Design Document

VENote, Team 14
Killian Le Clainche, Jarett Lee, Aaron Neustedter, Mehul Patel, Chad Stucky, Ryan Sullivan

Purpose

We will design a system that enables engineers (our users) to store their ideas in a manner that will be useful in later turning those ideas into legally bound patents. This system will consist of a front end web interface written using a modified MEAN stack (MEAN + Javascript, JQuery, CSS/Preprocessing, Facebook React and Redux libraries) and a back end server that will handle requests via a mixture of Node.js and Firebase. In order to achieve the most functionality between the two layers of our design, we will use an API to communicate over https between the two.

Functional requirements

1. **Users can manage users**
   As a manager, I would like to
   a. Assign user permissions to other users
   b. Handle a large number of users

2. **Users can create a notebook**
   As a user, I would like to
   a. Add images to the notebook
   b. Add text to the notebook
   c. Caption pictures in the notebook
   d. See an introductory tutorial

3. **Users can manage notebooks**
   As a user, I would like to
   a. Query all notebooks for a specific notebook
   b. Query all notebooks for a specific timestamp
   c. Query all notebooks for specific text
   d. Create notebook groups
   e. Share a notebook
   f. Export notebooks as PDFs containing 3D-viewers
   g. Back up a notebook to a secondary storage location
   h. Access the notebook from the internet
   i. Access the notebook offline if time allows

As a manager, I would like to

j. View a group of user's notebooks
k. Submit a notebook for legal approval
l. Assign specific users access to certain notebooks

4. **Users can manage notebook pages**

   As a user, I would like to

   a. Sign pages
   b. Know the number of pages
   c. Know the timestamp of pages
   d. Query a notebook for a specific page
   e. Query a notebook for specific text
   f. Query a page for specific text
   g. Choose what formatting and rules are applied to my page
   h. Location to be logged and have the option to keep track of where I edit my pages from.

   As a manager, I would like to

   i. Sign pages

5. **Users can markup a page**

   As a user, I would like to

   a. Make corrections in the proper format
   b. Comment on pages
   c. Know why I can't edit previous pages
   d. Tag notebook pages
   e. Use a paintbrush tool to add custom drawings to the notebook
   f. Upload general files to the notebooks
   g. Upload STL to the notebooks if time allows
   h. Upload FBX to the notebooks if time allows

6. **Users can collaborate with others**

   As a user, I would like to

   a. Share a notebook through email.
   b. Share a notebook through a link.
   c. Share a notebook to another person without an account.
   d. Control read/write access those who I share the notebook with have

7. **Companies can interact with developers**

   As a user, I would like to

   a. Give feedback to developers

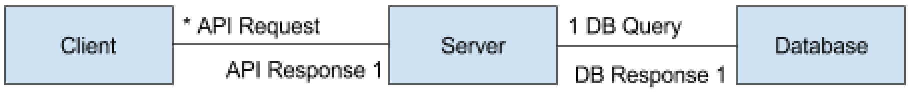
Non-Functional

- Intuitive UI - it should be easier for customers to use our tool that it is for them to use existing tools
- Fast site loading times (under 1 second)
- Handle large usage load
- Handle large number of users
- Secure communication line using https
- Secure communication line using a direct connection
- Developers maintain server backend using Node.js
- Developers maintain a secure database using MySQL
- Developers maintain a querying mechanic to interact with MySQL database
- Developers maintain config file for easy minor tweaks
- Developers alter the server through a centralized location

Design Outline

Design decisions and components

- The project will be created following the client-server model (diagram below):



  - **Client**
    - Responsive web application written using a modified MEAN stack. Javascript, JQuery, CSS (we will consider preprocessing), Facebook React and Redux libraries
    - Will display users their pages, allow users to perform all transforms to pages and engineering notebooks (text edits, image insertions, signatures, legal formatting, sharing, etc), perform client side encryption, and communicate with the back end to coordinate the synchronization of pages to be saved and fetched from the database.
    - Architecture of the client will be decoupled as much as possible.
      - Will strive to follow MVC (model view controller) pattern. In this pattern, 'events' triggered from views will be propagated to the controllers, where business logic will occur and/or update the model.
  - **Server**
    - Thin client
    - Carries out any necessary logic (user auth, determining pages and notebooks specific users have access to, fetch and/or store to the database and/or return data to the POST for specified request from client.
    - Uses HTTP over SSL

        - Code written in Node.js, and Ruby
        - Server API will be built in a restful way so that AJAX calls can be as powerful as possible.
        - AJAX calls from the client will allow it to update data via POST and GET to the server without users having to reload their pages
        -
    - **Database**
        - Stores all models (users, pages, notebooks, corporations, etc)
        - Uses Firebase
    - **Miscellaneous Details**
        - We will aim to make interaction with the server seamless and non apparent to the user, as the main purpose of the two components is to enable users to log in to our tool from any web browser.
        - The Login page will interact with the server by using server side authentication.
        - The page editing page will …
            - post and fetch pages, notebooks, users, organizations.
            - allow users to visually edit pages by adding text, images, links
            - allow users to specify formatting.
            - allow users to share page with specified others.

Design Issues

Functional issues

1. What kind of client will we make?
    a. **Web client**
    b. iOS app
    c. Android app

The group decided to make a web client because entering text is the most important feature for our application and the primary text interface for our target user is through a laptop computer rather than through phones or tablets. Web clients fit best for this usage. While the mobility of an app is also a nice feature, it is not as important. However, we would like to create an iOS client if time allows because the ability to add pictures directly to the notebook is a very important feature and is best done with devices that can easily take pictures. While developing our web client, we will also aim to make the UI mobile optimized in order to facilitate the possible creation of an iOS app later on.

2. How will the users create the documents?
    a. **Form-like input**
    b. Text editor-like input

Our tool is intended to reduce the effort required to format the notebook. In order to accomplish this, having a narrow input field fits with our concept better. It also allows us to store the data in a more simple manner, allowing us to focus on the formatting rather than the UI of the input.

3. How should the pages in the notebook be organized?
    a. **Chronological**
    b. Categories
    c. **Tags**

The pages in the notebook should be primarily organized by time since a key feature of engineering notebooks is seeing the progress of the project. However, the ability to filter pages by certain criteria is also very important for summaries. Therefore, we decided to primarily organize by chronological order, but allow filtering with tags.

4. How should the notebooks be organized in the document?
    a. **List**
    b. Categories
    c. Tags

We considered having a filesystem or tagging system for our notebooks. This allows for a more complex system for organizing the notebooks. However, we opted for a more simplistic version that sorts by recently used and recently changed among other things, but controlled by the programmers rather than the users. This is primarily to keep a clean ui and reduce the complexity of the project. For example, notebook applications like OneNote only list notebooks rather than allowing more complicated organization.

Email notifications

5. Allow multiple editors to edit the same page?
    a. Yes, concurrent editing
    b. No, only one person can have the app open
    c. **No, single editor with a locking system**

We knew allowing concurrent editing to the notebook was relatively important if we wanted to enable people to switch between devices while writing, however, this is a difficult feature to implement, and one that large programs such as Google Drive are just beginning to master. Since this feature is important, but difficult, we looked for a way to make a simpler version. We decided to use a locking mechanism that only allows one editor, but allows the user to switch between devices quickly.

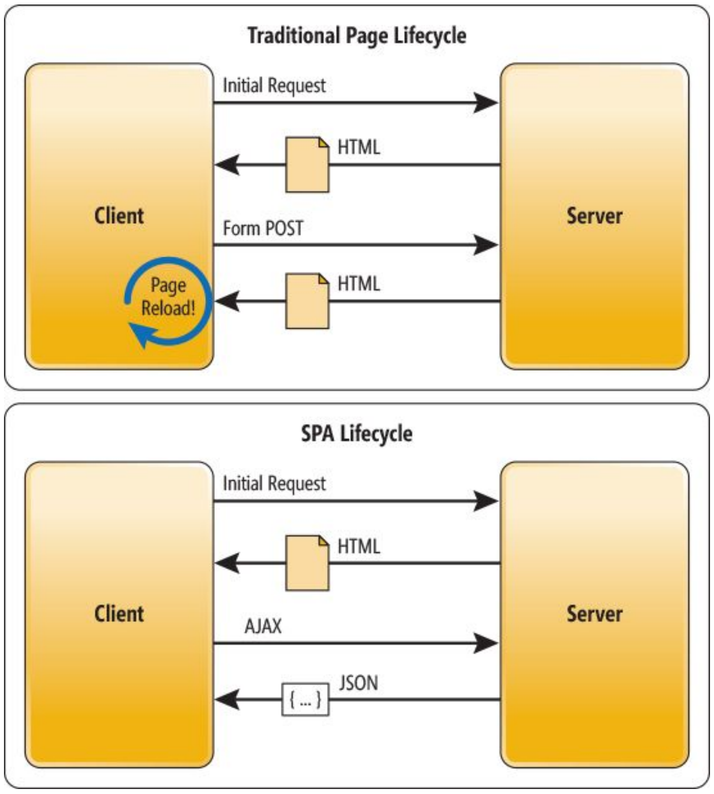
Non-functional issues

6. How should we authenticate API requests?
    a. Use JSON Web Tokens
    b. Use OAuth
    c. Use Purdue CAS

d. **Firebase Authentication SDK**

We decided not to authenticate with Purdue CAS because we envision our system being used by other educational institutions and separate companies. A drawback of using a social network OAuth provider is that many people do not want their personal lives to be mixed with their professional lives, so connecting the two could cause friction with our users. Hoverever, employees could use their work google accounts to login instead. Firebase Authentication SDK is already built into firebase and does not require us to implement it ourselves, while still being secure and easy for the user to use, making it an obvious choice. Firebase additionally allows pure email-password for those who do not have work accounts

7. How many pages should the web client have?
   a. **Single-page ui**
   b. Multi-page ui



Single-page websites load HTML once then update the page using AJAX requests. This removes the need to refresh the page every time the user has to update the page, separates the HTML markup and code, and lends itself to having a server API rather than requiring specific pages for every request.
Source: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

8. What language/framework should we use to write our server application?
   a. **Node.js**
   b. Java + Spring
   c. Ruby on Rails

The team considered Java + Spring because of our experience with Java and the security of Spring, but overall we didn't like using it on past applications. We considered Node.js and Ruby on Rails because both are popular, easy to write, and have a large library. We wanted to learn Ruby on Rails, but considering the benefits of having a JavaScript software stack (server-side and client-side run JavaScript) we chose Node.js.

9. Where should we host our server?
    a. Personal server
    b. Amazon Web Services (AWS)
    c. Purdue server

The team has had experience with AWS before, and it a very stable and scalable platform with many prebuilt web and database systems. However, in order to scale our AWS server we would have to pay. Several of our team members have personal servers that we could host our project on. Scalability would be very expensive and time consuming and if there are other projects running on these servers that could get in the way. In addition, since our personal machines are VPS's, they provide little flexibility in (backend) OS choice. No one has used the a Purdue VPS before, however Prof. Turkstra has offered to set one up for us. In order to get experience using purdue's equipment as well as having a potentially more powerful vps and a clean slate to work off of, we choose to go with the Purdue Server

Decision

10. What language/framework should we use to write our web client application?
    a. Pure javascript
    b. React
    c. jQuery

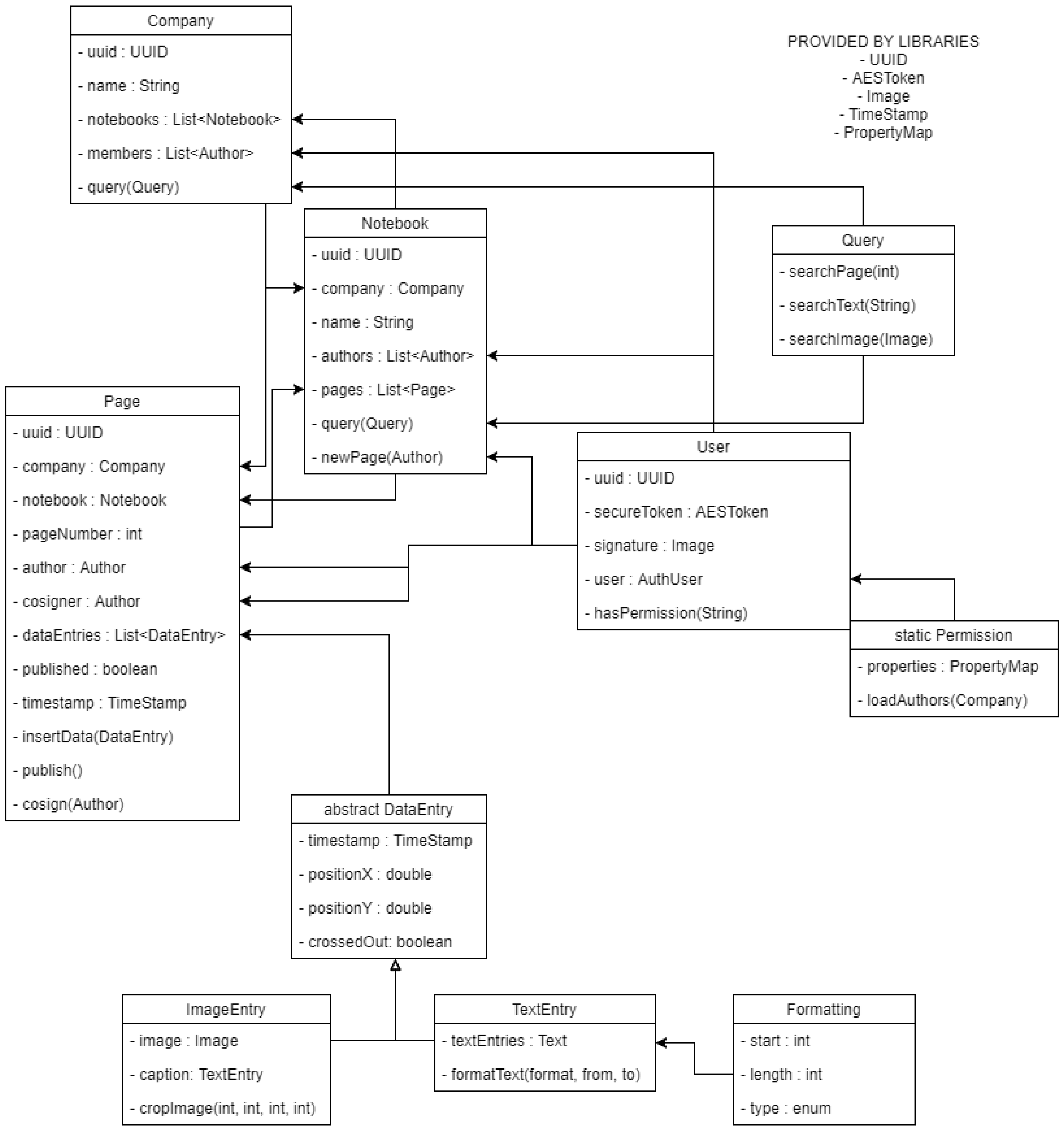
The group is most comfortable with pure javascript in terms of experience, but it takes time to develop using it. Both React and jQuery increase development speed, but we chose React over jQuery because React because the team felt like it would be very valuable to learn. Additionally, React is apparently easier to port to phones, which is a major positive for our plans to create a phone app in the future. In each of these implementations, the languages used would be HTML, CSS, and Javascript.

11. Which database should we use?
    a. MySQL
    b. PostgreSQL
    c. Firebase
    d. MongoDB

We looked at MySQL and PostgreSQL because SQL has a large community with stable tools and standardized format. However, it also requires learning SQL. On the other hand, Firebase and MongoDB are NoSQL and thus is less stable, but they use javascript data structures and thus are easier for our team to understand. Firebase requires basically no setup. Setting up the other databases aren't particularly difficult, but it's more work. Also, as a NoSQL database, we don’t have to learn SQL to use the database. While learning SQL was an option, we chose development speed over learning SQL. This decision was also influenced by Firebase’s built-in authentication system.

Design Details

Class-level design



Company - A company object represents an account for a collection of collaborative users. It contains a unique identifier, a company name, a list of notebooks, and a list of members. A company is able to search its users and notebooks for specific images or text.

User - An account for an individual using the service. It contains a unique identifier, a signature for signing notebook pages, and permission list.

Notebook - The data object representing an engineering notebook. It contains a unique identifier, the company that created it, the notebook’s name, a list of Users that contributed to it, a list of pages added to it. A notebook is searchable by tags, text, or images. Pages can be added to a Notebook.

Page - A page object represents an entry in the engineering notebook. It is created by submitting a new entry through a form. It contains a unique identifier, an author (User), a cosigner (User), a list of Data Entries, and a timestamp.

Query - An object that is used to perform searches on notebooks.

Data Entry - A data object representing a contiguous part of the page. It is created by the client and contains a timestamp and flag to determine if it is crossed out or not.

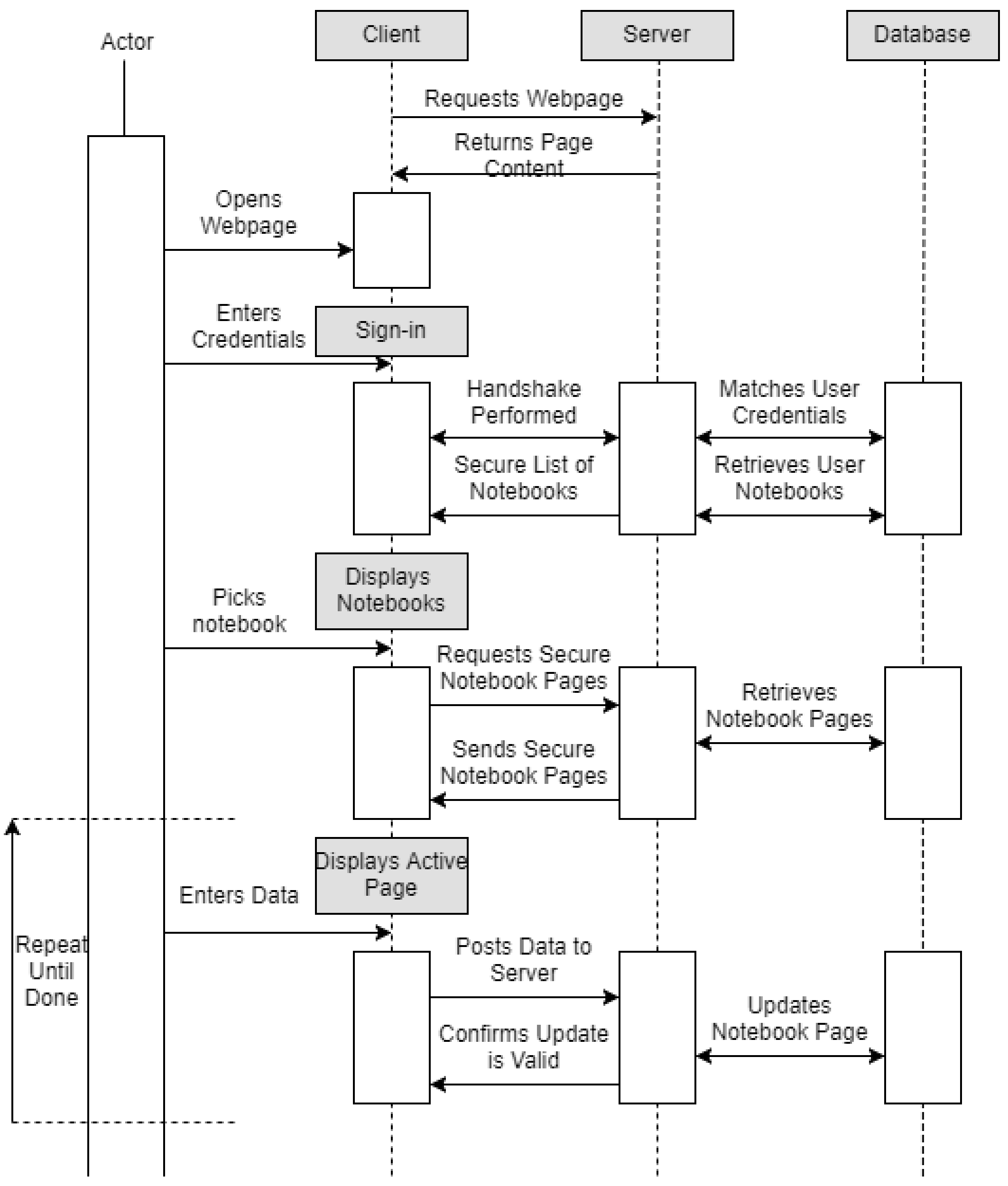
Text Entry - A data object containing text data and formatting data.

Image Entry - A data object containing image data, a caption, and the crop parameters.

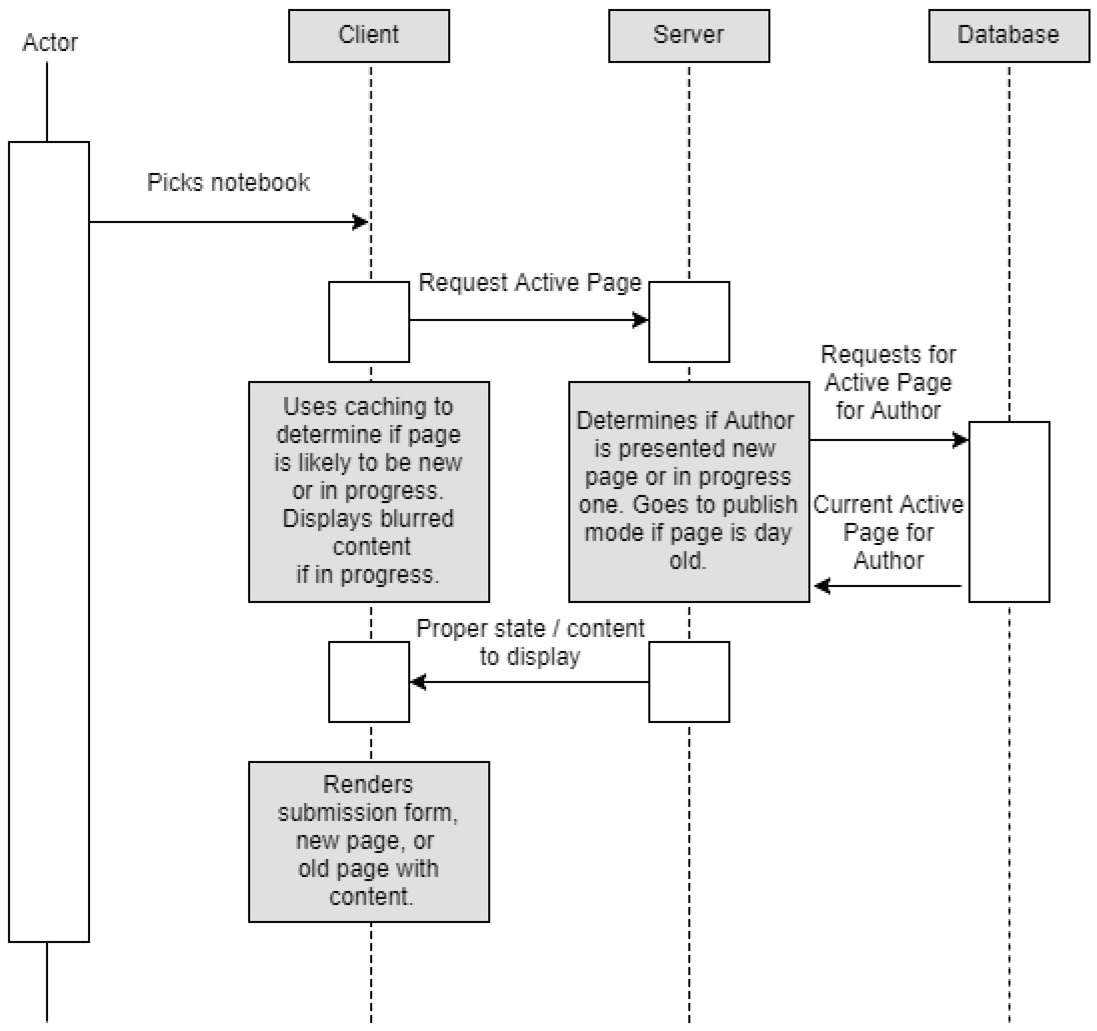
Formatting - A data object that will be sent with the text entry that will contain the formatting options for the text, including strikeout, underline, and bolding.

Sequence diagrams

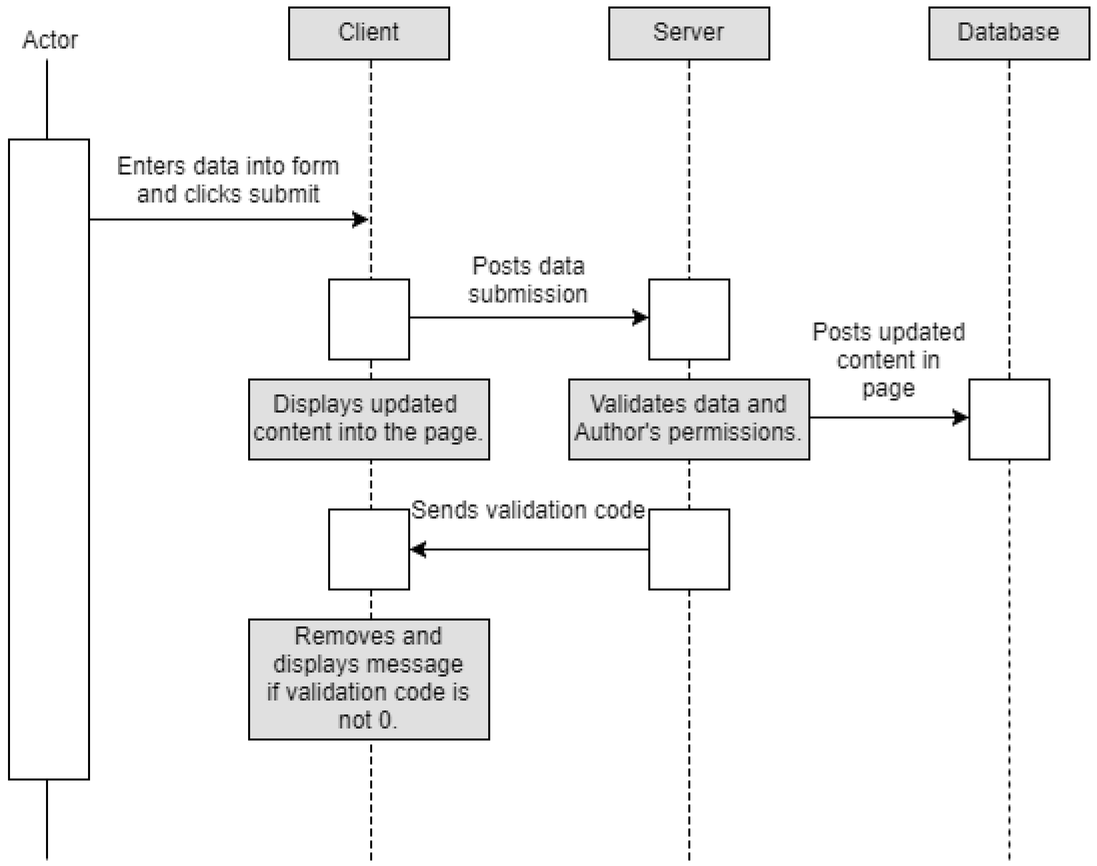
Overview



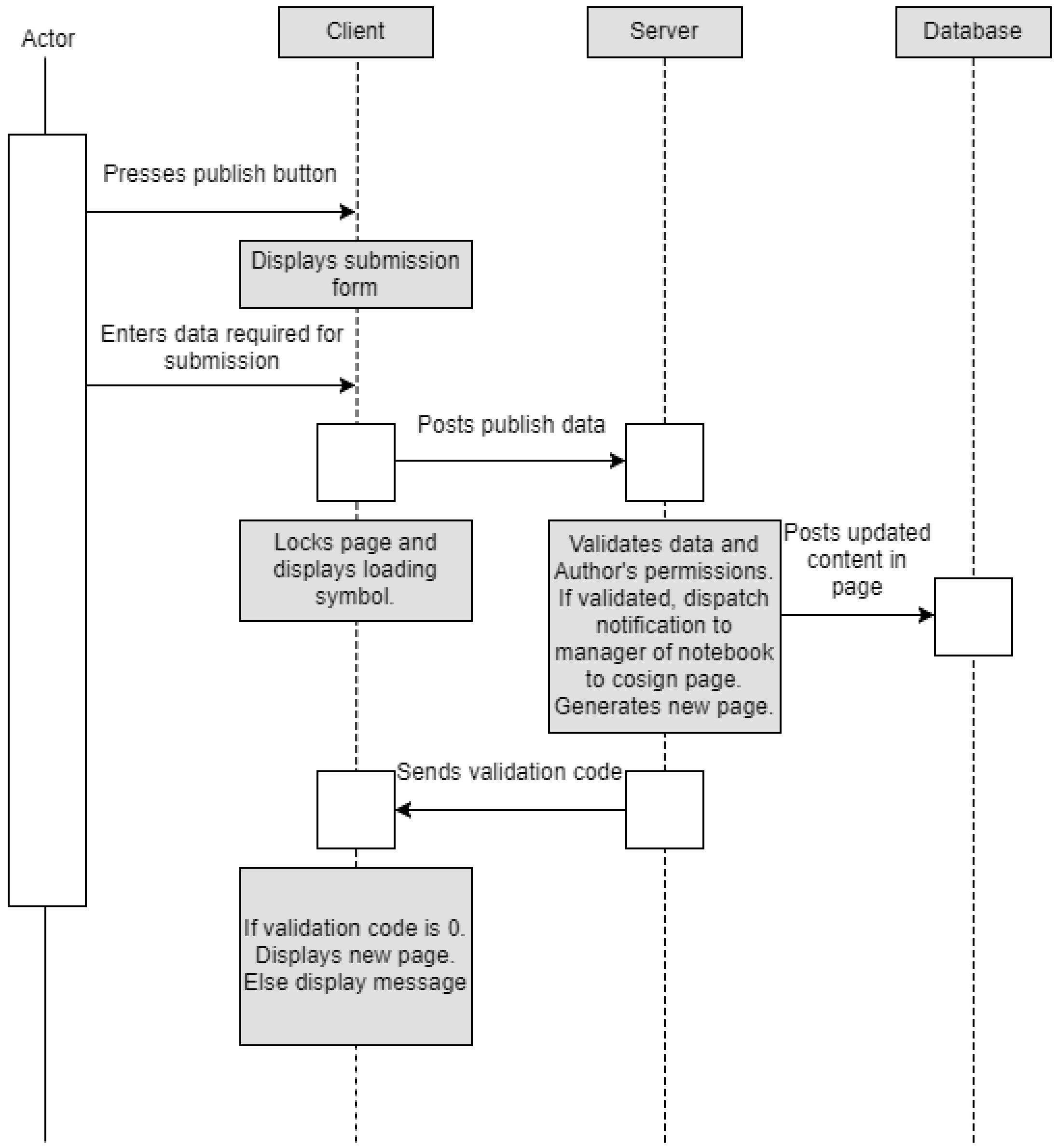
Start or continue work on a page


Actor
Client
Server
Database
Picks notebook
Request Active Page
Requests for Active Page for Author
Current Active Page for Author
Uses caching to determine if page is likely to be new or in progress. Displays blurred content if in progress.
Determines if Author is presented new page or in progress one. Goes to publish mode if page is day old.
Proper state / content to display
Renders submission form, new page, or old page with content.

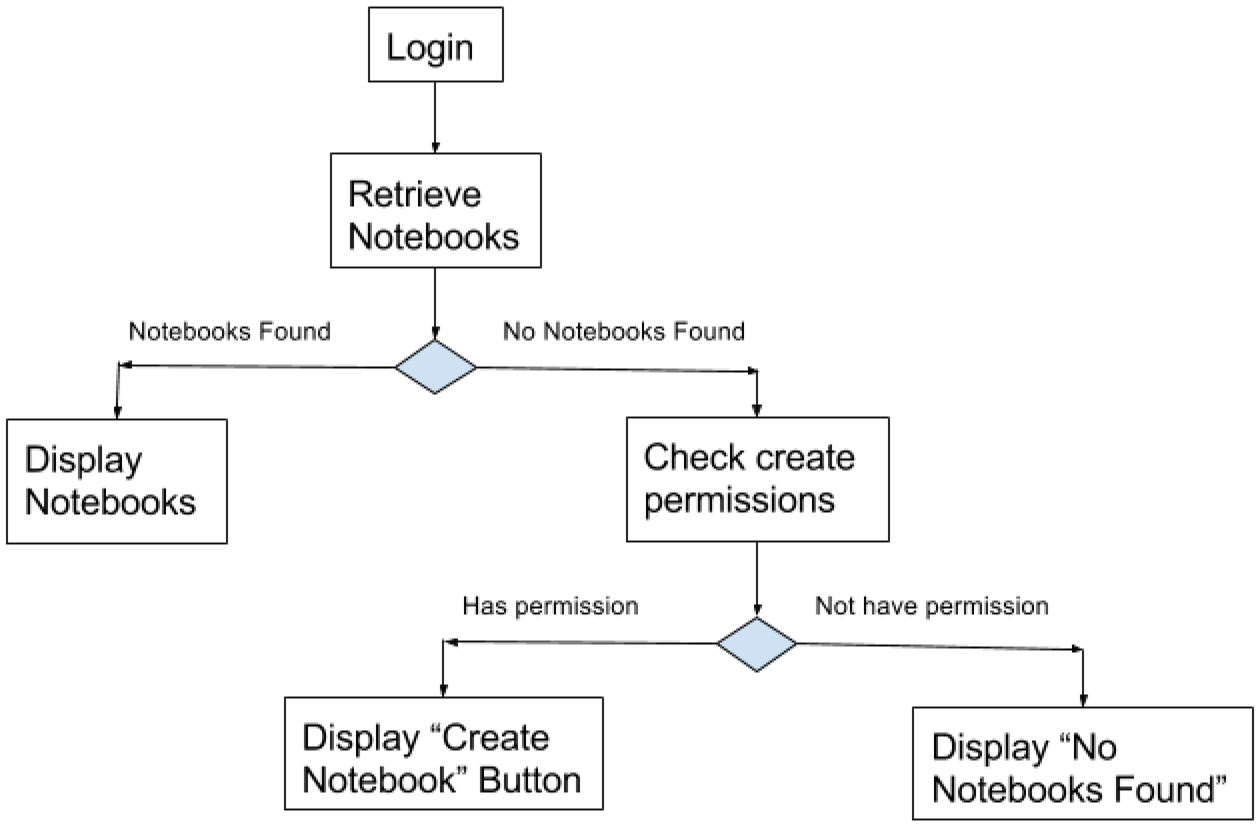
Update page


Actor
Client
Server
Database
Enters data into form and clicks submit
Posts data submission
Posts updated content in page
Displays updated content into the page.
Validates data and Author's permissions.
Sends validation code
Removes and displays message if validation code is not 0.

Publish page to notebook


Actor
Client
Server
Database
Presses publish button
Displays submission form
Enters data required for submission
Posts publish data
Locks page and displays loading symbol.
Validates data and Author's permissions. If validated, dispatch notification to manager of notebook to cosign page. Generates new page.
Posts updated content in page
Sends validation code
If validation code is 0. Displays new page. Else display message

Activity diagrams


Login
Retrieve Notebooks
Notebooks Found
No Notebooks Found
Display Notebooks
Check create permissions
Has permission
Not have permission
Display “Create Notebook” Button
Display “No Notebooks Found”