****

**Team 31 – Design Document**

Pooja Tewari, Shivangi Chand, Siddharth Dhar, Sripath Mishra, Zachary Moore

**Index**

1. Purpose 2
2. Design Outline 4

* High Level Overview
* Flow of Events

1. Design Issue 7

* Functional Issues
* Non-Functional Issues

1. Design Details 12

* Class Details
* Database Schema Mockup
* Rest API Routes
* JSON Web Token Authentication
* Docker
* Hosting
* CD/CI Pipeline
* UI Mockups

**Purpose**

Taking notes in class and keeping them organized can be difficult; sharing those notes amongst friends, classmates and any other prospective users and getting feedback can be beneficial to both parties in terms of the quality of the information that is being exchanged. To simplify this, our project aims to develop a web app that would allow students to create and organize notebooks on their personal accounts in addition to allowing other students to subscribe to him/her, allowing them to provide feedback to these notes. Notebooks will be readily available across all web-enabled devices.  
   
Our functional requirements include the following items:

1. Users  
   a) To be able to register for a Carnet account.

b) To be able to login and manage my Carnet account.

c) To be able to reset my password if I forget it.

d) To be able to easily navigate to a personal Notebook.

e) To be able to open any page of my notebook

f) To be able to easily access Carnet across all of my web-enabled devices.

g) To be able to create new notebooks.

h) To be able to create new pages in any notebook.

i) To be able to insert text in the notebook.

j) To be able to update text in the notebook.

k) To be able to delete text in the notebook.

l) To be able to beautify text in the notebook.

m) To be able to insert pdf as images in the notebook.

n) To be able to update pdf images in the notebook.

o) To be able to delete pdf images in the notebook.

p) To be able to insert text over pdf images.

q) To be able to update text over pdf images.

r) To be able to delete text over pdf images.

s) To be able to beautify text over pdf images.

t) To be able to save any changes made on the notebook.

u) To be able to edit notebooks after they have been saved.

v) To be able to organize the notebooks in folders.

w) To be able to share my notebook or make my notebooks public.

x) To be able to make my notebooks private.

y) To be able to search other public notebooks by their titles, owners, etc.

z) To be able to like, dislike and linear comment about a notebook.

aa) To be able to subscribe to another user or any updates on a notebook.

ab) To be able to edit any notebooks which I have been given access to by the owner.

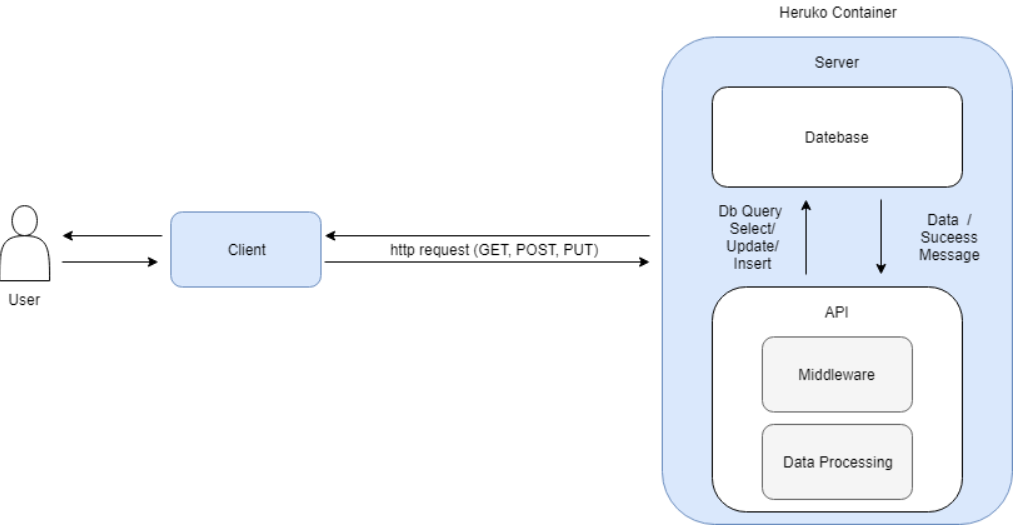
ac) To have a way to connect to the developers for providing feedback

1. Software Developer   
   a) To have a continuous development and continuous integration pipeline.

b) To have a Docker container for local testing

**Design Outline**   
   
**High Level Overview**

Carnet will be using a restful API and client model. The client will be used to successfully interact with our server and to beautify the response received from the server. The Client will be using react.js to decipher the response received from the server, display the notebook the user is currently working on, etc. On the other hand the server will process the clients request and securely interact with the database.



**Server**

Server receives http requests such as GET, POST, PUT.

Server interacts with the database to insert new data, update old data, and delete expired data.

The server securely processes the data received from the database and the client

The server sends out a customized response to the client from which it received the http request.

The server authenticates and authorizes the user and prevents any malicious activity.

**Client**

The client sends out http requests to the server.

The client processes the data from the user before sending it to the user if required.

The client ensures that the user is authorized and authenticated until the session or 10 hours of continuous work is over. This will be done by renewing the client, which will be done under the hood by the client.

The client will present the data received from the server in a designed manner, which would allow easy interaction with the server.

**Database**

The database stores the data about a user account and notebooks in multiple tables.

The database provides a quick and efficient method for data retrieval.

**Flow of Events**

A typical session (entire interaction between the user and client) which encompasses every feature provided by Carnet will be as follows:

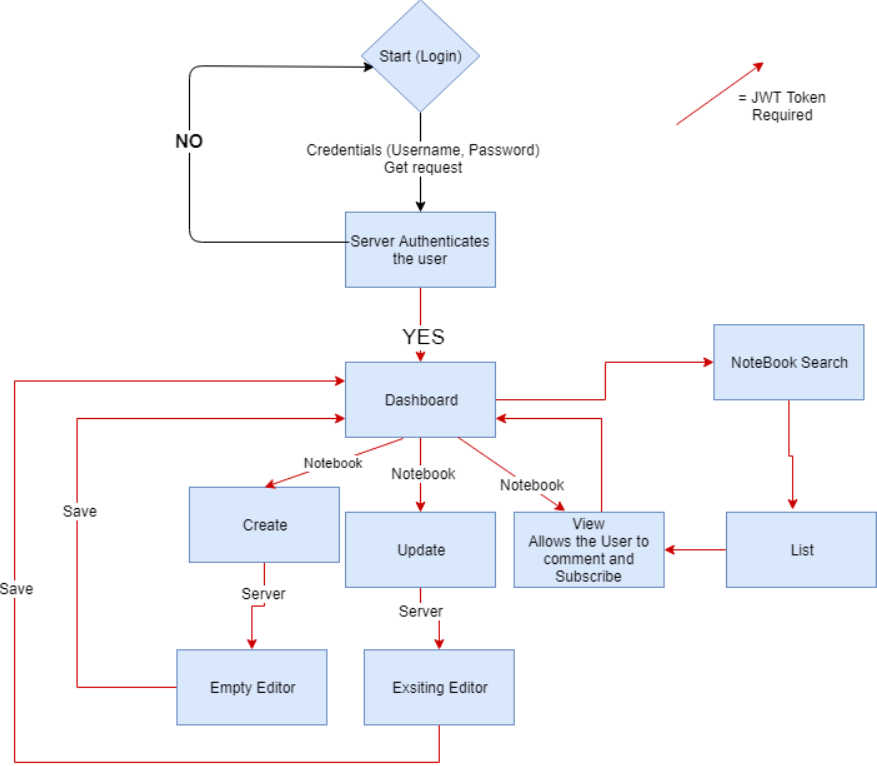
The user will login/create his or her account from any web enabled device. The client under the hood will interact with the server to determine if the user is authenticated and authorized.

After successful authentication and authorization, the user will create/ update a notebook, while the client under the hood will get the data about the notebook and send the final version of the notebook to the server when the user exited the notebook. The server will end the data about the notebook and acquire the most version of the notebook.

The user will search for other public notebooks, users, and subscribe to a particular user. The user will comment on a particular notebook. Under the hood the client will interact with the server to search users, notebook, and will update the server if the user has commented on a particular notebook.

If user 1 is an owner of a particular notebook, and user 2 has subscribed to user 1, then if user 1 updates his/ her notebook, then user 2 will receive a notification about the same from the server through the client.

During this session the client will process the data received from the server, if an error arises it will notify the user of the error. If the preprocessing of the data is successful, the client will send an http request to the server. After receiving the request from the client, the server will process that data, make database queries if required, authenticate/ authorized the user if requested. After the aforesaid steps if an error arises the server will send the errors to the client in response, send a successful if it was a POST, PUT request, send the requested data if a GET request.



**Design Issues**

**Functional Issues :**

Issue: Should the User be required to create an account and log in to view notebooks?  
 **● Option 1:​Yes**

● Option 2:​No

Security is an important part of our software we will have to ensure that the user has created an account and will be logging in to access the notebooks. This will allow us to prevent any malicious activities. Another advantage would be that we will be able to provide customized dashboard to the user by tagging data to the userID.

Issue: Should the User be able to insert pdfs as images?  
 **● Option 1:​Yes**

● Option 2:​No

One of our goals is to Provide the best tools to enable the user to take notes for his/her benefit. In today’s classrooms, lectures are provided through presentations which are in PDF format. By allowing the users to insert pdf as images the user would be allowed to mark on those images and write over them. Please note that the user would be uploading a PDF and we will be converting them to images to attach them to the notebook. We choose the PDF format as it is an universal format and any document can be converted to a pdf very easily.

Issue: Should the User be able to comment on notebooks?  
 **● Option 1:​Yes**

● Option 2:​No

One of the best ways to gauge the quality of the notebook is to have it reviewed by the public. Allowing users to comment on the notebook will allow any new user to get an idea about the advantages and the flaws about that notebook without opening the notebook. The comments can also be used to point the user to a better notebook. Another feature which commenting would allow is the chance for a user to improve the quality of his/her notebook.

Issue: Should the User be able to update subscribed notebooks?  
 ● Option 1:​Yes

**● Option 2:​No**

As seen in various sites such as Wikipedia. Allowing any user to update pages can be sometimes harmful. If we allow the user to update any subscribed notebooks then the updates made by the user might not be desirable by the owner of the notebook. To prevent such cases the team has decided that users would only have read only access to their subscribed notebooks.

Issue: Should the User be able to update notebooks while being offline?  
 **●** Option 1:​Yes

**● Option 2:​No**

It is reasonable to assume that the user would not have access to internet connections everytime they are trying to take notes. But, If we allow offline updation of notebooks then the user might have to download large sums of data which is undesirable. Another disadvantage is the problem with syncing notebooks. In offline case one process would be used by the client to keep on pinging the internet to check if the internet connection is back up. This would lead to a undesirable loss of energy. If time permits we will allow the users to work in offline mode. But, the current decision is that the team will not allow the student to update notebooks while being offline.

Issue: Should the User be able to duplicate subscribed notebooks?  
 **●** Option 1:​Yes

**● Option 2:​No**

We will not tolerate nor allow any methods which can be used for plagiarism. All the notebooks will be the property of the owner and we will disable any methods which can be used to copy the owner’s work. If the owner would allow the user, we will provide him a way duplicate the notebook but all the pages will have a mark signalling the use of the owner’s work.

**Non-Functional Issues:**  
   
Issue: What type of architecture should we use?  
 **● Option 1:​Client-Server Architecture (Rest API)**

● Option 2:​Client-Server Architecture (Sockets)

● Option 3: Unified Architecture

As we are web software, we researched and found that having an server-client information will be the best option for us for the following reasons: Firstly, it will be less data stored on the client side as the data about notebooks will be stored with us. It will allow users to use any web enabled devices to access those notebooks. Secondly, it allows users to search and view public notebooks of other users. Thirdly, It will allow us to be independent of the OS on which the software is run on as we will be running on the web browser. Having a unified architecture would have been disadvantageous as we would not have been able to differentiate our product from any of the current products such as Microsoft Word, Onenote etc. Finally, having a server-client implementation will allow us to divide and conquer as the front-end team will not wait for the backend team to finish creating any route. Parameters can be discussed between the teams. We decided to go with Rest API as it would allow us to end a connection and process request from other clients. Keeping up an connection up is not required as server will not be initiating any connection to send data to the client.

Issue: What back end framework should we use?  
 ● Option 1: Python (Django)  
 ● Option 2:​Java (Spring)

● Option 3:​Java (Spark Java)  
 **● Option 4: NodeJS (Express)**

We researched about frameworks to use for the backend. Django is an old framework which is an heavy framework. We needed a light framework as the server will only be used to some basic data processing and getting Django seemed a little too much for this project as it will accompany with a huge learning curve. Java Spring framework is an amazing framework but it is heavy and setting it up takes time. It also downloads other features which will not be used. So it was eliminated due to being heavy as heroku allows only 500MB of code base along with the dependencies in the free version. Spark Java framework is a lightweight framework but is only accessible through maven. Another disadvantage was the language JAVA as the team wanted to work with NodeJS. NodeJS is the perfect fit for us. It allows us to have dependencies which we will consume completely in our code base and it has a internal framework which is lightweight and highly customizable.

Issue: What database software should we use?  
 ● Option 1: Microsoft SQL  
 **● Option 2: Postgres**  
 ● Option 3: Mongo

We wanted a relational database due to which Mongo was off the options we had. We are determined to have a fast and scalable database. Microsoft SQL is a fast database but we had no free version in the Heroku. Another important feature that Postgres other than being fast is that it's highly scalable. It is very similar to SQL. Another feature that Postgres has is the ability to use vectors for select queries. This feature will be very useful while searching for public notebooks. Postgres also has an frequently updated docker image and its free as compared to Microsoft SQL. We did research on other options but they were either not fast or scalable.

Issue: How should we handle user permissions?  
 ● Option 1: Create a different routes for Owner of notebook and viewer of notebook.  
 **● Option 2:​ Embed permission roles in the JWT tokens.**

We need a way to prevent users to edit an notebook if they are not an owner of that notebook. Creating different routes leads to duplication of code. We want to prevent that. JWT tokens provide the perfect method to ensure that no code is duplicated and no user is able to edit the notebook they are not the owner of.   
  
Issue: How should we authenticate API requests?  
 ● Option 1: Send username/password on each API request  
 **● Option 2:​Use JSON Web Tokens for stateless authentication**  
 ● Option 3: Authenticate with an OAuth provider.

Authenticating and Authorizing users is an important process. We want to make sure that there are no malicious activity. Sending username and password with every call is slow and unsecure. As the server will have to execute a query on the User table to check if the credentials are correct. Using OAuth providers like google or college systems is an amazing option but its disadvantageous to the subset of people who are not a part of College/Organization or who don't have a google account. JWT tokens is the best option. If the client passes the token with their requests, they can be authenticated and authorised in less time. Another advantage would be store additional data about that user which can be used by the server when future request calls are made. Therefore we decided to go with JWT tokens.

Issue: What front end framework should we use?  
 ● Option 1:​Angular  
 **● Option 2: React** ● Option 3: HTML/CSS

● Option 4: Vue.js

The HTML/CSS is the most basic option but it would require us to do most of the heavy tasks which can be troublesome and foolish. All the other options are very similar as they provide set of tools to create a robust frontend. But React has an additional feature which allows us to port the web application to ios/android apps if the time permits. Due to this and the interest of the group in leaning React we decided to use React for frontend development.  
   
Issue: How should users navigate through the Site material?

● Option 1:Sidebar  
 **● Option 2: Main menu bar at the top**

We have numerous pages to which the user can browse though. Stuffing all of them into a sidebar can be sometimes a little crowded. And accessing self hiding sidebars though an mobile device browser can be difficult. A famous example of the bad design can be seen by accessing the BlackBoard from a small screen. The user is unable to view the sidebar and the mainpage simultaneously. Therefore we will be displaying the other pages at the top of the page through a menu bar.   
  
Issue: How should users receive notifications about important Notebook updates?   
 ● Option 1:​Email  
 **● Option 2: Badges**

We will be using badges as the primary way to notify the user of a subscribed notebook updates. If time permits we will be implementing Email notifications as well. But the reason behind using badges is that it is the best way to ensure that the user has received the notification. As we are not in control of the spam filters of the user’s email account.

Issue: Where should we deploy our server?

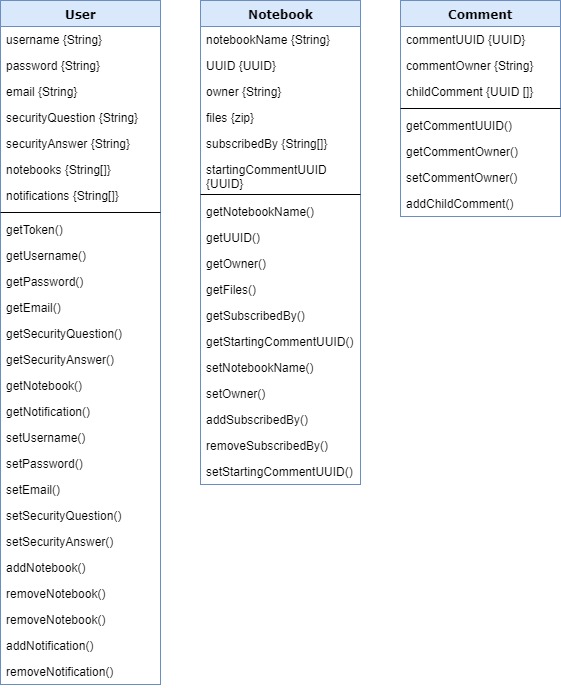
● Option 1:​Local Server  
 **● Option 2: Heroku** ● Option 3: AWS

Setting up a local server can be expensive and has to be monitored for unexpected downtimes. AWS (Amazon Web Services) is an amazing tool for deploying our server but it’s only free for six months. Heroku is the best option for free hosting of servers as it provides enough resources to deploy a small scale server along with the database. Heroku also allows a staging and production server deployment for testing. Therefore we will be deploying our server on Heroku.

**Design Details:**

**Class Details:**

As shown in the figures below we will be having three classes. The user class will deal with the user objects. It will be storing the user details as shown in the figure. The notebooks will be used to connect to the Notebook class. Notebook class will contain all the details about a notebook. The third and final class is the comments class. It will contain all the details about a comment made by an user on any notebook. One point to note is that the three classes are interconnected. The user has notebook objects which will contain all the public and private notebooks owned by the user. Another interaction will be between the notebook class and the comments class as the notebook class will hold details about the starting comments on that notebook.The user class can interact with the comments class as the user can find all the comments made by that himself/herself. More details about the interaction between these classes can be found in the flow diagram and the details provided in the Flow of events.

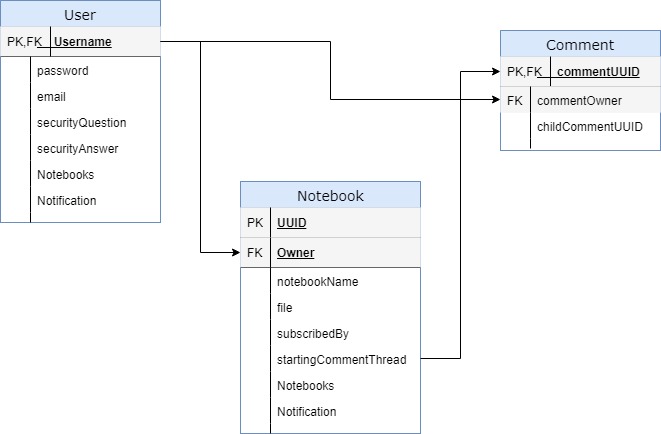
****

**Database Schema Mockup**

Description of database tables-

We will be describing three tables in Postgres. The first table will be used only to store the user accounts and details while the second table will be used to store data about a particular notebook. Finally, the third table will be used to store the comments of a particular user about an notebook. Schema might change in the later. We will also be using the vectors feature of Postgres to speed up the query speed.

1. User : This is the only table containing details about the user. It will contain the following data about any user:
2. The username of the user which will be unique and will be used as an primary key.
3. The hashed password of the user which will be encrypted by Bcrypt which uses blowfish algorithm.
4. The Email ID of the user which will be verified if the time permits to implement it.
5. The list of UUIDs of the notebooks which are owned by the user.
6. The security Question of the user in case of forgotten password.
7. The lowercased hashed answer to the security question provided.
8. Notebooks: This table will be containing the data. It will containing the following data:
9. The username of the owner of the notebook.
10. A unique UUID which be used to identify a particular notebook.
11. The Zip file containing the notebook data (pages, images, text, etc.)
12. The subscribers of the notebook (usernames).
13. The comment thread UUID.
14. Comments: This table will accrue the comments given to a particular notebook. The main reason that we are not putting it in the Notebooks as it will be easier to implement thread comments as compared to linear commenting. The following data will be stores in a particular row.
15. The UUID of the comment.
16. The user who owns that comment.
17. The child comments if time permits to implement thread commenting.



**Rest API routes**

The following API routes will be defined for successful and complete interaction between the client and the server:

|  |  |
| --- | --- |
| Routes | Http Protocol used |
| /get\_token | GET |
| /register | POST |
| /reset\_password  /user | PUT  GET |
| /user\_notebooks | GET |
| /comment | GET |
| /comment | POST |
| /comment\_child | POST |
| /notebook | GET |
| /create\_notebook | POST |
| /notebook | PUT |
| /search\_notebook | GET |
| /subscribe\_notebook | POST |
| /notificatiosn\_user | GET |

These routes will be implemented. Some routes will be exclusive to admins and notebook owners. More information about the base URL, parameters and headers will be decided at the time of implementation and will be visible in the API documentation.

**JSON Web Token Authentication**

As discussed in the design issues JWT (JSON web tokens) will be used to authenticate and authorize the user. Security is an important feature in our software as we have to protect private notebooks of an user and we also have to ensure that the user is not able to edit an notebook if he is not the owner of that notebook. JWT tokens will help us in ensuring about the same. The JWT tokens will work in the following manner:

1. The user will pass his credentials(username and password) from the client and the API will receive a GET request.
2. The Server will hash the password and match it with the password stored in the DB along with the username.
3. If the credentials are missing or invalid then an error will be returned. Upon successful authentication, the server will return a JWT which will be containing the user name and the list of notebooks the user is owner of and in a diff list the list of subscribed notebooks. This will allow us to minimize database query if a request arrives from the user about a notebook he is not authorised to edit or view.
4. The JWT will be valid for 5 mins after which the client (under the hood) will call another GET route and get an updated token for 10 hours after which the user will be forcefully logged out.
5. Any route accessed by the client should contain the token and the server will contain a middleware which will test the token’s validity and will prevent any unauthorized or unauthenticated requests to move any further.

**Docker**

One of the most important problems faced during developing and deploying servers is the process of setting up the local system to support. Having environmental variables on the Local system also makes the code vulnerable. To ease the development process we will be incorporating Docker. This will take the code base into an closed container with only one exposed port. Then the requirements and database will be set up in that closed container and will remove and local time setup. This also helps us with one of the important rules of software engineering, i.e. to have the build and deployment in one single line command. In this case it will be “docker-compose up.”

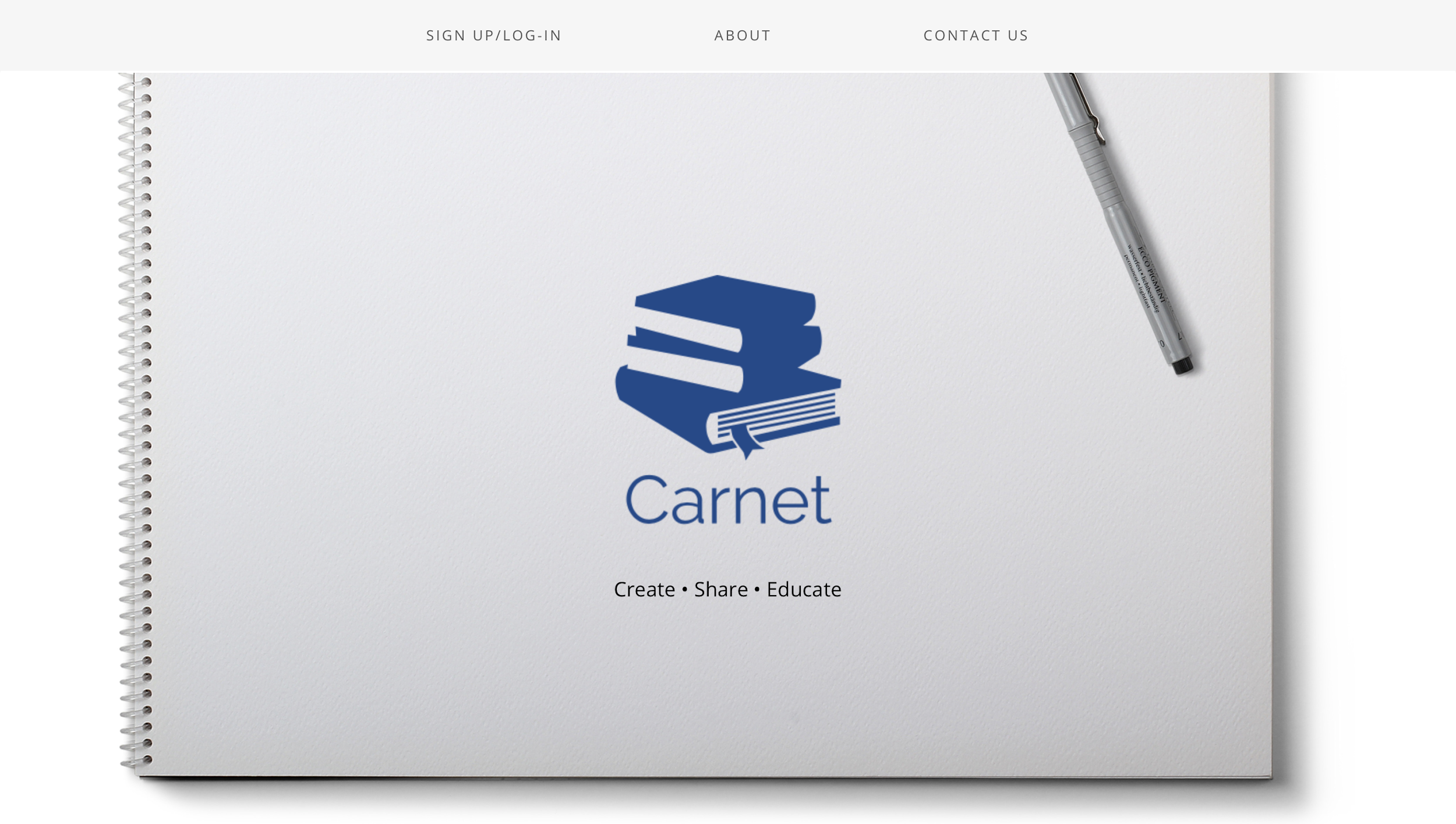
**Hosting**

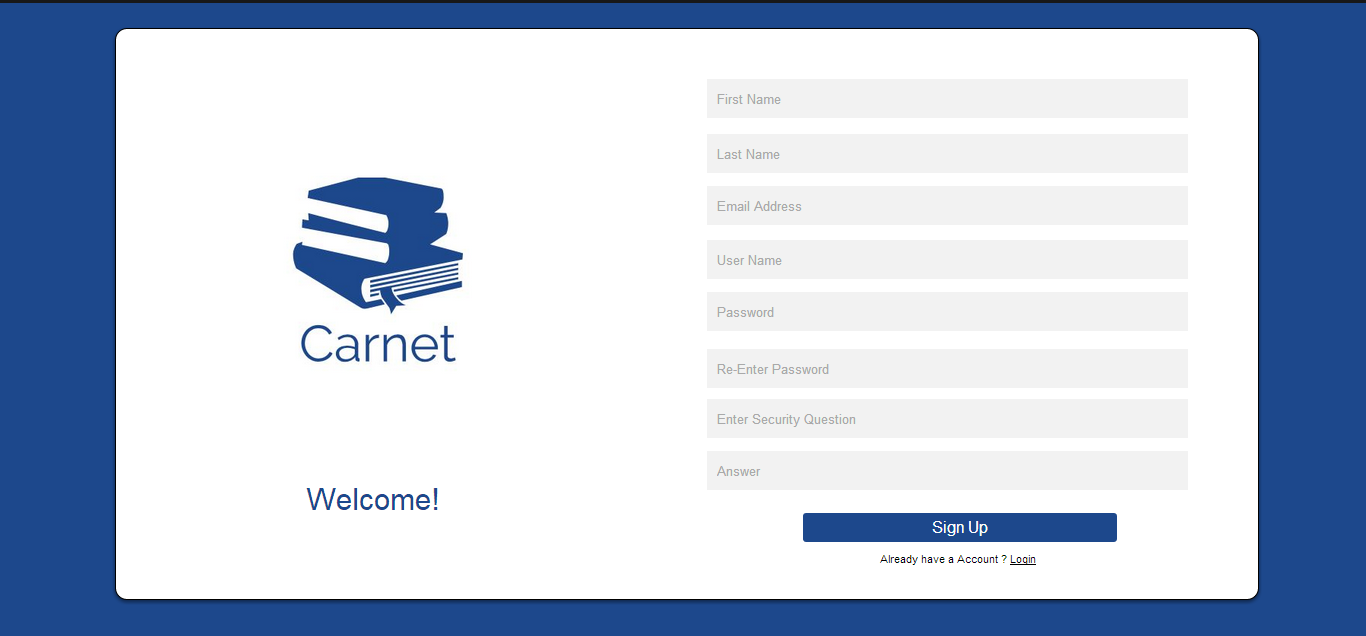
As we are using an Client-Server implementation we will be hosting the server on Heroku. We will be using the Free version of Heroku which will allow us to receive a max of 10,1000 request call which should be more than enough for testing and small deployment. Another useful feature is that it allows two deployments. One can be used for testing and staging and with a click of a button we can promote it to be in production. Heroku has free Postgres instances available which can store a maximum of 10,000 rows of data which should be more than enough for this project during the development.

**CD\CI pipeline**

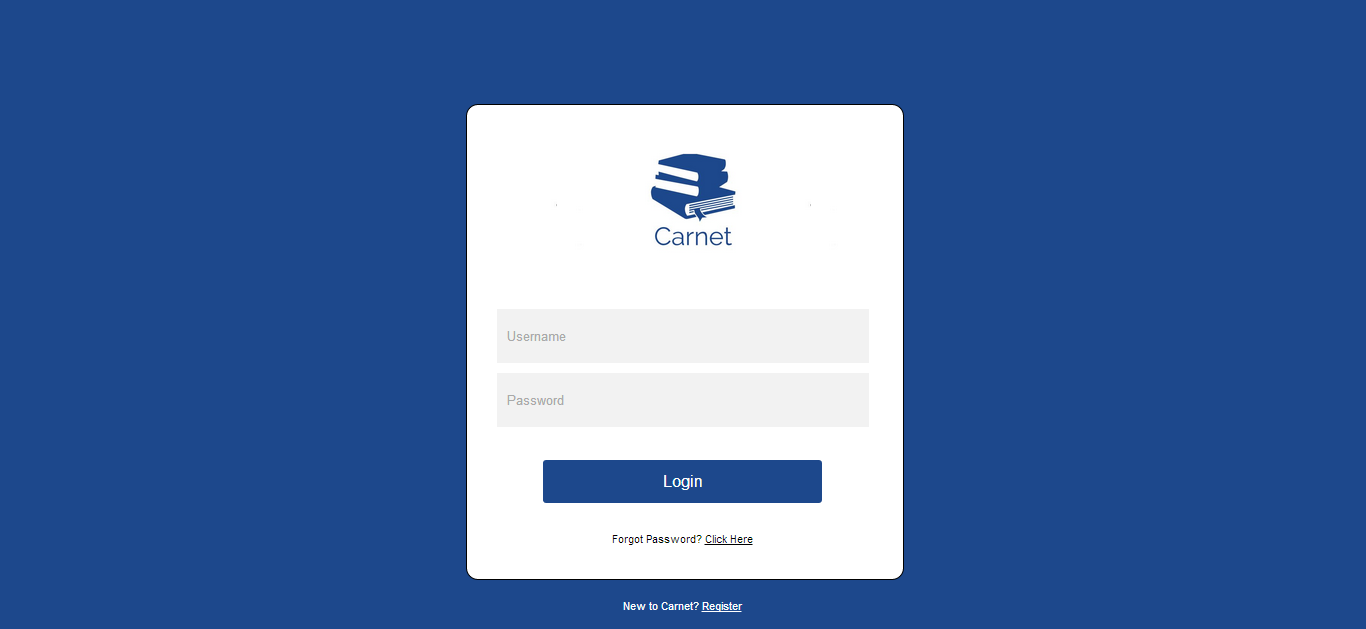
In today’s world building fast can create problems and It can take minutes to completely test the whole code every time. Another problem is that there is no way to ensure if the developer is following all linting rules. Another problem can be the hassle related to deploying on heroku. These are the main reasons to implement an CD/CI pipeline (Continuous development/Continuous Integration). We will be using CIrcleCI to implement this. CircleCI creates a hook to a Github repository. This way whenever a push is made on the remote repository, It will run test cases on that branch, ensure all linting rules are followed. If changes are made to the master branch then an additional script will be executed by the circleCi which will push the code and deploy it on heroku.   
   
**UI Mockups**

We have designed the UI to be visually appealing and enhance user experience. When the user enters the url, the user will be directed to the landing page, from which the user can navigate to the login and sign up pages. Once the user logs into his/her account, the dashboard with an array of options is displayed. On the dashboard, users can check their feeds since the last time they logged in, upload notes from their web-enabled devices, remove notes from the website and check the list of subscribers following them, search and comment on public notebooks.In addition, users can have private and public collection of notebooks. Users are also provided with an option to reset their passwords if need arises.

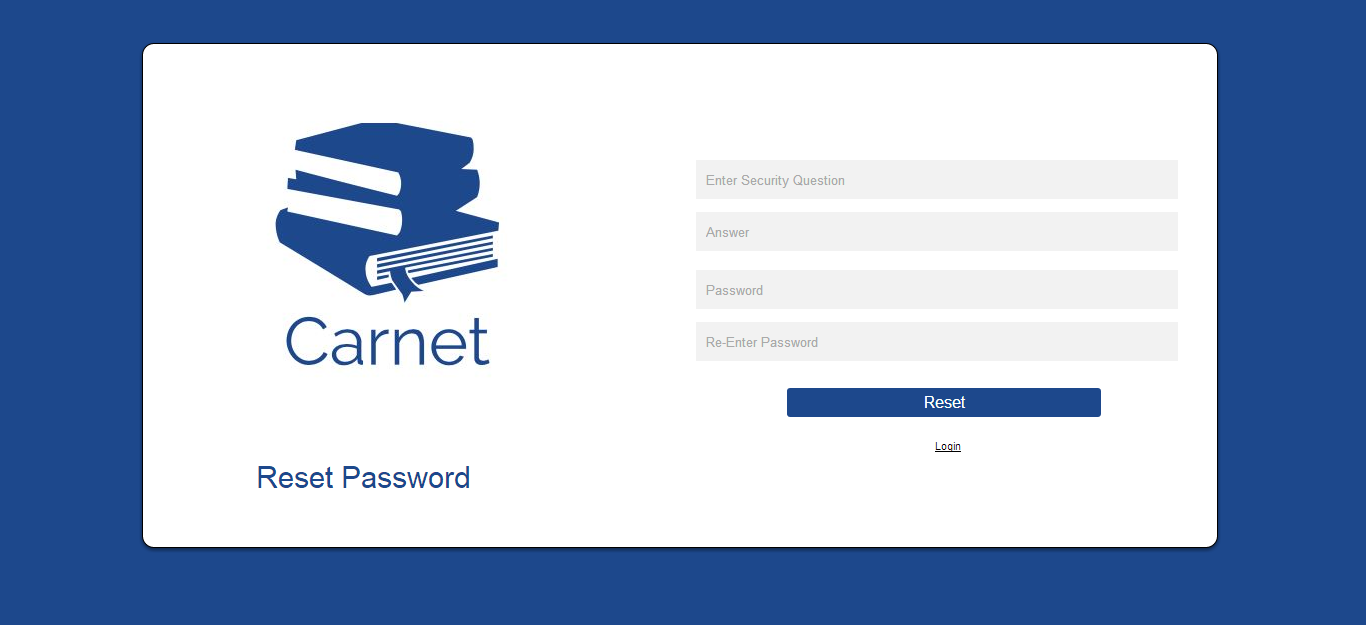
  
  
This is the landing page, from here the user can navigate to the sign up and login pages. The user can also navigate to the pages telling about us (Carnet) or the contact us page.



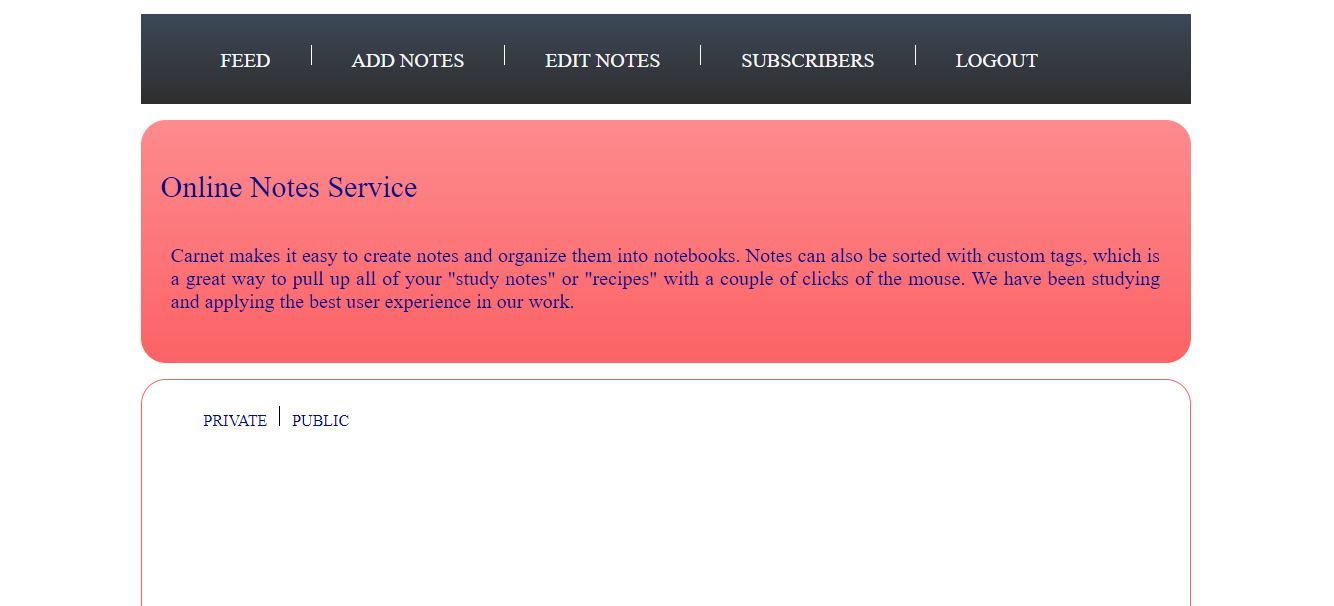
This is the sign up page, where we ask for the user’s credentials to create his/her account



This is how the login page looks like.



If a user ever forgets his/her password then the user can reset their password, this is the page that we would use for resetting the password.



The Dashboard is the page, where the user will directed once they enter their correct credentials. From the Dashboard the user can access their private and public notebooks, search for other users and notebooks.