

RANDOMNOOBCODES

Emil Persson <peem18ma@student.ju.se>

A Project Work in *Web Development Fundamentals*

Jönköping University 2019

Table of Contents

Introduction.....	2
Architecture.....	3
Middleware architecture	3
Database.....	5
Graphical User Interface	7
GUI first drafts.....	8
GUI result.....	9
Web Application	12
Implementation.....	12
Structure.....	13
Security.....	13
SQL Injection	13
Hashing Password	13
Cross site scripting.....	14

Introduction

The goal for this project was to make a fully functioning web application that uses resources from a database to generate the web pages. In this case the resources that the web application should use is a blog, portfolio and comments. Every blog post should be able to contain comments that visitors of the site have written. The blog posts, portfolio projects and comments should be managed by the web applications administrator, but every visitor of the site should be able to view the different resources and to comment on the blog posts.

The finished product should include a web application with a fully functioning log in system for the administrator such that when the admin is logged in to the web application, he or she should be able to perform CRUD operations on the different resources there is in the application.

As you can see in Figure 1 Use Case diagram of user and admin interactions you can see how the users and the admin of the website will interact with it. On this website only, the admin will be able to login and manage the posts, comments and project resources. The users of the website can interact with the website through viewing blogposts, where they can leave comments on them and get responses from other commenters. The users can also go to the portfolio to find previous projects that the project owner have worked on before. There is also a contact page and an about page that the users can view, but there is no real interaction between the web application and the user there.

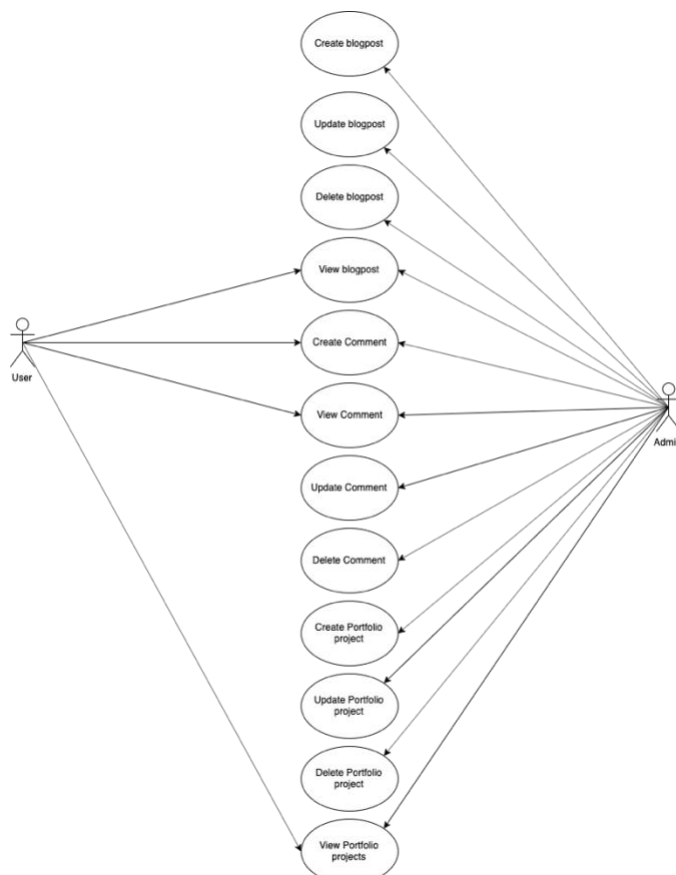
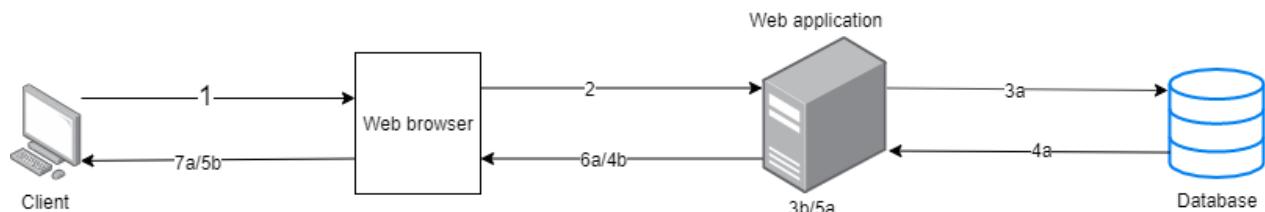


Figure 1 Use Case diagram of user and admin interactions

Architecture

The architecture of this website is really simple, they only use a web application, a database, web browsers and a client.

As you can see in Figure 2 Interaction between the different architectural parts of the website there is two routes that the request could take. Depending on what the request is about the web application can do two things. It can use a SQL query to get some data out of the database, then use the data to generate a response and send it back to the client. Or the web application already has all the necessary information needed to generate the response and send it back to the user. For example, when loading the blog site, it needs to send a SQL query to the database to get all of the blogposts objects out of the database. Then uses the blogpost objects to generate the response and sends it back to the user. Then there is the other way, if the web application needs to load the about page then it has all of the necessary information needed to generate the webpage and send it back to the client.



Interactions

1. Interacts with
2. Sends HTTP request
3. a) Sends SQL query
b) Generate response
4. a) Sends back table with data
b) Sends back HTTP response to web browser
5. a) Generate response
b) Display result to end-user
6. a) Sends back HTTP response to web browser
7. a) Display result to end-user

Figure 2 Interaction between the different architectural parts of the website

Middleware architecture

This web application is built on a middleware architecture. As you can see down below in Figure 3 Basics of how a middleware architecture works this means that the client sends a request to the server. The server then sends the request to different middleware's. When a middleware receives a request, it tries to handle it. If it can't handle the request it sends it to the next middleware or does a small change to the request before sending it along to the next middleware. This keeps on going until the request finds a middleware that can handle it. When it does, the middleware generates a response and sends it back to the client.

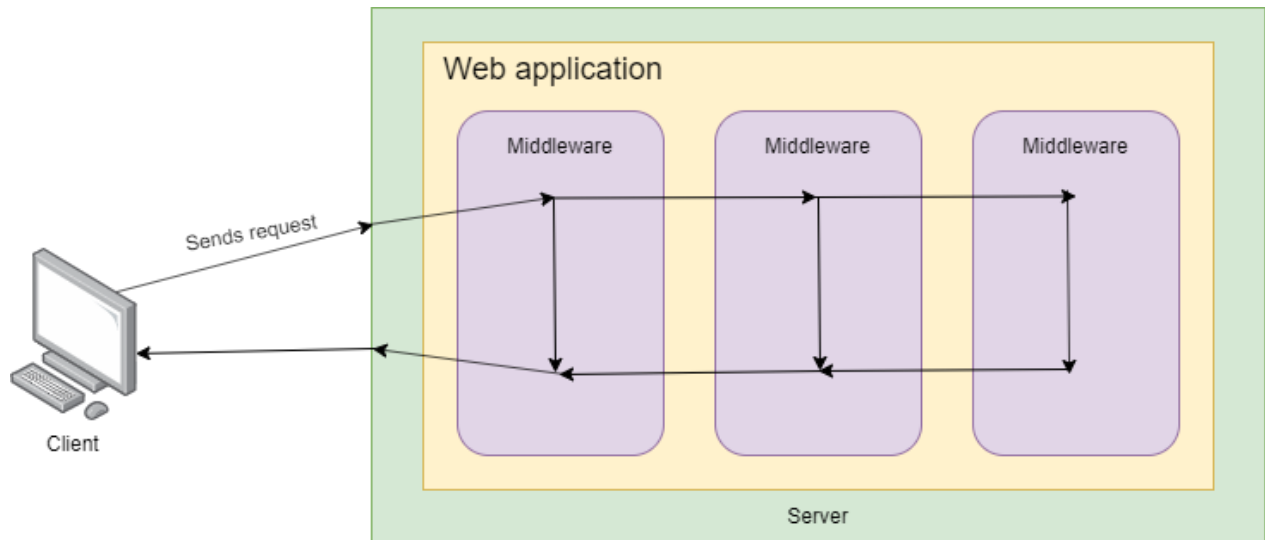


Figure 3 Basics of how a middleware architecture works

Database

The web application uses a relational database, the file is saved on the system as a database file. As you can see down at Figure 4 Internal structure of the database this database contains blogposts, comments and projects. The blogpost contains basic information about the blogpost such as an id to be able to distinguish the blogposts from each other. They also contain a header so that the user knows what the content of the post will be about. Every blogpost also contains a main text that contains the main text that the admin of the website posted.

The comment entity uses a comment id to distinguish the comment from other comment. This comment id is used for the admin to be able to delete and edit comments. It also utilizes the blogposts id as the foreign key. By using the blogposts id as it's foreign key constraint it means that every comment belongs to a specific blogpost. By doing this the web application can get every comment that belongs to a specific blogpost from the database when loading in the blogpost. When the user creates a comment on a blogpost it will use the blogpost id to connect the comment to the blogpost through the foreign key constraint.

The project resource consists of a name a description, a link and an id. The name is just a short name of the project in this case it is usually just a short string consisting of a few words. Each project contains a specific id that is used to identify a specific project. It is often used for when the admin wants to edit or delete a project from the database. Instead of posting to much on the server the project object contains a link to the project, this link can link to whatever but preferably in this context should link to a GitHub project or a finished project that is up and running. The last attribute that is in each project is the description, this contains a short text that should contain a short description of what the project is about.

Relations that exists in the database is only existing between the blogpost and the comments. This is just because the user would want to interact with the blogpost through leaving comments on them such that the admin can read response and leave response on comments that they would like to.

RandomNoobCodes ER

Emil Persson | October 20, 2019

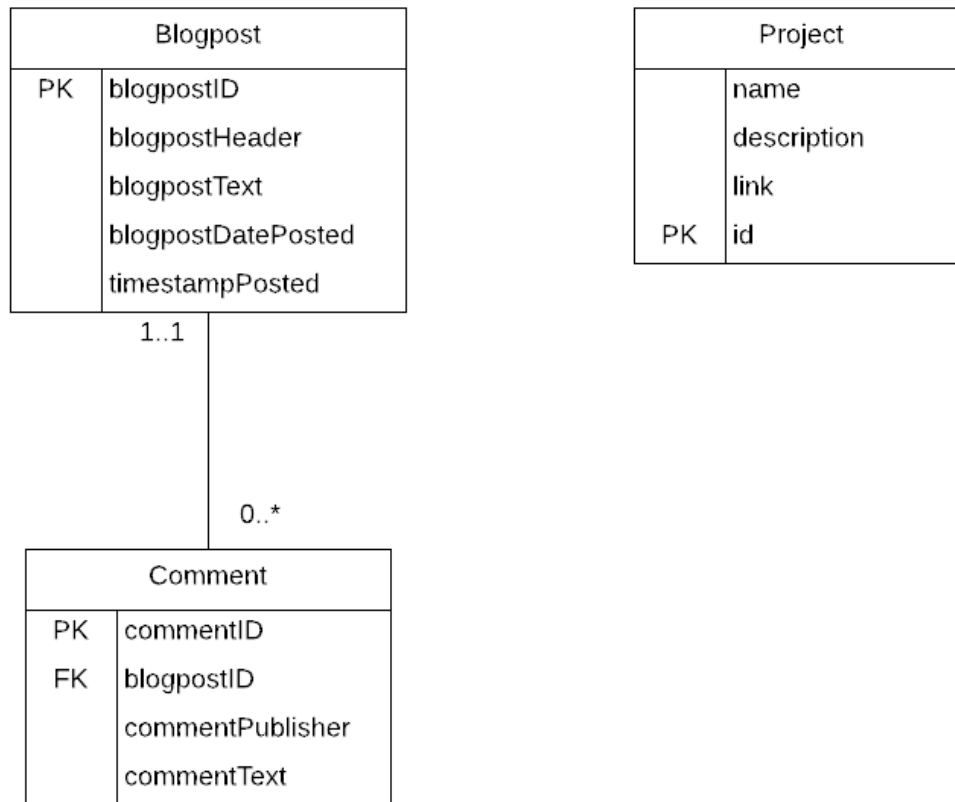


Figure 4 Internal structure of the database

Graphical User Interface

This website uses a simple but fairly straightforward design. This is so that everyone that uses the design should know where to click and what to do on the website. At the figures down below, you can see how the original sketch for the website looked and the screenshots are the finished result.

GUI first drafts

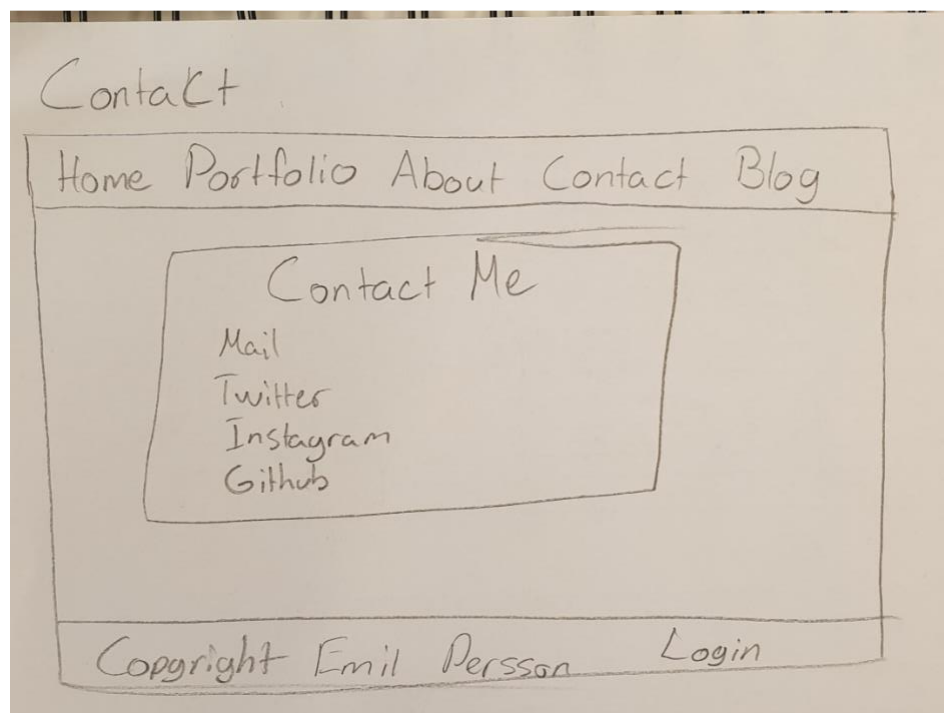
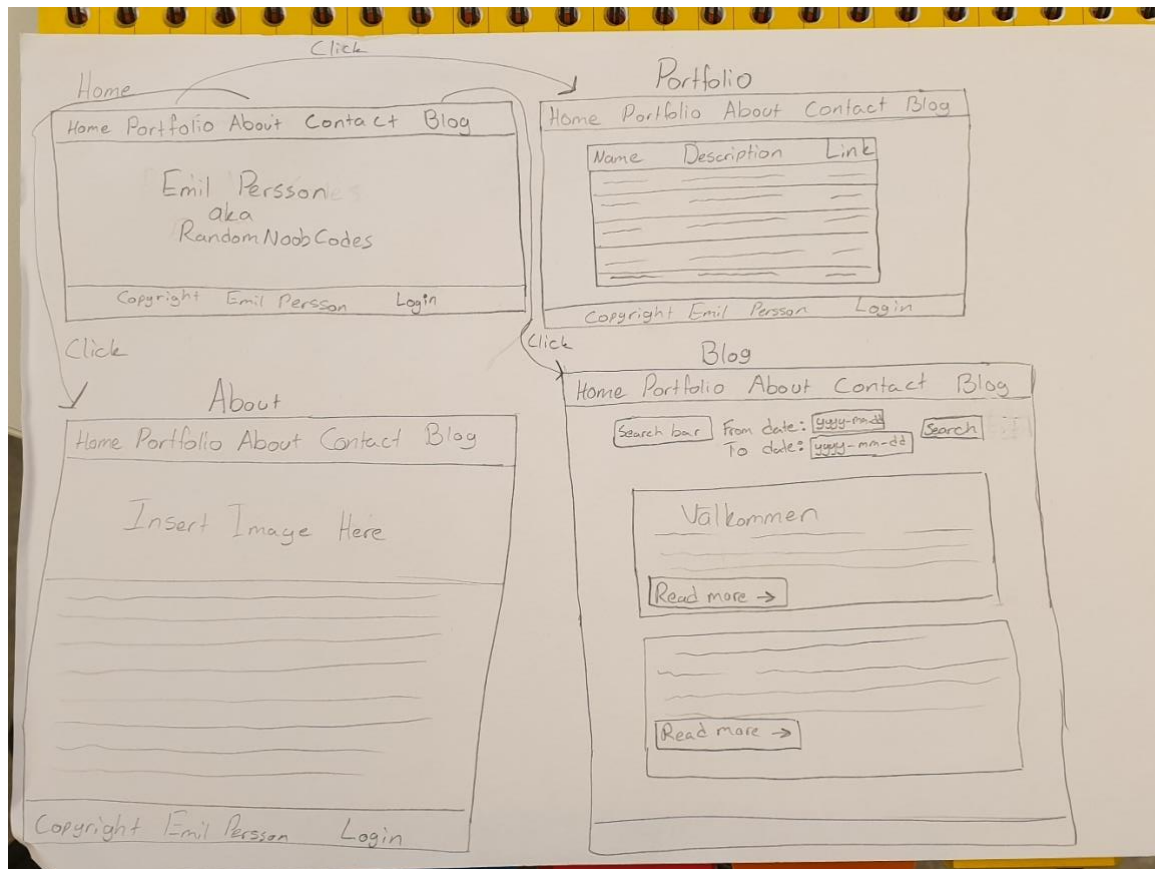


Figure 5 Sketch of GUI design

GUI result

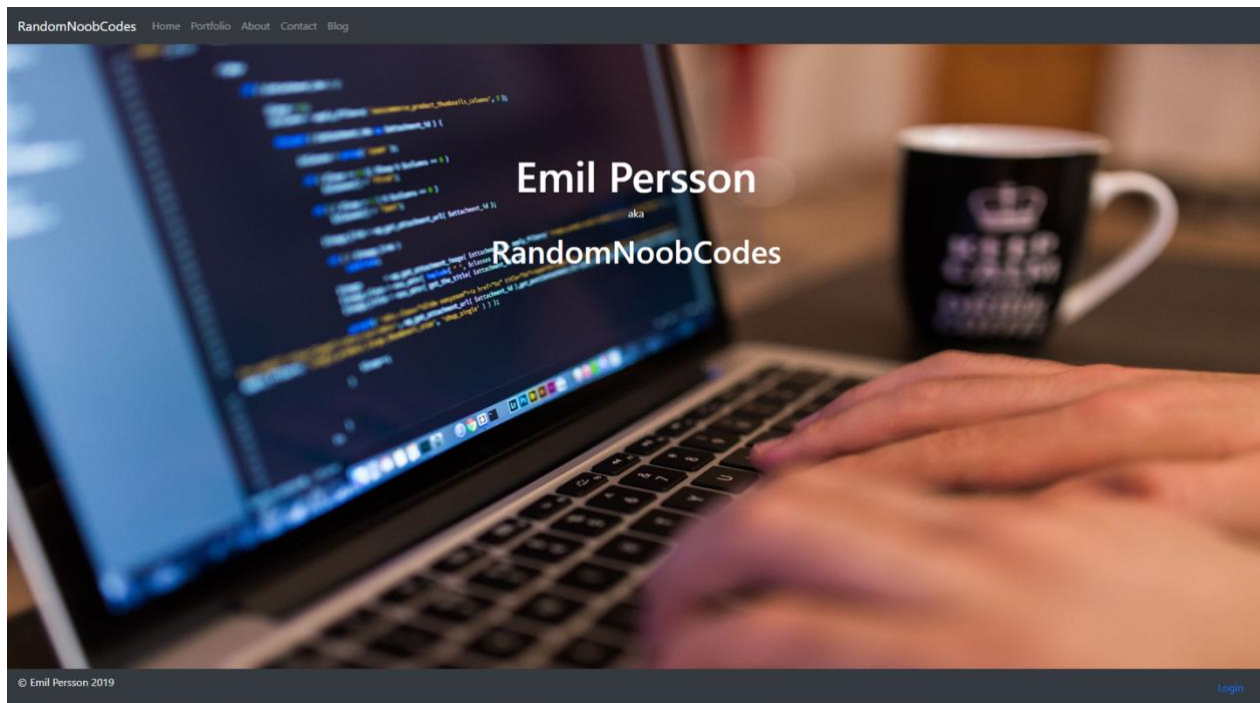


Figure 6 Result of homepage

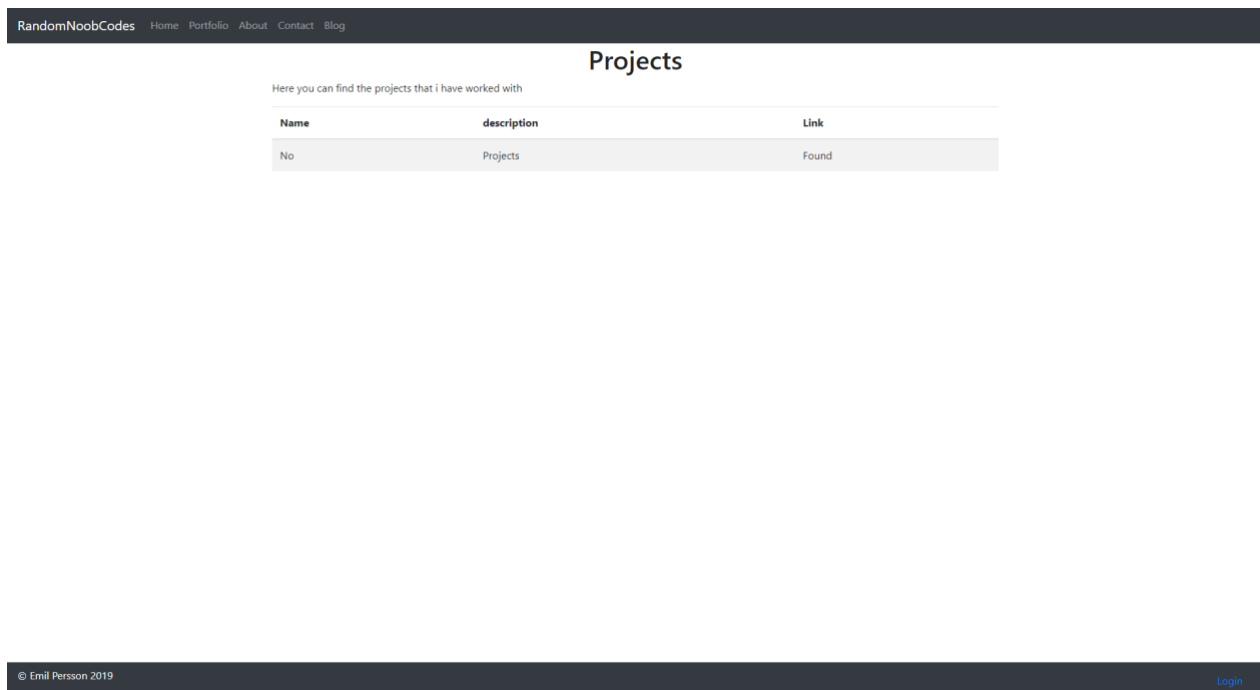


Figure 7 Result of portfolio page

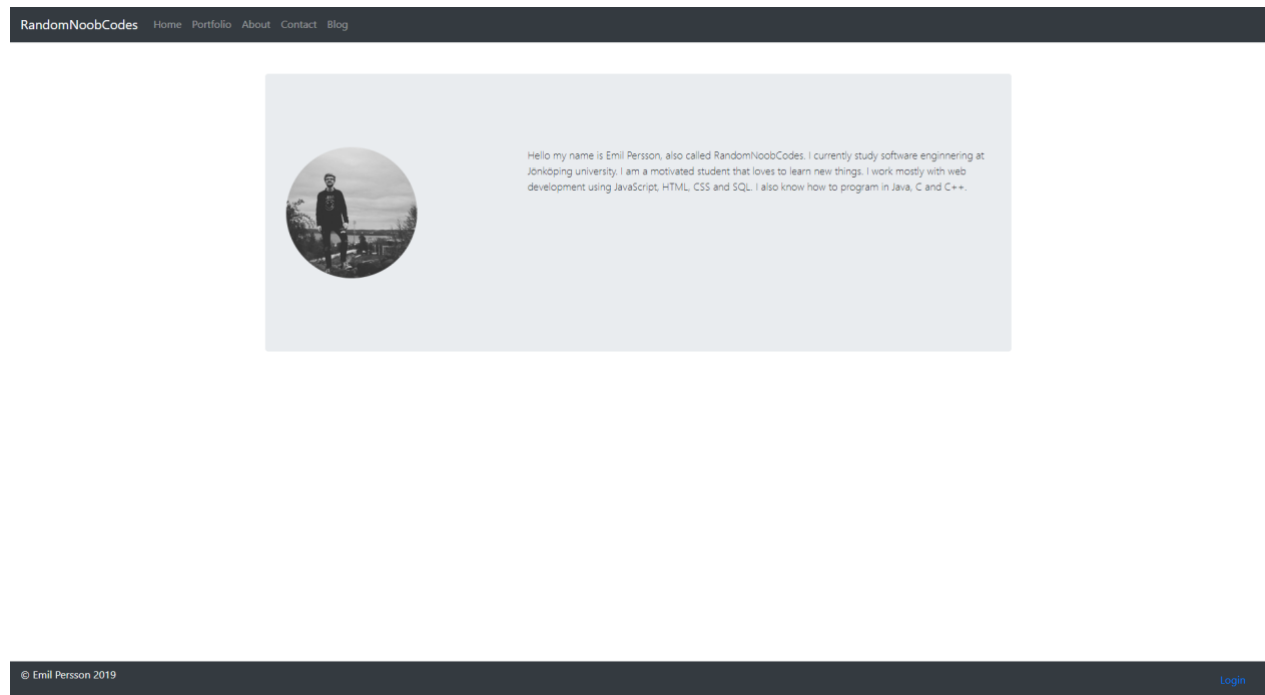
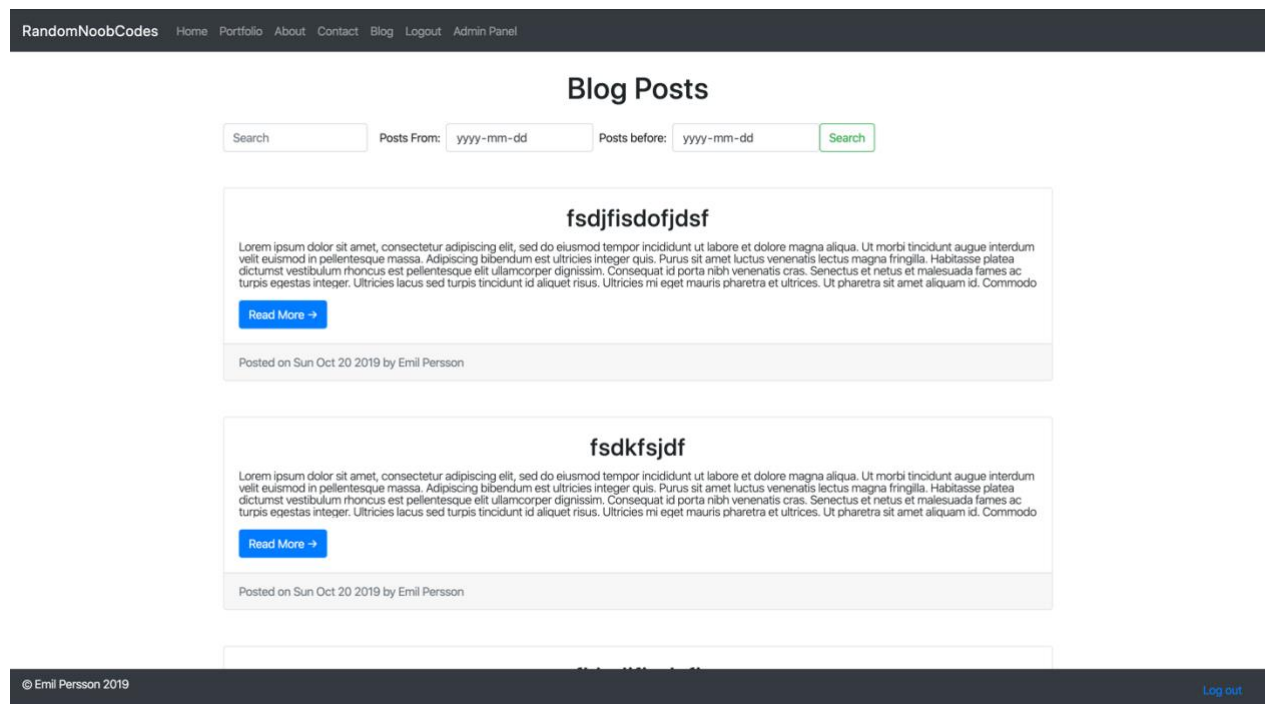


Figure 8 Result of About page



RandomNoobCodes [Home](#) [Portfolio](#) [About](#) [Contact](#) [Blog](#) [Logout](#) [Admin Panel](#)





Title	Edit	Delete
fsdjfisdofjsdf	Edit	Delete
fsdkfsjdf	Edit	Delete
fklsdjfiodsfj	Edit	Delete
Hello	Edit	Delete

© Emil Persson 2019 [Log out](#)

RandomNoobCodes [Home](#) [Portfolio](#) [About](#) [Contact](#) [Blog](#) [Logout](#) [Admin Panel](#)

Contact Information

Here is some information about how to get in touch with me

-  If you want to contact me about work or anything like that feel free to send me an email at emilper1999@gmail.com
-  I also have a [Twitter](#) account where you can follow me to get quick updates about me and my life.
-  Here you can find my [Instagram](#) account where I mostly post pictures of code and the life of me
-  You want to find my projects or see what I have worked on you can find it [Here](#)

© Emil Persson 2019 [Log out](#)

Web Application

Implementation

When the web application was first implemented the basic layout for the web application was made. The layout for this web application is made using HTML, CSS, Bootstrap 4 and express-handlebars. HTML, CSS and Bootstrap 4 was used for creating the visual design for the webpages. For this project there has not been much effort on visual design the webpages, so most of the visual design have been implemented using a CSS framework called Bootstrap 4. By using a CSS framework, it is much easier for a programmer to implement design and be able to focus more on the functionality. All of the backend functionality has been written in JavaScript. The reason that the project was written in JavaScript is because it is one of the most used programming languages on the web. There is also a lot of support on the web for JavaScript, like Node.js, Express and npm. For this web application a SQLite database have been implemented using the SQL and JavaScript. It does this with the help of the node package sqlite3 for express. This package is used to be able to handle database queries with JavaScript.

Because this web application runs on backend JavaScript code it runs Node.js as it's runtime environment. On top of this the web application is running the node package express. Express is used to handle incoming HTTP requests using a middleware architecture (As seen in Figure 3 Basics of how a middleware architecture works). In Figure 9 Example of a middleware in express you can see how a basic HTTP GET request is handled in express. In this case it is just for the URI /. So for example say that I have a website called somewebsite.com if I would have implemented this in my web application if I would go to <http://somewebsite.com/> then this middleware would be used to render the homepage.

```
app.get('/', function (request, response) {  
  response.render("home.hbs")  
})
```

Figure 9 Example of a middleware in express

In this example it sends back a response to render the home handlebars file. To be able to render this file the application uses the view engine express-handlebars. The way this works is that if you would like to parse the response render some data you could do this and use the data inside the like the example in Figure 10 Example of how to give data to handlebars file.

```
<div class="text-center mb-4">  
  {{#if validationErrors.length}}  
    <p>Errors:</p>  
    <ul>  
      {{#each validationErrors}}  
        <li>{{this}}</li>  
      {{/each}}  
    </ul>  
  {{/if}}  
</div>
```

Figure 10 Example of how to give data to handlebars file

Structure

The project consists of many different types of files, because of this it has to be structured into files and folders. The backend JavaScript code have been structured into different files depending on what they should handle. It has been structured into different routers that handle a specific resource. For example, the admin router only contains middleware's that control operations and handles URIs that the admin can use when logged in to the web application.

The folders have a specific structure too. Because the web application is using express-handlebars it needs to have a specific structure so that it knows where to look for the views and the layout. The express-handlebars package looks for a layout that is used to render the webpages with the help of views. When the response render function is passed a view and a model it looks for the view in the view folder. All of the backend JavaScript files are located inside the root folder.

Within the files there is a certain structure that is followed. The JavaScript files start with getting all the requirements for running the web application. After that the different middleware's have been implemented.

Security

An important aspect of every project is security and especially when you make a web application. This web application has been made with security as one of the focus points. The security vulnerabilities that this web application handles is SQL injections, CSRF (Cross Site Request Forgery), XSS (Cross Site Scripting) and Hashing passwords.

SQL Injection

An SQL injection is when the user of the web application tries to enter some SQL query into a field on the website that is used to send an application. To prevent this type of attack what we do it use placeholders in the queries that's sent to the database. In SQLite the question mark symbol is used as a placeholder. The SQLite middleware then takes in a query and some values. Values then replace the placeholders and sends a legitimate query to the database and get back some information. By doing this we prevent people from sending in their own query and getting data that they should not have.

Hashing Password

Another security aspect for this project is login validation and how to store passwords. When storing passwords, it is not just enough to encrypt them because an encrypted password is possible to decrypt. What the developer must do is to hash the passwords and then store them. This web application uses a node package called bcrypt when handling passwords. In this project there will only be one user, so the user information is hardcoded into the application. But storing a password in plain text is not really that good, so what the application does is that it has a stored hashed version of the password. When a user tries to login the application uses a compare method that compares the password that the user inserted, with the hashed value that is in the application. The reason to do this is so that if a hacker gets a user's information out of the web application, they can't get the password in plain text, instead they get a hashed value of the password that they will have to crack using brute force.

Cross site scripting

A security issue that is very important to protect the users from is cross site scripting. Every time that a user has the opportunity to input information into a form you will need to secure the information that the user has given. The information that have been inputted by the user should not contain any HTML tags or types of quotes. What the web application does is that it escapes the tags and uses the escaped value to replace the tags that the user has written. This means that the users are unable to insert any of the HTML tags and therefore can't post any malicious script on the website, thus stopping the hackers from doing performing the action that you can see in Figure 11 Basics of cross site scripting.

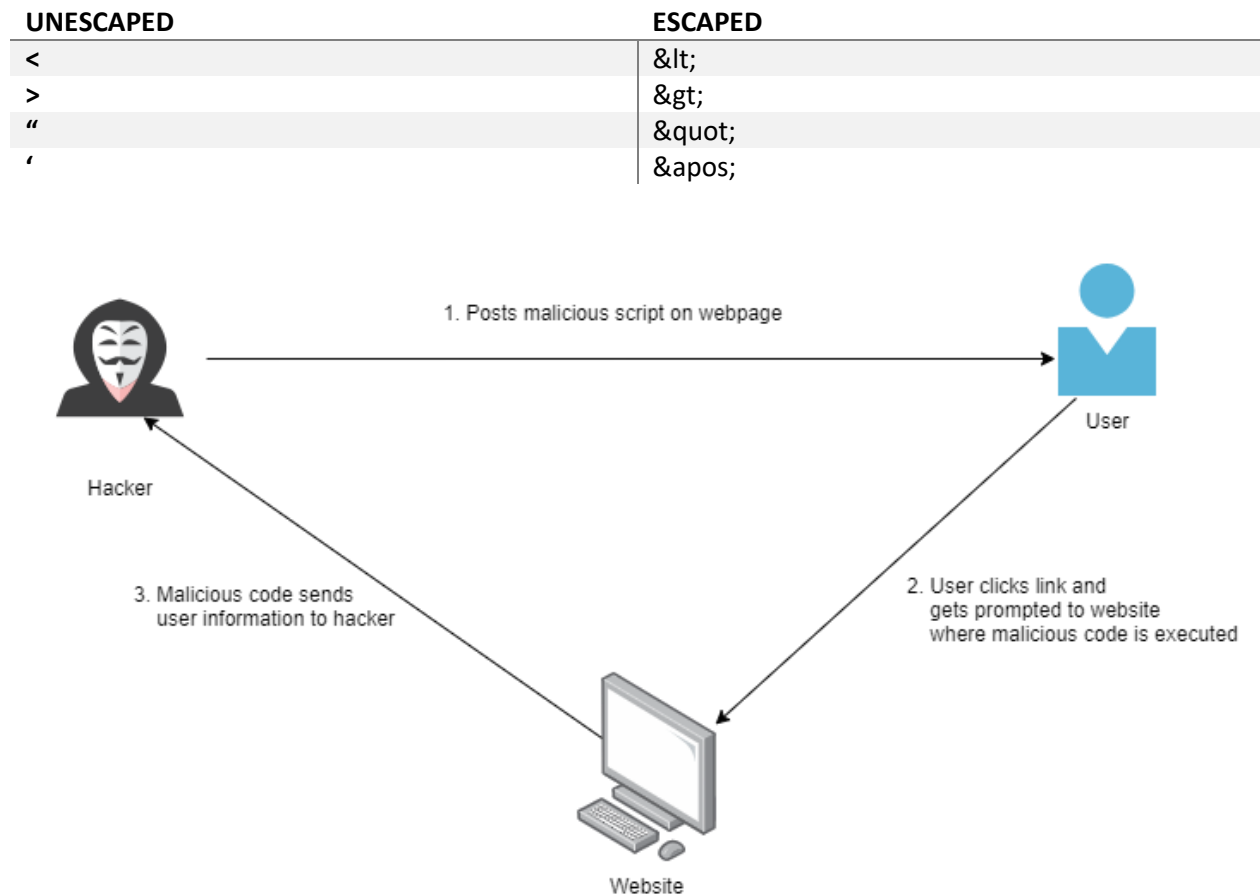


Figure 11 Basics of cross site scripting

The way that this is done is through implementing the web application with express-handlebars. As you can see in Figure 12 Example of handlebars escaping, this is the way that handlebars escape HTML tags. So, if the user would input some of the forbidden tags handlebars would convert this to the escaped version of the tag.

```
>{{blogpostHeader}}
```

Figure 12 Example of handlebars escaping