in the process. The advantage is that this approach avoids the interruption of the user experience - since it makes the application to behave more like a desktop application.
When using the Knockout.js in the index.html file the following code makes it sure that the SOVA becomes an SPA.

```
<div class="col-12" data-bind="component: { name: selectedComponent, params: selectedParams}"></div>
```

In the browser after looking at the inspect element option we see the following the code:

After clicking on a given search result the following code is created:

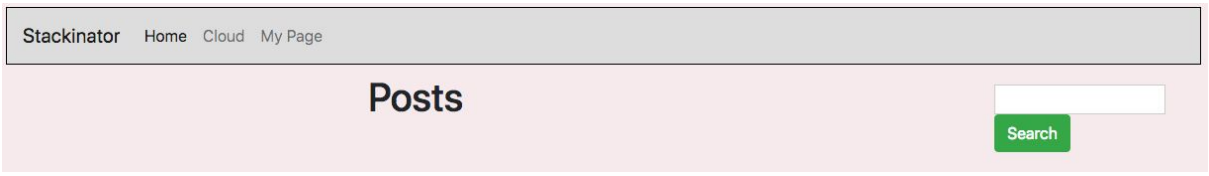


The content in the index.html file was filled up the the content that the particular component returned.

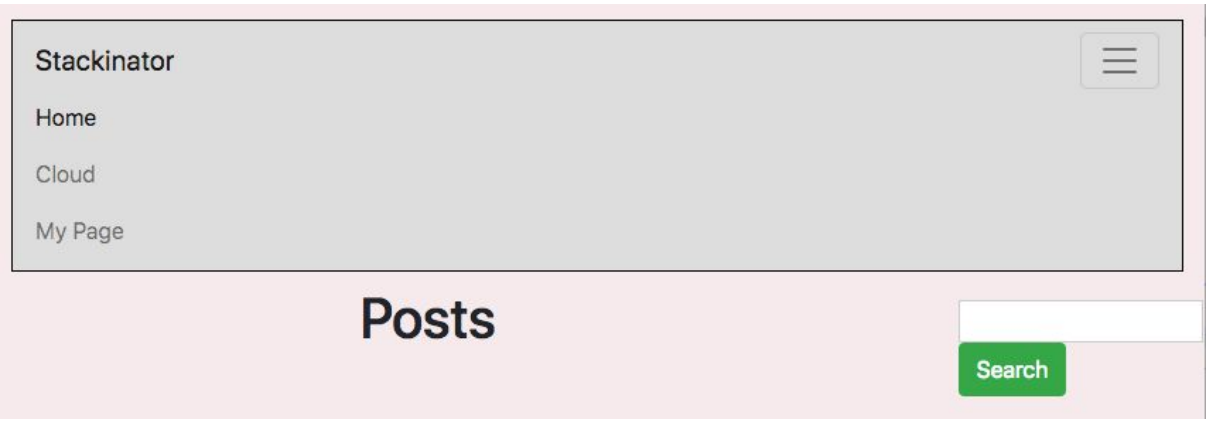
```
<!doctype html>
<html>
▸ <head>…</head>
▾ <body> == $0
  ▾ <div class="container-fluid">
    ▾ <div>
      ▸ <nav class="navbar navbar-expand-lg navbar-light mt-3" style=
      "width: 100%;">…</nav>
      </div>
    ▾ <div class="col-12" data-bind="component: { name:
      selectedComponent, params: selectedParams}">
      ▾ <div class="question">
        ▸ <div data-bind="with: currentPostComment">…</div>
        ▸ <div data-bind="with: currentPostAnswer">…</div>
        </div>
      </div>
    ▸ <div class="footer">…</div>
      <script data-main="js/main.js" src="js/lib/requirejs/
      require.js"></script>
    </div>
  </body>
</html>
```

The content of the particular component will be displayed the index.html file. This structure of the front-end architecture will make sure that there is a dynamic communication with the server behind the scenes.

D.2

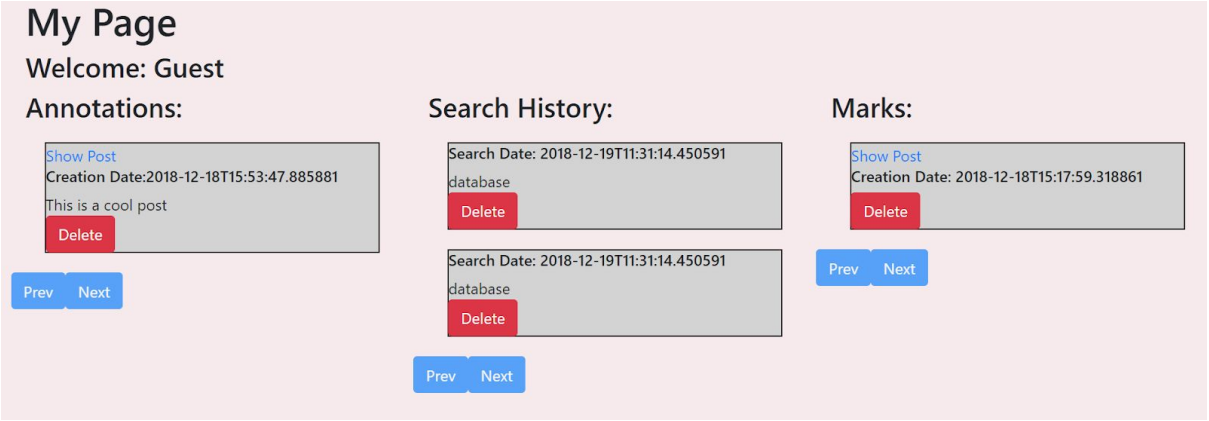


After changing the size of the browser the navigation bar switch to a burger menu.

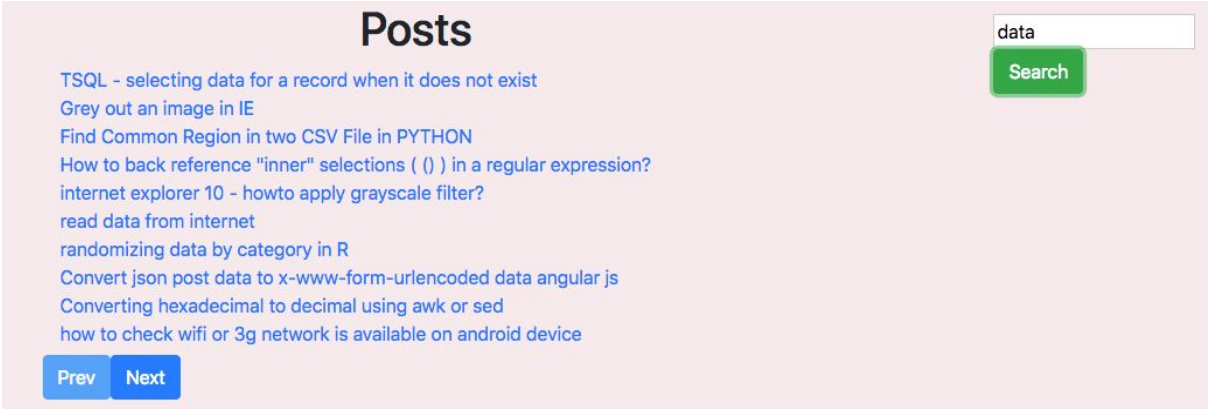


Home is the landing page where the user finds himself at when the page is loaded. Here is the search functionality where a list of posts can be displayed as links with a pagination functionality. The cloud link will lead to page where data visualisation takes place. After typing in a particular term a word cloud data visualisation will be generated.

My page link will lead us to an individual user and various data that is associated with that particular user. The various data would be annotations, search history and marks.



D.3



We include pagination for all the listed information on the web page. We use it for listing posts, as well as for the annotation, search history and mark information in the userPage view. Pagination improves the user experience as the user doesn't have to scroll down a long page of listed information, but can instead click forward and back to find info.

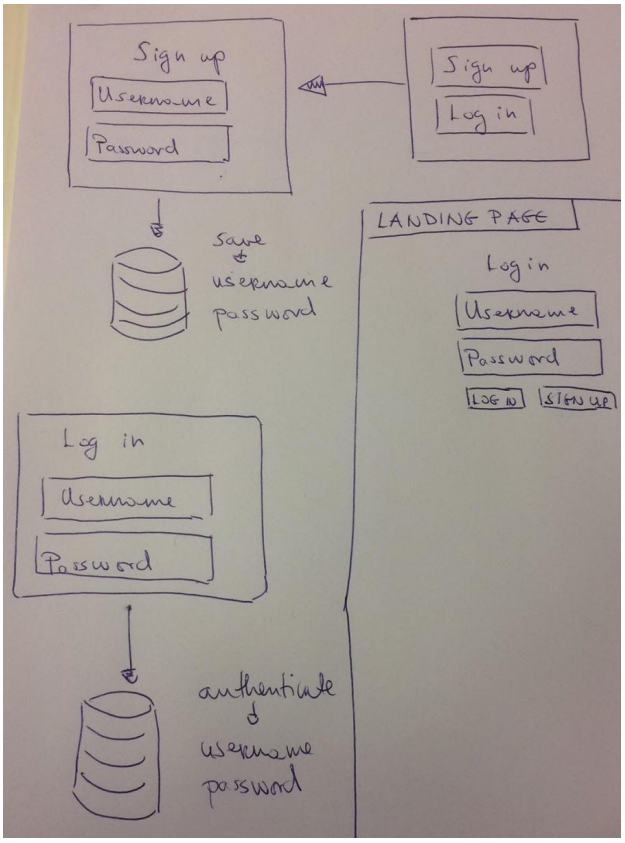
D.4

The group did not have the time to implement a form that would provide validation. The general logic for validation with a form was discussed in the group:

- For the creation of the user there are two necessary pieces of information, the username and the password which would be saved in the database

- After the user creation the user would login where the given username and password would be compared to the username and password that is stored in the database
- The username would be displayed on the landing page

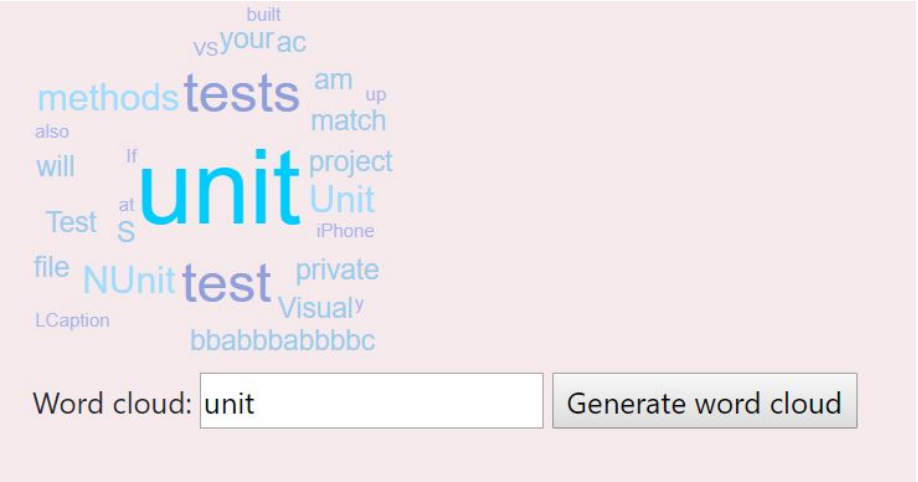
The following sketch shows a possible solution:



D.5

The frontend must include visualizations for the features developed in section 3, i.e. word clouds, association graphs and ranked search results based on the advanced information retrieval

After clicking on Cloud link we find ourselves on the page where we can look for a term and a word cloud data visualisation would be generated.



The result of the search is based upon the following query:

```
create or replace function  wrdCloud (term text)  RETURNS table
("text" text, weight double precision)
AS
$$
select word, sum(weigth) from tfidfi,
(select id , sum(tfidf) weigth  from
(select distinct id, tfidf from tfidfi where word = term ) t1 group by
id) t2
where tfidfi.id = t2.id and  tfidf > 0.02
group by word   order by sum(weigth) desc limit 20
$$
language sql;
```

In our dataservice file we specified the number of term we can use for which is one:

```
public List<WordCloud> GetWordCloud(string text)
{
    using (var db = new SOVAContext())
    {
        var cloud = db.WordClouds.FromSql("select * from wrdCloud({0})",
text);

        return cloud.ToList();
    }
}
```

Reflective Synopsis

1. Introduction

For this reflective synopsis we will make a short summary of our application, how it is in its current state as well as some information about using the application. We have also prepared some topics for discussion highlighting the theory behind it and how it has affected our project work. The topics were written individually and cover: testing, security, information retrieval and design patterns. To round the synopsis up we will have a discussion on the reflections we have made for improvements in the application. These reflections include discussions on our subprojects, implementing security and validation for users, inconsistencies in our program code and better UI and navigation.

2. Portfolio project summary

Important User Guide:

Before starting this summary, we would like to make a small guide for a user (or rather for the examiners) of the application, as we are missing a vital part for the easy use of the program. We assume that the user table in the database is empty for a new user, so when the user starts the SOVA application they need to click on the button “Generate user” to create a user. Then in the questionList.js and user.js files they need to write the id for that user that is in the database to the global variables named “userid” in both files. Then the user should be able to make marks,

annotations and search histories for that user. This is not an ideal solution and a better one should be implemented.

id	username	creationdate	password
50	Guest	2018-12-18 14:50:	Guest

Project Summary

The SOVA application as it is now includes concepts that we have learned throughout the course of the semester. The database holds all of the information needed in the application, though there is some information that is present in the database that we do not implement in the application, but can be used in future work. The application uses the repository pattern to interact with the database, being able to perform database operations quickly and sending this information to the client side. It implements useful information retrieval, the user can search for terms and display relevant information using the tf-idf information retrieval technique.

The user can create, read and update marks and annotations to specific posts, as well as it being tied to a specific user. Data visualization can be viewed in the form of a word cloud, that weighs terms that appear in the same documents as a search term. The user can navigate the different views with a navigation bar and that application uses a responsive multi-device layout pattern.

The main drawback of the application is the security aspect as there is no form of authentication or login, the user is hard coded in the backend which is not ideal for applications in the real world. The front-end is also not using, or displaying, all the information we can access in the backend. We will discuss these things in the discussion chapter.

3. Topics

Software Testing

Software testing is a widespread phenomenon in the computer science industry. The reason for this is that testing software can find bugs, errors or other unintended behavior and furthermore it can prove that code works as intended. If these bugs or unintended behavior are not found it can be costly for organisation, corporations or even nations(Guru99, Unknown Date; Wikipedia, 2018). But what do these so called test, test for? There are multiple ways to test software. Testing can be split up into different methods: White-, Gray- and Black Box Testing. White Box Testing is a method also referred as Clear Box Testing because the test person most commonly is the software developer of the program and therefore can see what is happening inside the box while the test is running on for e.x. the structure, design or implementation of the software (Software Testing Fundamentals A, Unknown Date). Black Box Testing is the test where users are unaware of what is happening inside the box because it is often not a developer who is testing but a user and therefore only a look on how the software behaves depending on the input and output and in that way errors and bugs are found(Software Testing Fundamentals B, Unknown Date). Gray Box testing is a combination of White- and Black Box Testing and the code is partly known by the user (Software Testing Fundamentals C, Unknown Date). We will look deeper into White Box Testing because it's what we have used for the project with the Unit Test. White Box Testing covers a wide variety of testing methods, and of those tests we will look at Unit Test and Integration Test.

For our project we used XUnit framework to create our Unit Test. This framework provides functions to assert the code which can confirm the code works as intended. Unit Test is useful for validating if a unit works properly. A unit is defined as the smallest individual testable part which could be a method or a function etc. Unit Tests helps with maintaining code, makes sure that the new code or the altered code does not change behavior or break the current code and furthermore makes sure that the code works as intended in the first place (Software Testing Fundamentals D, Unknown Date).

When Unit Tests are combined and tested as a group it is called Integration Test. Integration Test is used to find errors and bugs when there is interaction between layers or different parts of the system talking together. This form for testing is on an abstract level and higher than Unit Test and would be the next logical step

for this project to implement to ensure the behavior of the program is under control and maintained properly in further development(Software Testing Fundamentals E, Unknown Date). Commonly, multiple test methods are used when developing a software application and the type method depends on who is testing and which phrase the development is in. Overall, costly mistakes can be found with the help of a good test environment but they can not ensure that they won't happen.

Design Patterns

Application development is a complex process that requires careful design and organised cooperation between its developers. As novice programmers ourselves we find it can be quite overwhelming to decide on how to approach big projects, which is why we would like to discuss software design patterns.

Experienced programmers know to solve problems with solutions that has worked for them before, which they can then reuse when necessary. Other programmers who can identify design patterns can apply them to solve problems and not have to rediscover them(Gamma, E et al; pg1). In a team development environment, if all developers know these patterns they can collaborate much easier with a common methodology.

In *Design Patterns: Elements of Reusable Object-Oriented Software,* design patterns are defined as *“descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.”*(ibidl; pg 3). This description portrays various topics of interest, mainly a general problem and solution in a particular context. The problem describes in which context to apply a design pattern, and the solution is an abstract description of a pattern that solves a design problem. A design pattern is not a concrete implementation of a solution, but rather a template that can be applied to different situations(ibid;pg3).

This relates to the project portfolio as there are many parts building up the application we created. We therefore use patterns throughout the project to solve problems using different techniques and technologies. These patterns improved the

way we developed the project and enabled us to work on different functionalities simultaneously.

One such design pattern that we found incredibly useful is the Model-View-ViewModel design pattern for developing the frontend. The intent of the MVVM design pattern is to separate the concerns of the view, the state and behaviour of the view and the data behind it. Each view is separated by the information it displays, such as a list of questions for searching, or a personalized user page. Each view can have a view model associated with it, which provides the view with the data to display. The view model has access to the data model, which is the data service in the case of the SOVA application, and can manipulate the database when certain events are triggered. If someone works on the UI they do not need to know what the logic behind the view model is, rather just the key functions to databind view element. This design pattern solves the problem of having to write code in view, and reduces the dependency between the developer and designer.

Another major part of our application is accessing the database and doing operations on it. For this we use the repository pattern, which solves many problems; operations need to be written once and can be reused, being able to test database operations without needing a UI, as well as adding a layer between the business logic and the database (Dhananjay Kumar, 2016). In conjunction with entity framework core this pattern gives quick access to the database and enabled us to create, read, update and delete information quickly. During development whenever someone made an operation someone else could make a unit test to ensure that it does what is intended. This testing made it easy to create HTTP methods in the web service because if it didn't work then we knew it was a problem with the HTTP not the database operations.

Information Retrieval

In this section we will introduce the relevance of a document by using the Tf-idf weighting technique. We will start by introducing what information retrieval is and the key concepts of information retrieval, then we will introduce the main topic of

our discussion which is weighting of a document or a term using tf-idf approach. Mainly we will focus on weighting using tf-idf and ranking. Finally we will explain how is been used in connection to our project.

Information retrieval is a way of retrieving relevant information from a collection of documents(Manning, Raghavan and Schütze, 2008). When a user searches for a term the system will compare the term to the collection of documents and as a result the system gives an output of the most relevant documents in order from the most relevant document to the least relevant one. Relevance of a document is the core concept of information retrieval and can be explained as the measure of how well the document satisfies the user's information need or a query (Svn.spraakdata.gu.se, 2018).

To measure the relevance of a document there are different approaches like tf-idf weighting, similarity measure and probabilistic relevance which deals with the probability of a term occurrence in a document.We will mainly focus on tf-idf weighting on this section. We are not going to discuss the construction of inverted index like collecting all the documents to be indexed, tokenization, linguistic preprocessing of tokens, index the documents that each term occurs in. We will assume all the preprocessing of inverted index is done before hand.

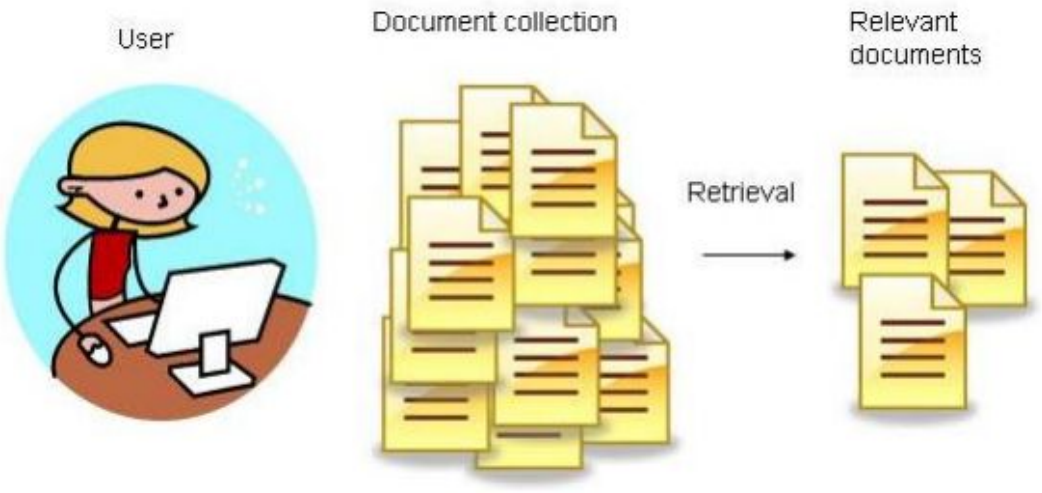


Fig 1: This figure shows the general overview of information retrieval (Svn.spraakdata.gu.se, 2018)

What is tf-idf?

Tf-idf stands for term frequency-inverse document frequency and it is a technique which is widely used in the area of information retrieval and explains the relevance of a term in a document or collection of documents(corpus) by calculating the weight of a term. The weight is the measure used to evaluate how important a term is to a document or to a corpus.Tf-idf is calculated by the product of term frequency with inverse document frequency but first let's explain what each of this two mean.

Term frequency is the number of times a word appears in a document, divided by the total number of words in that document and measures how frequently a term occurs in a document. The reason term frequency is divided by the total number of words in a document is the length of a document differs from document to document, for example let's say we have a 300 words in a document that has a word java 10 times and we have a 1000 words in other document that has a word java 10 times. In this case we can't say their term frequency is equal without considering the size of the document. Document frequency is the number of documents in the collection that contain a term t and inverse document frequency can be calculated as logarithm of total number of documents divided by the number of documents with a term t in it (see equation 1.2). A rare terms has higher idf than frequently occurred terms.[(Csee.umbc.edu, 2018)]

Term frequency (TF)

$$\mathrm{tf}\,(\mathrm{t},\mathrm{d}) = \frac{Frequency\ of\ term\ t, in\ document\ d}{Total\ number\ of\ terms\ in\ document\ d}$$

Equation 1.1 (Keet Malin Sugathadasa | Sri Lanka, 2018)

Inverse document frequency (Idf)

$$\text{idf}(t) = log\ \frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it}$$

Equation 1.2 (Keet Malin Sugathadasa | Sri Lanka, 2018)

Finally we can compute the tf-idf based on the formulas of term frequency and inverse document frequency.

$$tf - idf = tf * idf$$

Equation 1.3

Ranking of a document

When a user searches for a term the system needs to retrieve the most relevant information(document) and this relevance is called document ranking which ranks documents according to their relevance to a given query. By using tf-idf a document can be ranked by summing all the tf-idf of the terms occured in a document and based on this value documents will be retrieved according to ranking.

$$\text{Score}(q, d) = \sum_{t \in q \cap d} tfidf(t, d)$$

Equation 1.4 (Keet Malin Sugathadasa | Sri Lanka, 2018)

Where q is a query, d is a document and t is a term. Basically what this formula does is summing all the tf-idf weights of each term in a document and this will be the score of the document.

Ranking of a document can also be done using the different approaches like boolean model, vector space model and probability model. These are the different models in information retrieval.

Tf-idf is implemented and used in the project as it is an essential requirement to have when dealing with searching for information. For example in our database we created the tfidfi table in which the weight of terms is calculated. The mathematical formula in equation 1.3 is written in sql. The first part of the query belongs to term frequency and the second part of the query for inverse document frequency.

```
create table tfidfi as
(
  select  mwi.id, mwi.word,  cast((select wi.count  from wi where wi.id = mwi.id limit 1) as
double precision)
  /
  cast((select wordcount from wrdcount where wrdcount.id = mwi.id ) as double precision)
*
 (select log((cast((SELECT count(distinct id) FROM "wi") as double precision)
/
cast((select totterm.count from totterm where totterm.word = mwi.word ) as double precision
)))) as tfidf
, (select totterm.count from totterm where totterm.word = mwi.word) as nt

from mwi);
```

Functions are also created, for example a function called TFIDF_Match is created for ranking a document based on the above mathematical formula of ranking a document and as we can see in the screenshot below the function is used in visual studio in the dataservice.

```
public List<SearchResult> GetQuestionsByString(string title, int page, int pageSize)
{
    using (var db = new SOVAContext())
    {
        var question = db.SearchResults.FromSql("select * from TFIDF_MATCH({0})", title);
        return question
            .Skip(page * pageSize)
            .Take(pageSize)
            .ToList();

    }
}
```

We used one of the functions wrdCloud in the backend and this function takes a word as an argument and displays words with their weights which co-occur with the word.

```
public List<WordCloud> GetWordCloud(string text)
{
    using (var db = new SOVAContext())
    {
        var cloud = db.WordClouds.FromSql("select * from wrdCloud({0})", text);
        return cloud.ToList();
    }
}
```

Authentication

In the following I would like to elaborate on the concept of authentication and various aspects such as the types and factors of authentications. The goal is to look at some aspects of authentication to get a better but more general overview of the process itself and the different approaches that might occur in various situations. Finally I would like to refer a survey which would bring up an ever growing issue namely that in a digitalised world why the concerns regarding security are not taken into consideration on a much higher level (Dawn M. Turner, 2016).

Authentication in itself is the process in which the truth of an attribute that belongs to a piece of data is confirmed by an entity which is in most of the cases is the user. There is a clear difference between authentication and identification. Identification is the process where indicating a claim that would provide to a person an identity. It might include the confirmation of the identity of the person by validating their identity credentials and the authenticity of a website with a digital certificate. Authentication is considered as a process in multiple fields. Authentication can have three types.

The first type is the process in which a proof of identity is given by a credible person. This credible person should have a bulletproof and first-hand evidence. Communications that are centralized are authority based. Meanwhile decentralized technologies are more peer-based. This is also known as web of trust and used mainly for services such as email and files.

The second type is the process in which the attributes of the object is compared to the one that is stored and considered to be the original one. This type of authentication method is commonly used with currency.The third type of authentication method uses external documentation. In computer science a user can have access to a system on the basis of the credentials of the user.

It is important to look at the factors that affect the elements used in the authentication process. The first factor is the knowledge factor which would assume that the user in the particular situation knows something, which could be a password, challenge response or question. The second factor would assume that the user owns something which can for example be an ID card, software token, implanted device. The third factor would assume that the user inherits an attribute which could be a fingerprint, DNA sample, retinal pattern. Considering these types these are the various ways in which a user who should be authenticated can fall into a category which is characterized by the fact that the user either knows, has or inherits an attribute.

By combining the factors we can categorize the most frequent types of authentication that are available and in use:

In the case of using one factor we might get a single-factor authentication which is considered as the weakest level of authentication. The reason is that the use of only one factor can not provide a high level of authentication.When two factors are required it is considered a two-factor authentication. Networks may require the user to provide a password which the user should know and a token which the user would get an ownership. In the case of multiple factors we refer to a multifactor authentication. It is the process in which the user gets an access only after presenting two or more pieces of credentials to the mechanism of the authentication.

Another type of authentication is the so called strong authentication. The problem with strong authentication though is that it is often confused with two-and multi-factor authentication. It is important to mention that strong authentication does not necessarily mean multi-factor authentication. The main idea for a strong authentication would be that fact the factors need to be mutually independent and that the requirements would be more strict.

In case of digital authentication extra identity factors are included. The National Institute of Standards and Technology has made a model for the process of digital authentication. This model is used in order to make a secure authentication (NIST, 2017).

The first part is the enrollment where the user is initiating the process via getting a service provider. After the user became the subscriber the user would receive a username as a credential and a token. Within a session the user must provide the proof to perform transactions. When the user provides the necessary credentials these credentials need to be maintained by the service provider while the user who is subscriber has the responsibility to maintain their own credentials. This maintenance is called the life-cycle maintenance.

As our daily life and more of its aspects get more and more digitized the importance of security is growing as well. Just to point to a recent event that took place recently and had a big effect we could mention GDPR (general data protection law). Since more data is handled everyday than ever before the lack of proper internal processes can mean serious security threats to big data. As Internet of Things become a part of daily life the level of interconnectivity and reliance on internet technologies for data transfer creates a high vulnerability. Although the need for a high level of security is clear there are more and more surveys coming up with the shocking results that companies do not spend enough on security.

The Budapest Business Journal conducted a survey in which 1400 company leaders and IT professionals from 60 countries participated(BBJ, 2018). The conclusion of the survey and the article is that the companies should invest more time and money into security and that security should be treated as a more serious concern. Even though it seems that there is a delay in the response from the IT field there is a growing number of people and specialised organisations who are making

the effort in order to bring the problem into the light and are willing to discuss possible solutions for the future.

4. Portfolio project discussion

Discussion on the Subprojects

In the first subproject we started designing our stackoverflow database, which was not well designed at the start. We decided to redesign the database in later iterations which made the development of the app easier in the long run. We noticed things that could have been avoided in the beginning like redundancy, poor normalization, naming conventions. It was first later we corrected these things which was due to the lack of understanding what those concepts are at first. For example normalization was one of the concepts that took us long time to grasp and implement it to the project but as we moved on to subproject 2, we started noticing some of the things we done better while designing our database.

The first part of subproject 2 was dealing with data access layer which is the layer between the database and web service layer and provides access to the database. This was the first time we started to learn working on microsoft entity framework beside the assignments we worked in and one of the challenges was to learn how this framework functions and at the same time learning to create the data access layer. For example, one of the latest development we learned in this project was using linq which makes the code more compact and readable to the code.. In the first subproject we designed the database to be table per type which explains having separate tables for the domain class and we changed it to table per hierarchy which is one table for the the entire class hierarchy, because reduced the number of tables needed and simplified the CRUD operations. The downside to this being some null values in the main table, but with the table being so small it doesn’t affect the whole as much.

In subproject 3 , similar to subproject two we started by following the assignments we had. Information retrieval is one of the crucial concepts to have during the development of an app which requires way of searching and retrieving unstructured data. After we learned the core concepts of information retrieval like inverted indexing, tf-idf weighting, ranking documents etc. The challenge was to write all the mathematical formulas into sql and make the computation or calculation to rank the most relevant documents. The problems we faced during the computation were to compute two or three different things at the same time without thinking about the computation time. Also, if the system had to do these calculations each time the user searches, the searching would be very slow. We learned that precomputing the tfidf of the words beforehand and putting them into a table would speed up the process considerably.

Inconsistency in KnockoutJS

Regarding the front end and knockout.js as a framework there is an amount of inconsistency in the written code. In the service folder there is the file called ds.js which contains methods to retrieve information. This file contains getJSON() methods to get data. Other functionalities such as post and delete are placed in the viedmodels. The main reason this happened is that the group had difficulties with figuring out how to implement the functionalities such as put or delete therefore these functionalities were implemented in the files where the functionality would be needed to work without considering the inconsistency of the code.

Data Transfer Object (DTO)

In the beginning of our project we used DTO (data transfer object) which is an object that carries data between processes. The goal of the data transfer object is to batch up the multiple remote calls into a single call. Another advantage is that we

can encapsulate the serialization process for transferring data over the wire. So we can see the use of such a solution would be an ideal one and this solution was implemented in our application. However, the data transfer objects became irrelevant later in the project since other methods were implemented in the web service that would control the output.

This overcomplicates things because we need to change syntax in view model and view to access the items being sent from the server when dto's are sent. Reasoning for this is when do things as we learn, but don't change them everywhere when we find a better solution. This also made it harder in the frontend because the data objects we accessed are formed differently and a lot of time was wasted trying to figure out why it didn't work.

User Interface

In the UI we wanted better aesthetics and feedback in the form of information to the user and feedback for the users actions. The aesthetics of the site mostly reminds us of a 90's, maybe early 00 website. Here we would have liked to give different colors and change the placement and size of the elements in the UI. We lowered the priority of this because making the frontend function seemed more important.

In the UI, we wanted to give more information to the user as well displaying it better. For example, when we show the date to the user it is displayed as plain unaltered text. Here we could have changed it to show, for example, how long time ago a question was posted, like in most social media sites. Currently the list of posts only gives the user the title of the post so it will be hard for a user to decide which post is relevant based only on that. Other information such as score, tags, number of answers and authors name could help the user decide on picking a relevant post. When a user has picked a post and the post contains no comments or answers, there is only text which states "Comments" and "Answers". Here it would be better if there were some messages to tell the user there are no comments or answers if that's the case.

Also every place where pagination is used, it should show the number of pages there is in total and in the users page(my page). Or perhaps displaying the number of annotations, marks, and searches made.

The navigation in the project could also be improved as the only reliable way to navigate is through the navigation bar. Specifically there is no consistent way for the user to go back to the previous page. The only back button is in question view that takes the user back to questionList. if user clicks on a Show Post link for a annotation or mark in My page

In the view of the fact that the navigation isn't always straight forward as the user has to use the nav bar to navigate, though there is a back button on the question page. However, if user clicks on a Show Post link for an annotation or mark in My page, the back button in the question page will navigate to question list, not back to my page.

Authentication

In the process of authentication we used JSON web token. JSON web token is an open standard that is using JSON object in the process of securely transmitting information between parties. In our application we used authorization with a user object that was hard coded and this hard coded user was used. This solution has its advantages and disadvantages. The advantage is that this solution provides a rather simple way for the authentication process. This process can be tested via Postman where the authentication type needs to be chosen accordingly and the necessary credentials such as username and password should be provided. If the credentials are not correct then the page with authorize attribute could not be accessed.
The disadvantage of this solution is that the authorization process is not dynamic. It is not dynamic because the only user that can be authenticated is the user we already hard coded. Although the functionality to create and store a new user in the database exist. This could be solved if there was a front end with a form where there would be the option to sign up where the user would give the necessary credentials then these credentials would be stored in the database. After the creation of the new

user the user would be able to use the login function which, from a practical sense, is the authentication itself. After logging in the user would be logged in and the data that is associated with that particular user it would be stored in the database and could be retrieved. For now the authorise tags are all commented out.

Project work process

Throughout this project we feel we needed a better overview, better planning in the group and communication. We often planned which days we meet but lacked scope for the days and responsibility to deliver. We tried on multiple occasions to do pair programming but the gain from this was lower than first expected because the discussion and learning between group members was something that could have been read up on instead of one teaching the other. In the process to work better together as a group we sometimes lost overview over the project, which resulted in inefficiency during the project work.

Conclusion

This project has been a big learning experience in full stack development, and has helped us develop skills in all areas of making a web application. It has been challenging as all of us in the group had very little experience in all the technologies involved, but now feel we have a better understanding. However, there are some aspect of our application that are lacking. For future work we should try to finish authentication and a user login, as well as working on the frontend to display all of the information we have available in the database. Also, just general aesthetic in the styling of the application should be improved.

Bibliography

Addy Osmani (2011) *Understanding the Publish/Subscribe Pattern for Greater JavaScript Scalability*. URL: https://msdn.microsoft.com/en-us/magazine/hh201955.aspx

Addy Osmani (2017) Title: Learning JavaScript Design Pattern. Chapter: *JavaScript Design Patterns*. URL: https://addyosmani.com/resources/essentialjsdesignpatterns/book/#designpatternsjavascript

BBJ (2018) *Firms not spending enough on info security EY survey*. URL: https://bbj.hu/analysis/firms-do-not-spend-enough-on-info-security---ey-survey_157758

Dawn M. Turner (2016) *Digital Authentication - the basics*. URL: https://www.cryptomathic.com/news-events/blog/digital-authentication-the-basics

Developers Google (2018) *Responsive Web Design Patterns*. URL: https://developers.google.com/web/fundamentals/design-and-ux/responsive/patterns

Dhananjay Kumar (2016) *How to implement the Repository Pattern in ASP.NET MVC Application*. URL: https://www.infragistics.com/community/blogs/b/dhananjay_kumar/posts/how-to-implement-the-repository-pattern-in-asp-net-mvc-application

Gamma, Erich (1994) *Design Patterns: Elements of Reusable Object-Oriented Software.* [Book]

Guru99 (Unknown Date) *What is Software Testing? Introduction, Basics & Importance*. URL: https://www.guru99.com/software-testing-introduction-importance.html

Keet Malin Sugathadasa | Sri Lanka. (2018). TF-IDF in the Field of Information. URL: https://keetmalin.wixsite.com/keetmalin/single-post/2017/06/05/TF-IDF-in-the-Field-of-Information-Retrieval

Manning, C., Raghavan, P. and Schütze, H. (2008). *Introduction to information retrieval*. New York: Cambridge University Press. [Book]

MDSN Microsoft (2013)*Understanding the basics of MVVM design pattern*. URL: https://blogs.msdn.microsoft.com/msgulfcommunity/2013/03/13/understanding-the-basics-of-mvvm-design-pattern/

NIST (2017) *Digital Identity Guidelines*. URL: https://pages.nist.gov/800-63-3/sp800-63-3.html

Paul Sherman (2018) *How Single-Page Applications Work.* URL: https://medium.com/@pshrmn/demystifying-single-page-applications-3068d0555d46

Richard Bovell (2013) *Understand JavaScript Callback Functions and Use Them.* URL: https://javascriptissexy.com/understand-javascript-callback-functions-and-use-them/

Svn.spraakdata.gu.se. (2018). URL: https://svn.spraakdata.gu.se/repos/richard/pub/sv2122_web/STIR.pdf

Stackoverflow (2016) *Differeence b/w mostly fluid and column drop responsive design patterns*. URL:

https://stackoverflow.com/questions/40065487/differeence-b-w-mostly-fluid-and-column-drop-responsive-design-patterns

Software Testing Fundamentals A (Unknown Date) *White Box Testing*. URL: http://softwaretestingfundamentals.com/white-box-testing/

Software Testing Fundamentals B (Unknown Date) *Black Box Testing*. URL: http://softwaretestingfundamentals.com/black-box-testing/

Software Testing Fundamentals C (Unknown Date) *Gray Box Testing*. URL: http://softwaretestingfundamentals.com/gray-box-testing/

Software Testing Fundamentals D (Unknown Date) *Unit Testing* .URL: http://softwaretestingfundamentals.com/unit-testing/

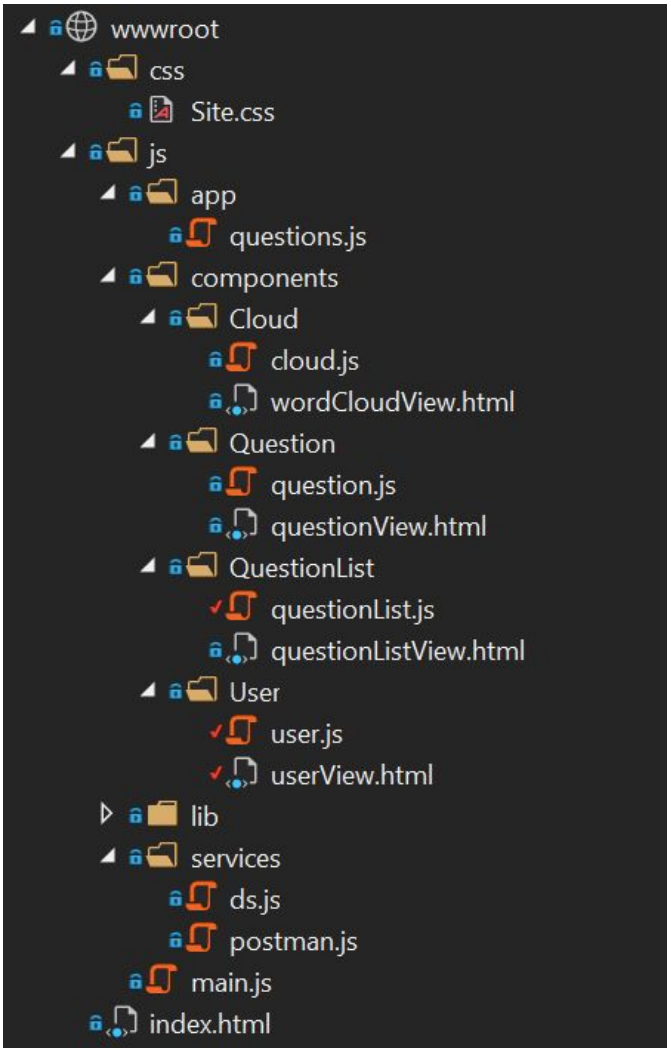
Software Testing Fundamentals E (Unknown Date) *Integration Testing*. URL: http://softwaretestingfundamentals.com/integration-testing/

Tfidf.com. (2018). *Tf-idf :: A Single-Page Tutorial - Information Retrieval and Text Mining*. URL: http://www.tfidf.com/

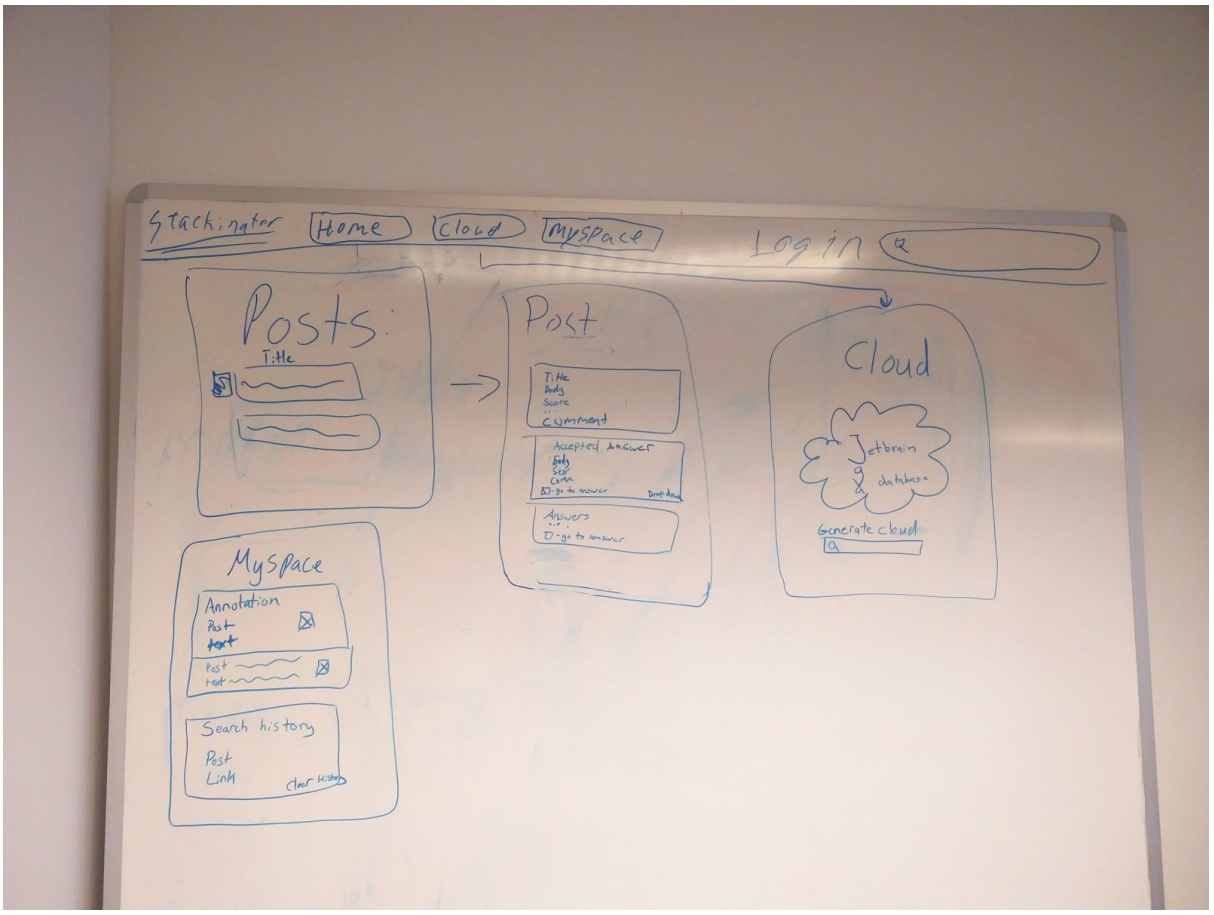
Wikipedia (2018) *Software testing.* URL: https://en.wikipedia.org/wiki/Software_testing#Economics

Appendix

Appendix 1:



Appendix 2:


Stackingtor
Home
Cloud
Myspace
Login
Posts
Title
Post
Title
Body
Score
Comment
Accepted Answer
Answers
Cloud
Jetbrains
database
Generate cloud
Myspace
Annotation
Post
text
Search history
Post
Link
clear history

Appendix-3

```
//get functions
var getPosts = function (url, callback) {
    url = url === undefined ? "api/questions" : url;
    $.getJSON(url, callback);
};


var getPost = function (url, callback) {
    $.getJSON(url, callback);
};

var searchPosts = function (terms, callback) {


    $.getJSON("api/questions/name/" + terms, callback)

};

var getUser = function (userId, callback) {
    $.getJSON("api/users/" + userId, callback)
};

var getSearchHistory = function (userId, callback) {


    $.getJSON("api/searchhistory/" + userId, callback);

};
```

```
var getAnnotations = function (userId, callback) {
    $.getJSON("api/annotations/" + userId, callback)

};

var getMarks = function (userId, callback) {
    $.getJSON("api/mark/" + userId, callback)
};

var postSearch = function (postData) {
    $.ajax({
        type: 'POST',
        url: 'api/searchhistory/add/',

        data: JSON.stringify(postData),
        contentType: 'application/json'
    });
}

var createcloud = function (term, callback) {
    $.getJSON("api/wordCloud/" + term + "", function (data) {
        callback(data)
    })
};
```