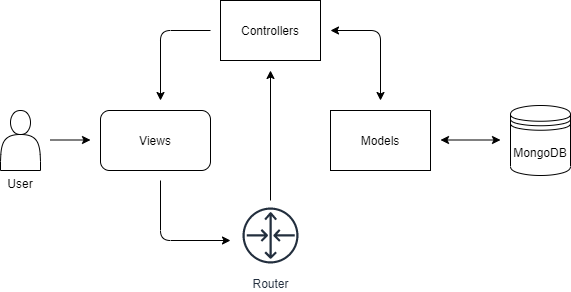
# 1. Design and architecture

## 1.1 Architecture design

We use the MVC design pattern to implement our application. The general design is shown in Figure 1.1.1



*Figure 1.1.1. MVC pattern*

MVC architecture makes the update of features and maintenance easier, allowing us to develop more efficiently.

## 1.2 Communication design

The communication design between modules is straightforward. A user makes actions on the view, which is our web page, then the front-end script sends the requests. The requests are handled by the router, and then execute the corresponding controllers to deal with the requests. The router codes are shown in Figure 1.2.1.



*Figure 1.2.1. router design*

The controllers receive the requests, then call the methods defined in the model to send queries to MongoDB. Then the controllers receive the data, build the JSON object, and return the object to the front-end scripts.

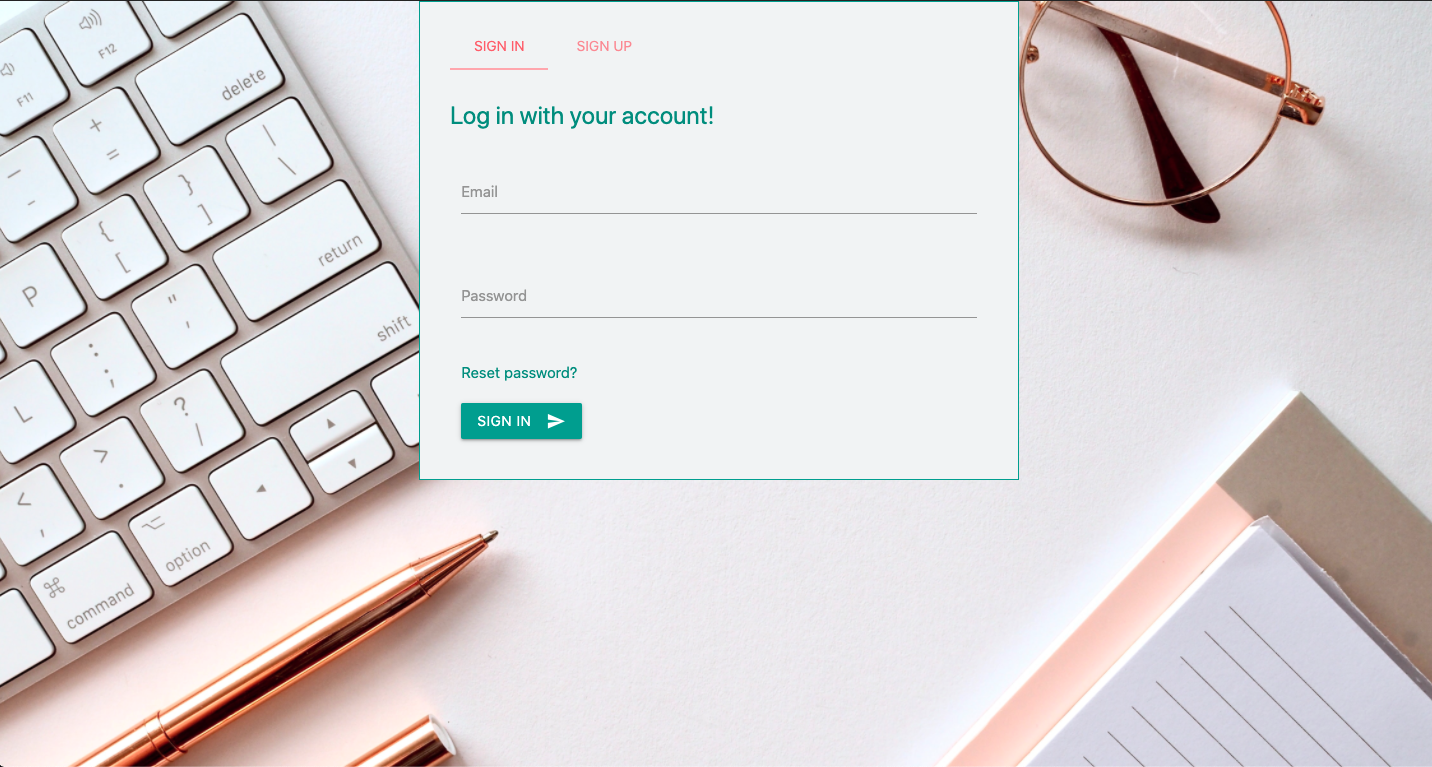
## 1.3 Main components

### 1.3.1 Views

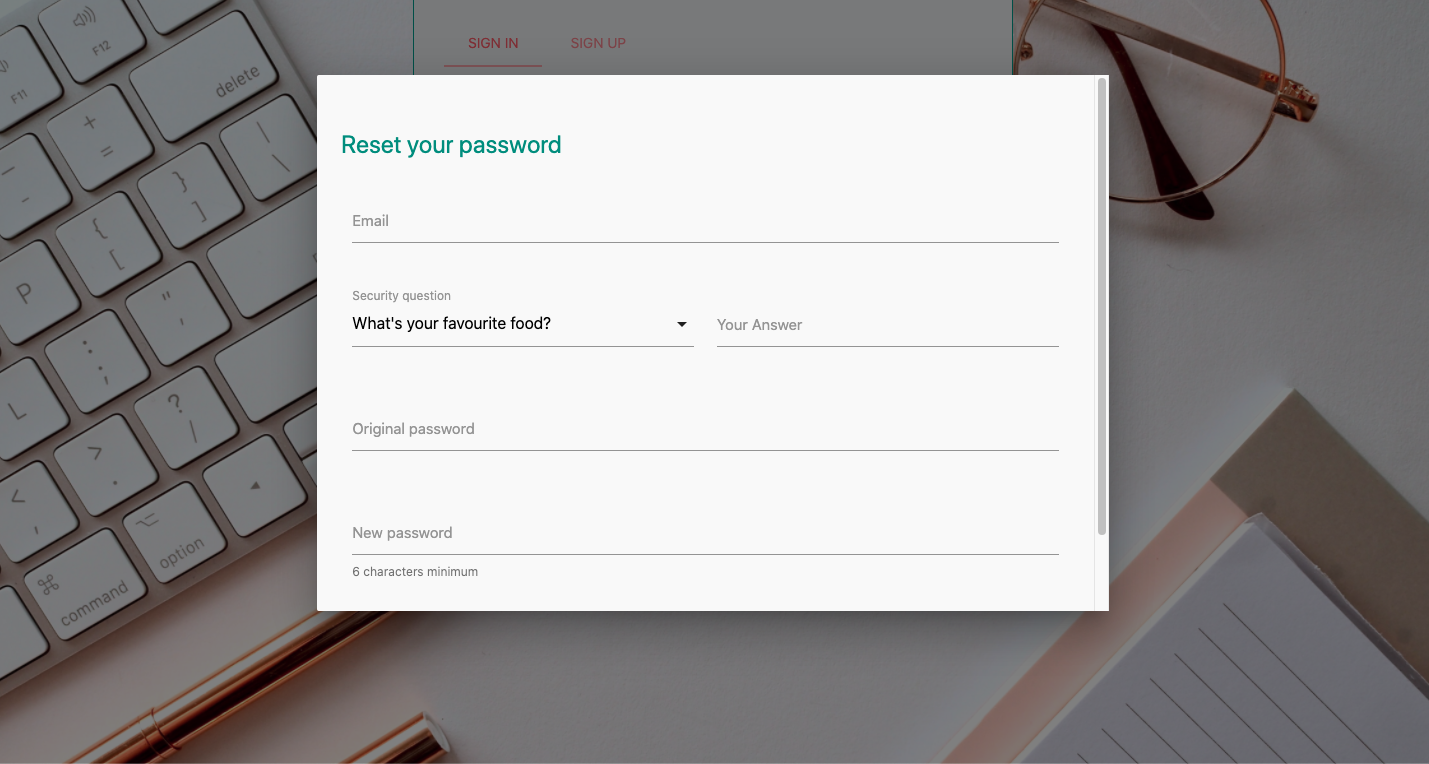
The front-end page is based on materialize, a modern responsive front-end framework. Materialize is simple to use, and provides a large number of components and javascript extensions, which makes the page beautiful and improves human-computer interaction.

The front end is divided into two parts, namely *index.ejs*, which is responsible for user login, registration and password modification; *main.ejs* is responsible for displaying the analysis function of the web page.

*index.ejs*. After starting the project, the user can enter http://localhost:3000 in the browser to enter the application's login page (as shown in Figure 1.3.1). The login and registration forms are presented through the built-in tabs of materialize; when the user clicks the "reset password?" Link, the password change form will pop up (as shown in Figure 1.3.2). It is worth noting that the user needs to complete the login function before accessing the subsequent web analysis function. If the user visits http://localhost:3000/main without logging in, the request will be intercepted and the user will be prompted to log in, and then redirected to log in page.

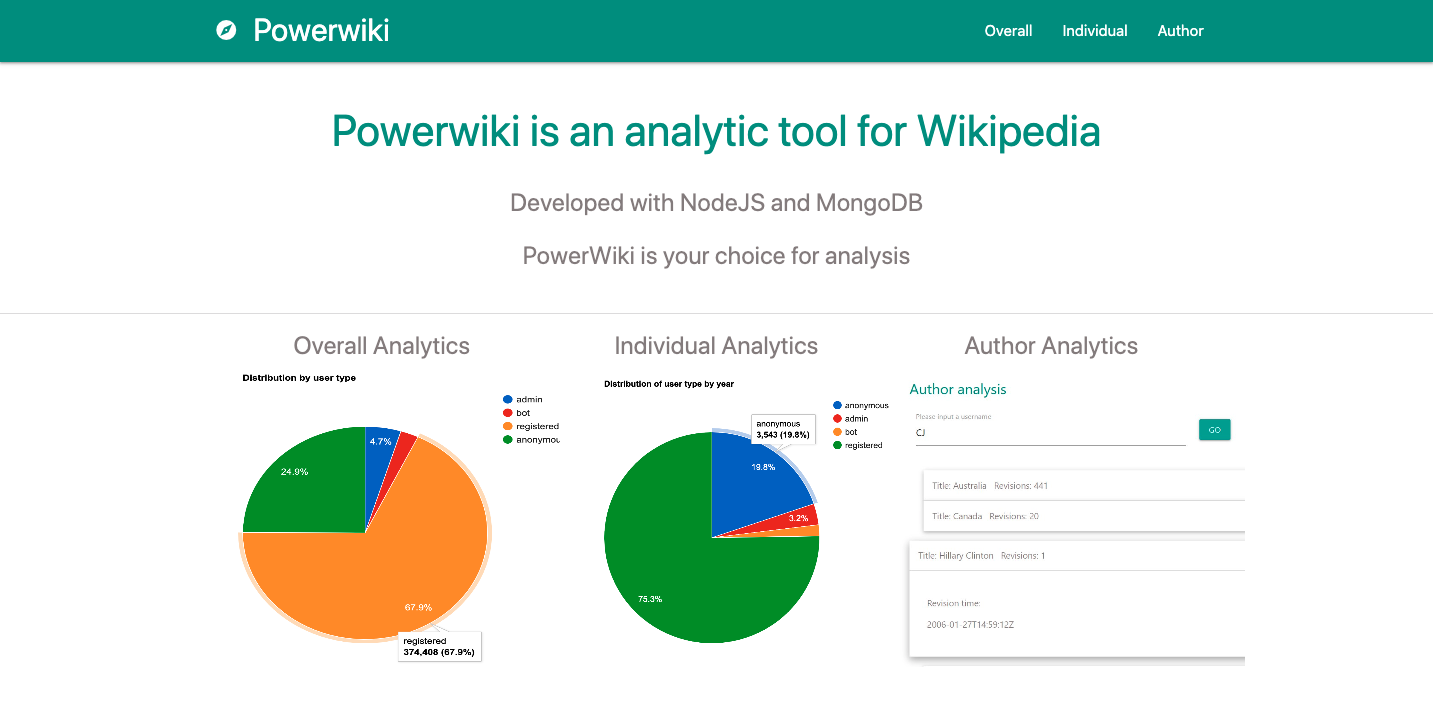


*Figure 1.3.1 Log in page*



*Figure 1.3.2 Reset password modal*

*main.ejs*. After the user logs in, the page will jump to the URL http://localhost:3000/main, where the user will be able to access the analysis function of the webpage. the analytic page is mainly composed of four parts, namely welcome page, overall analysis, individual analysis And author analysis. The display of these functions is also realized by tabs written by javascript. Users can click the link on the navigation bar to let the page display different functions. The page is shown in Figure 1.3.3.



*Figure 1.3.3. Analytic page*

### 1.3.2 Controllers

We implement most functions that process the requests and database operations. Every request sent by the front-end scripts will go to the router, then redirected to corresponding controllers.

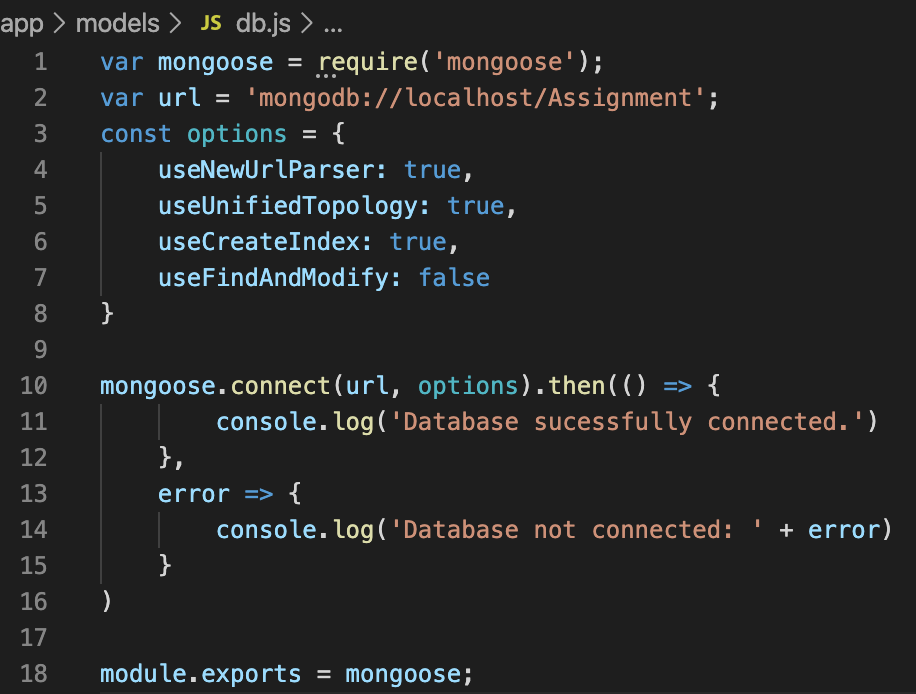
We split the controller into two parts: one for login-sign-up page and the other for analytics page. In index.controller.js, we implemented the login, sign-up and reset password function. In analytic.controller.js, we implemented all the analytics function and API calls including updating revisions from Wikipedia and getting news from Reddit.

The controller receives JSON objects from the front-end script as DB query parameters and returns the query result in JSON format so that front-end scripts can easily handle the data.

### 1.3.3 Model

The model layer is mainly responsible for interacting with the database management system. The database being chosen is MongoDB, which is a type of NoSql. It has the following features: 1) High Performance. It provides high performance on data persistence, e.g. the support for embedded data models reducing the I/O operations and the index for faster querying. 2) Rich Query Language. It provides a rich query language for CRUD operations, especially the Data Aggregation. Data Aggregation has been applied throughout the whole model. There are also other useful features such as High Availability, Horizontal Scalability as well as the Support for Multiple Storage Engines. In order to use MongoDB in node.js, the module called mongoose is recommended. Mongoose provides a straight-forward, schema-based solution to model the application data.

The model consists of two parts, one is the db.js for connecting the MongoDB, another one is the models.js for modelling user registration and revisions records.

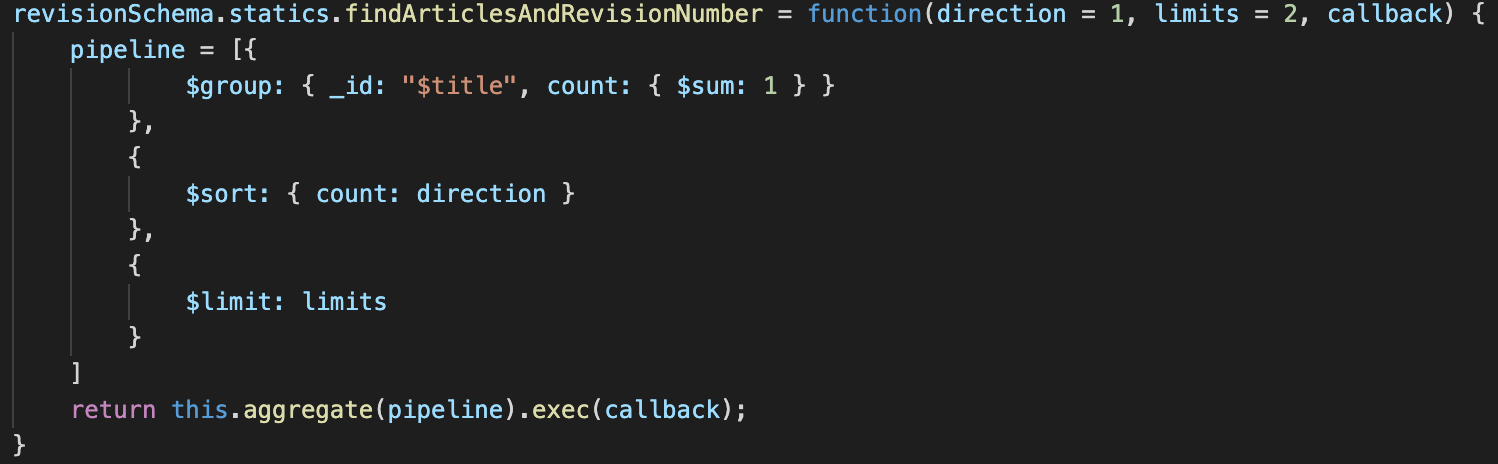


*Figure 1.3.4 db.js connecting database*

After connecting to the database, models.js created two schemas, the userSchema for storing verifying user information and the revisionSchema for storing and manipulating the revision history.

|  |  |
| --- | --- |
| *Figure 1.3.5 user schema* | *Figure 1.3.6 revision schema* |

Both model schemas have several statics methods. For example, userSchema can implement user sign in, log in and reset the password. The revisiosnSchema is the main object for our application, therefore, it has many methods related to the functional requirements of this application.



*1.3.7 revision schema statics method*

Noticeably, most of these statics methods use the Data Aggregation as mentioned before. It looks like the pipeline style shown above. Mongoose will handle the request step by step. In 1.3.4, the revision data would be grouped at the beginning, then sorted in either descending or ascending order and finally output a number of results as specified by the limits.

# 2. Design Decisions

## 2.1 Async database query

In JavaScript, non-blocking codes are executed in asynchronous ways, which greatly improves the efficiency of the server. However, there are some situations where we want some code to be executed after other operations. We do not want to use callbacks for its complexity and potential callback hell. Therefore, we chose the promise which is introduced in ES6 standards.

For the operations that should be executed in async ways, we put them in an async function and return a promise. After the DB calls, we use the “then()” method to return the data to the front-end side.

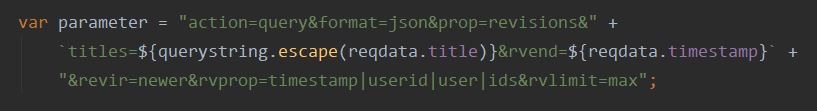
Figure 2.1.1 shows one example of utilizing the promise mechanism.



*Figure 2.1.1 Handling synchronous execution with promises*

## 2.2 SQL injection prevention

To prevent potential SQL injection, we use the “querystring” module to escape the input string as shown in Figure 2.2.1



*Figure 2.2.1 Escaped string*

## 2.3 Divide revisions of 4 user types

As specified in the requirements, the user, i.e the author or editor, of revisions could be a bot, an administrator, a regular user or an anonymous user. But the original revisions history has not explicitly specified the user type of each revision(anonymous user is indicated as anon: true). Since the bot and administrator lists been provided, it is natural and reasonable to specify the user type of each revision beforehand.



*Figure 2.3.1 Update revisions for user type*

## 2.4 Security

Considering user privacy and their account safety, their password should not be seen directly in the database. The encryption should be applied. Hence, the module ‘crypto’ is used to encrypt the user’s password.

|  |  |
| --- | --- |
| *Figure 2.4.1 Sign In* | *Figure 2.4.2 Sign Up* |



*Figure 2.4.3 User Info in Database*

As 2.4.3 shows, the test user has the plaintext password as the registered user has the hashed one.