8.2 Code Repository Organization .................................................. 24
8.3 API Description ........................................................................ 25
  8.3.1 Profile API ......................................................................... 26
  8.3.2 Search API ......................................................................... 27
  8.3.3 File API .............................................................................. 29

9 Future work ................................................................................. 30

10 Lessons Learned ....................................................................... 31

11 Acknowledgements .................................................................... 32

12 References .................................................................................. 33
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VTDLP Home page</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>Current Home page</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>Search page</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>Current search page</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>Current user page</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>Edit profile page</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>Edit profile page</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>Application Layer Architecture</td>
<td>16</td>
</tr>
<tr>
<td>9</td>
<td>Msal-react component tags</td>
<td>17</td>
</tr>
<tr>
<td>10</td>
<td>Input CSV file for Python script</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>Output CSV file from the Python script</td>
<td>19</td>
</tr>
<tr>
<td>12</td>
<td>Git Branching Pattern</td>
<td>22</td>
</tr>
<tr>
<td>13</td>
<td>Git Repository File Structure</td>
<td>25</td>
</tr>
</tbody>
</table>
List of Tables

1  Files to be manually added to project .................................................. 23
### List of Listings

<table>
<thead>
<tr>
<th></th>
<th>Listing</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MongoDB Schema</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>Bash npm install</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>Auth config snippet</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>Frontend configuration code snippet</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>Frontend configuration code snippet</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>Backend configuration code snippet</td>
<td>24</td>
</tr>
</tbody>
</table>
1 Abstract

The project is part of the CS 4624 Multimedia, Hypertext, and Information Access capstone. The purpose of this project is to create an online repository for the CS 3604 course at Virginia Tech. This course, Professionalism in Computing, has students complete case studies on various ethical issues. The issues range from historical supreme court cases to ongoing struggles. Each year nearly 300 such studies are conducted. There should be a mechanism in place to store these studies so the repository is easy to navigate and search.

Previous work attempted to use a preexisting digital library tool hosted on AWS to implement this repository. Over time, the CS 3604 copy became out of sync and out of date, leading to a mountain of issues. Initially, this group sought to overcome those issues and stay with the previous approach. After attempting to resolve those issues, the group met with a software engineer from the team supporting the original digital library platform. This resulted in a switch to a custom website, built from scratch, to host the CS 3604 repository.

The new fullstack website used React.js, Express.js, Node.js, and MongoDB to accomplish this goal. Due to the late start, the group created a preliminary website architecture, before breaking into tasks of frontend development, application development, backend work, and authentication.

The new repository offers a user profile to each student in the capstone class that is accessed via a Microsoft login linked to their Virginia Tech account. Each user can upload a title, list of tags, and PDF document showcasing their case study. The rest of the site is publicly accessible and can be searched by title and tags. The searching features are less sophisticated compared to the prior website. However, the new website has the advantages of user login, linking of case studies to users via login, and easier maintainability.
2 Introduction

2.1 Client

Dr. Daniel Dunlap is the client of this project. Each semester, he manages the course CS 3604, Professionalism in Computing. Computer science majors are required to take this 3-credit course within their degree program. In CS 3604, students learn to apply ethical and professional lenses to incidents in the computer science field. These will be connected to various topics such as the internet, intellectual property, privacy, commerce, and artificial intelligence. Each student must choose one issue related to one of these topics. Throughout the course, each student is required to conduct research on their chosen case study and present it to the rest of the class.

2.2 Problem Statement

Each student’s case study is composed of a presentation and a report. After each semester, Dr. Dunlap stores each student’s case study within a digital library created by past student groups in CS 4624. The website was created to allow students to observe past case studies and make their own contributions. Currently, the digital library contains over 900 past deliverables. One glaring issue of the digital library is that there isn’t any functionality for authentication. Students cannot connect to the digital library using their Virginia Tech accounts. Due to this, it is not currently possible to link a student’s account with their case study. There needs to be a mechanism in place for individual students to upload their case study and have it permanently linked to them.

2.3 Motivation

Dr. Dunlap’s end goal is for the digital library to be similar to a social media platform. As students upload their case studies into the library, other students will be able to provide their input and ratings for them. The authenticated presentations and reports can have a “thumbs up” rating system, and comments from other students describing their quality. This would support a better sense of community among the CS 3604 students, helping them understand how to improve their case studies. The large catalog of past case studies will also inspire students to brainstorm their own case study ideas.

2.4 General Approach

To move from a general archive format to a more social format, there must be many different factors incorporated into the platform. A form of authentication must be established to provide a sense of self within the platform. This will allow users to connect their school identity with their account on the platform. Enabling this sense of self doesn’t just make it easier to create a social setting but also allows for safety precautions to be taken in case there is any bullying, harassment, or other forms of inappropriate online activity. After this, social tools can be added to help enhance a user’s online presence. A like button, comment threads, and marking favorites are all functionalities that can be added to the platform to create social interactions within the archive. Moreover, these functionalities can be used to later create algorithms for recommended tabs that will help users find other posts similar to those they’ve liked.

Initially, the group intended to continue using VT DLP (Virginia Tech Digital Library Platform) [1]. The 2022 and 2023 capstone groups [2], [3] used this tool as a basis for their websites. This tool offers a wide array of features:

- Audio, image, video, and PDF support
- Full text and categorical searching
- Configurable homepage and menus
- Full integration with AWS services
With all these features, searching and configuration is in theory very easy. But, there were some issues. First, the Github repository of the case studies website was 3 “forks” off of the original DLP repository. This made it difficult to consistently pull changes from the DLP repository. These changes are critical in bug fixes, feature updates, and handling AWS updates.

Even if the forking was done correctly, the DLP repository recently went private to handle an issue with syncing their code to AWS Amplify. Thus with the free to use tier of Github, updating the case studies repository is no longer possible. This synchronization factor, combined with the difficulty in updating the configuration within AWS, led the team to consider building a site from scratch.

2.4.1 Transition from Old Approach

In early April, the team met with Lee Hunter, a Virginia Tech software engineer working on the VT DLP. This was the first in-depth technical interaction with a member of the DLP team. Mr. Hunter walked the current capstone team through some of the features of the site, in addition to describing the setup of the configurations in AWS. The capstone team examined three options to move forward.

1. Continue with the DLP approach, now with the guidance of Mr. Hunter
2. Create an instance of DLP through the DLP team to host future case studies and translate all data to there
3. Create a custom hosting site from scratch

With option 1, we would be able to catch up on nearly 1.5 years of updates and start implementing newer features that closely fit the case studies use case. However, this would be a stopgap measure. AWS makes frequent updates, which is great for improving user experience, speed, cost, and the hosted product. A large downside is that these updates make existing code and configurations incorrect, leading to build errors.

Option 2 would be the easiest to implement. However, it leaves a significant portion of the feature control outside the hands of Dr. Dunlap. The VT DLP team manages several existing collections and archives, so huge changes are risky. In addition, uploading case studies would be difficult without user authentication. Either an admin account would need to be created and shared with all the students, or a staff member of CS 3604 would need to upload the files. Thus, the team’s development work would focus on scripting and automating this upload process for the CS 3604 staff.

Option 3 would require a fair bit of work. The group didn’t have a depth of knowledge with full stack web development, but the disjointed knowledge of the group merged to provide baseline knowledge. The obvious disadvantage of this is the steep hill of raw code that needs to be completed. Luckily, the user requirements were thoroughly understood. This project would have good longevity, with room for improvement.

Thus, the team decided that option 3 would be a good bet to pursue. Option 1 wouldn’t be maintainable for the long term. Option 2 would be the fallback plan, as old case studies could be uploaded quickly to a new site and automation scripts would be quick to make. Option 3 being selected, however, means that the current development is far behind what it would have been, given a proper start.
3 Requirements

Due to the redesign of the website, this requirements section will detail the requirements from the ground up. Requirements that have already been fulfilled on the current site will be described in less detail, with an indication of such.

3.1 Case Study Information

Since the CS 3604 repository revolves around case studies, the information needed to describe each case study is vital. At a minimum, the following requirements were determined.

- Virginia Tech “looking” website using the VT branding guides [4]
- Personal identifier of author based on account
- Student login to modify their case studies
- Ability to add a title and list of tags to each case study
- Ability to add one PDF to each case study
- Basic searching of case studies by title and tag

Considering the late change in project plans, these minimum requirements are bare.

3.2 Authentication

The main requirement for the custom site that was not present on DLP is allowing students to sign in using their Virginia Tech email address. This proved challenging for the 2023 capstone group [3].

Authentication would allow students to have their reports and presentations linked to their Virginia Tech accounts. It could allow students to interact with others’ posts to encourage discussion and interaction. The 2023 capstone group’s implementation allowed students to log in with login credentials given by Dr. Dunlap. Students were then allowed to upload their case studies themselves; this meant that those uploads were all associated with the same account. This poses a severe security risk considering the broad access to all case studies past, present, and future. Outside of security, it meant that there was no identifier linking students to their case studies.

The Spring 2023 group planned to implement authentication through OIDC (OpenID Foundation) [5] and when that didn’t work out, Google authentication [6] was the next goal. This was unfinished when the present team took over the project. Since then, Virginia Tech has moved from primarily Google services to primarily Microsoft services. This caused us to change to Microsoft authentication [7].

When students attempt to log into the application, they will be greeted with a pop-up to log into their Microsoft account via VT SSO. Once logged in, they should be able to upload their case study, edit the content (including metadata), and delete the content if they choose to.

3.3 Student interaction

Dr. Dunlap’s goal for the website is to make it more interactive, e.g., like a social media library, where students can post their case study and other students can interact with it by liking, commenting, etc. There were a lot of ideas in terms of what interactions Dr. Dunlap wanted to see implemented. These interactions were not intended to take place during the current capstone. However, the current capstone saw it necessary to make sure that such features could be added later. This called for authentication and Virginia Tech student accounts.
3.4 Future Work

Due to the late start and ground up nature of the project, there are several requirements that future groups should look to implement.

- More complex and robust queries of titles
- Multiple file uploads per case study with multiple types allowed
- Multiple case studies per user
- Liking and commenting features
- More complex search ranking policies to show “better” case studies or users
- Delete profile feature
- Likes, comments, and share features
- Update CSS after consulting with users
4 Design

4.1 Presentation Layer

For our presentation layer/frontend we wanted to stay consistent with the previous website layout. The previous frontend displays the home page of the CS 3604 Case Study Library as shown in Figure 1. That home page consisted of a search bar, a navigation bar that holds options such as upload, case study library statistics, and two options that allow students to view all studies or view studies by topic.

Figure 1: VTDLP Home page

Due to time constraints we decided to keep our website simple. For the overall layout, it contains a navigation bar, search bar, title, authentication text, and a website description. The authentication text simply lets the user know if they’re signed in or not. The navigation bar is used to hold links to the homepage, search page, an edit page, and the log in/log out button. The edit profile page is only visible when a user is logged in. If the user is not logged in, the edit profile will not be visible, as shown in Figure 2.

Figure 2: Current Home page
Using the search bar sends the user to the search page where they are able to view the case studies associated with the keywords they used as the input. Each result will have a preview of the case study, the title, and the tags associated with that case study, as shown in Figure 3.

Figure 3: Search page
The search bar remains at the top of the page. It also displays the amount of results found. Clicking on the case study displays the title of the case at the top, the author username, tags, and a PDF at the bottom. The tags are not currently hyperlinked to anything on the search results. The current search page is shown in Figure 4.

![Figure 4: Current search page](image)

Clicking on a case study should redirect to a user page. The user page will include the title of the case study, the author’s username, the tags associated with the author, and the PDF containing their case study as shown in Figure 5.

![Figure 5: Current user page](image)
When a user wants to edit their profile it will take them to a page that consists of their upload information. The edit profile page is dedicated to editing metadata and uploading a case study. Students will be able to enter all of the important information regarding their case study such as the title, file, and tags. We planned for the edit profile page to resemble Figure 6.

![Figure 6: Edit profile page](image)

Our current edit profile page is shown in Figure 7. It displays the title of the case study, a text box for it where the user can edit the title.

![Figure 7: Edit profile page](image)
4.2 Application Layer

The application layer is composed of Express.js and Node.js [8, 9]. Express is a simple API handling tool that enables users to create their own APIs to access databases and files. Figure 8 shows the overall structuring of these APIs.

![Figure 8: Application Layer Architecture](image)

The profile API is used to get, update, and create case study profiles. The search API is used to search by title and by tag. Finally, the file API is used to retrieve a specific file. These come together to form the core functionality of our application, linking the presentation and database layers. A full breakdown of the APIs can be found in the API Description section.

4.3 Database Layer

In order to store files and data, a database must be used that can be grown horizontally. This means that the information held by an account will usually be around the same amount but accounts will be constantly added. MongoDB is a great tool for horizontally growing databases and therefore was the choice of database for our tech stack [10].

MongoDB is a NoSQL database that stores data in flexible, JSON-like documents. This allows for easy retrieval and manipulation of data. Moreover, the use of tags and metadata that is allowed by Mongo will create subtopics and ways to be able to search for specific cases. Moreover, Mongo allows for intricate access control so that rights to change/replace documents are only given to admins and the person who posted the case. This allows for well-defined roles within the community (even allowing for moderator-type roles in the future).

Files will be stored on disk through the host file system. This is simpler than a MongoDB or other file database. This approach was chosen for its simplicity, along with the fact in enables rapid testing as developers can readily modify files in a testing environment.

4.4 Authentication

Students must be capable of being authenticated while browsing the case study library. Due to Virginia Tech migrating much of their cloud resources from Google to Microsoft, it was beneficial to have students sign into the library using their Microsoft account associated with their Virginia Tech account [11]. To do
this, we created a Microsoft Azure account to have access to a default directory. This directory contains a user pool to which Microsoft accounts could be added.

To enhance the website’s security, the only Microsoft Azure accounts that can be authenticated should be those with the email domain name associated with Virginia Tech, “@vt.edu”. It will prevent foreign accounts from hurting the website’s integrity by adding inappropriate files. This requirement is supported intrinsically by how we add students to the directory’s user pool. Each student’s Microsoft account in the CS 3604 courses has to be manually invited to the directory’s user pool as an external user.

Inviting each student can take an immense amount of time, so a way was devised to optimize the process. A Python program was developed to assist CS 3604 instructors in inviting their students. The program will take a CSV file, as input, listing the students’ PIDs. Those PIDs will have their email domain names appended to them, inserted into a newly created CSV file, and formatted for Microsoft Azure’s needs. Finally, the professor can import the CSV file into Microsoft Azure as a bulk invitations.

The case study library should limit what the user sees based on their level of authentication. A JavaScript library useful for this task is “msal-react” [12]. A simple example is shown in Figure 9. It has the <AuthenticatedTemplate> tag which only renders tags within it when the user is signed in. Otherwise, the <UnauthenticatedTemplate> tag will only render tags within it while the user is signed out.

![Figure 9: Msal-react component tags](image)

### 5 Implementation

#### 5.1 Frontend

The frontend was built using React, and styled using CSS [13, 14]. React allows us to have reusable components, making it easier to have a consistent layout across the website. It utilizes many libraries for client-side functionality such as routing. There are three main folders that contains the bulk of our frontend, which are components, pages, and styles. As of now, there are components for the navigation bar, the search bar, and the search result. The navigation component will be inserted before every page in the App.jsx file. The App.jsx file is the main entry point for our website and allows all of the pages to display.

The pages folder consists of folders for each page that is being implemented. Those folders also includes our the CSS files associated with that page. Currently, there is a page for editing the profile/uploading a case study, the home page, the search page, and the user page.
5.2 API and Backend

The key implementation for this section was a MongoDB schema. This is used to structure MongoDB entries. Here, we are requiring each user/case study to have a title, username (VT email), documentID (automatically generated), and a list of tags. The full schema can be shown in Listing 1

```javascript
const profileSchema = new Schema({
  title: {
    type: String,
    required: true
  },
  username: {
    type: String,
    required: true
  },
  documentId: {
    type: String,
    required: true
  },
  tags: {
    type: [String],
    required: false
  }
}, {timestamps: true})
```

Listing 1: MongoDB Schema
6 User Manual

6.1 Microsoft Azure

For CS 3604 students to be authenticated on the digital library, they must be invited to Microsoft Azure’s user pool as external users.

6.1.1 Batch Invitation

The method of invitation that will be using is to import a CSV file into Microsoft Azure. This CSV file contains rows of the students’ email addresses, the main redirection link, and invitation messages being sent alongside the emails. As the professor of CS 3604, our client, Dr. Dunlap will create a CSV file listing his students’ PIDs. This input file should be formatted in Microsoft Excel as depicted in Figure 10 below.

Using the Python script, make_BatchCSV.py, the initial file is used to create a new CSV file that will meet the formatting requirements to be imported. The initial file must be in the same folder as the Python script for the program to work properly. The desired file being inputted to the program can be configured by setting the variable input file to the file’s name in quotation marks. Furthermore, the name of the created CSV file can be configured in the program by setting the variable output file to the desired name in quotation marks. The created CSV file will look formatted in Microsoft Excel as depicted in Figure 11 below.

Once the formatted CSV file is created, it can be immediately imported into Microsoft Azure. The client will go to their Azure portal, navigate to Microsoft Entra ID, and then to the user section. Clicking on the “Bulk operations” button will display the option to “Bulk invite”. Picking this option will display a menu...
to the left where the CSV file can be imported. Once the file is submitted, the bulk invite operation will begin automatically sending invitations to the student emails listed.

6.1.2 Accepting Invite

Once the above process is complete, students will receive an email message asking them to accept the invitation. Upon clicking the “Accept Invitation” button, students will be requested to sign into their VT Microsoft accounts and accept the requested permissions. Next, Microsoft Azure will ask students to download the Microsoft Authenticator app to keep their accounts secure. Instead, students should use Duo Mobile. They should click “I want to use a different authenticator app”. In Duo Mobile, click the “Add” button and use the QR code given by Microsoft Azure to create the account. Once the account is set up, enter the code Microsoft Azure requests, which is provided by Duo Mobile. At this point, the student should be a part of Microsoft Azure’s user pool as an external user.

7 Developer Manual - Old Version

Much of the old important developer information is related to the hosting of this project with AWS. Though work with AWS has been abandoned, since such work proceeded through March 2024, it is documented in the rest of this section. For information about the final version of the project deliverables, see the next section.

For any issues that arise, AWS offers thorough documentation on most of its services. In addition, the application itself is React.js based, which is equally well documented online.

7.1 Prerequisites

Before getting started, please make sure you have access to the appropriate AWS account and are added to the associated Github organization. Please contact the client to arrange this.


7.2 AWS Services Guide

As described in Table 1, the Library makes use of AWS-managed solutions.

7.2.1 Cognito

Cognito provides managed user authentication for AWS applications [15]. The Case Study Library currently uses Cognito to manage access for case study upload and site administrative pages. There is a single admin account with a password that is used for all uploads. Cognito also offers authentication specifically through Microsoft Azure [16].

7.2.2 DynamoDB

DynamoDB is the primary database for the Library [17]. We have tables for the collections, archives, and site configuration data. This also includes paths on S3 buckets where the uploaded case studies (and their thumbnails) are stored. S3 is Amazon’s file object store, i.e., you can store arbitrary files in this durable and performing file store. Each S3 bucket can contain many objects which are organized using keys. Case study files are stored under public/casestudies/ and thumbnail images and other graphic content are stored under public/sitecontent/image/default/.
7.2.3 AppSync

AppSync provides a managed GraphQL API which the app contacts in order to perform backend operations [19].

7.2.4 Lambda

When a PDF or MP4 file is uploaded, the Library has a Lambda function that creates and uploads a thumbnail of the file [20].

7.2.5 Amplify Deployment

Note: Please ignore this section if the Amplify app is already set up [21]. These instructions are courtesy of the prior iteration of this capstone [3]. To resolve potential issues, please follow these steps:

1. Go to amplify/backend/api/¡api-name¿/parameters.json and add the key value pairs: “CreateAPIKey”: 0 and “APIKeyExpirationEpoch”: -1
2. Execute amplify push. This will take some time and result in a significant amount of output. The command should complete successfully. If not, you have to debug.
3. In the file, set: “CreateAPIKey”: 1
4. execute amplify push
5. In the file, remove “CreateAPIKey”: 0 and “APIKeyExpirationEpoch”: -1 from parameters.json
6. Execute amplify api update
7. Execute amplify push

The following environment variables must be set for the app:

1. `REACT_APP_REP_TYPE`: `Default`. This configures the collection type. The D must be capitalized.
2. `USER_DISABLE_TESTS`: `true`. This disables the test phase of the deploy process, saving time.

The following redirect must exist:

1. Source:
   ```
   <\*/[\.-]+\+$ | \.\.?!(css | gif | ico | jpg | js | png | txt | svg | woff | ttf)$(/[\.-]+\+$)/>
   ```
2. Target address: `/index.html`
3. Redirect type: 200( Rewrite )

7.3 Version Control Management

GitHub is used as the version control platform for this project. Specifically, a GitHub organization with the name vtcs3604-casestudy is used to contain the CS3604 repository. It is recommended that developers are added to the organization as a whole, not just the repository.
7.4 Git Branching Strategy

This section is mostly for semantics, but it shows this team’s approach to organizing code and progress. One branch was created for each active developer to work on their features. Once these features have been created, they can be merged with the staging branch. Then, once about every 1-2 weeks, changes in the staging branch are reviewed for deployment to the production branch (confusingly called dev). This flowchart is shown in Figure 12.

Amplify fits this strategy very nicely. Within an Amplify application, there can be several hosting environments, each synchronized to a git branch. It is recommended that each branch discussed above has its own Amplify hosting environment as it makes testing straightforward. It is also being examined how to test such changes truly locally without pushing to AWS.

To link a git branch with an Amplify production environment, please follow these steps:

1. Login to AWS and navigate to Amplify
2. Select the “casestudies” app
3. Under “Hosting environments” select “Connect branch”
4. Complete the GitHub authorization
5. Select the branch to mirror and use the existing environment before selecting “Next”
6. Once ready, select “Save and deploy”
8  Developer Manual - New Version

As this document becomes outdated with future changes, more information for developers can be found on the GitHub https://github.com/vtcs3604-casestudy/custom-casestudy-site/wiki for this project [23]. Upon departure of the current development team, it is expected that this wiki will continue to be updated.

8.1  Setting up the Application

Before cloning the GitHub repository to your machine, Node.js (with node package manager (NPM)) must be installed on your machine [9]. Please ensure that Node.js is version v20.11.1 and NPM is version 10.4.0. These tools can be more recent versions too if easier.

Once the repository is cloned, node modules must be installed locally. These modules are ignored by git by default to prevent the repository from being overloaded. The node modules must be installed in both the frontend and server folders. Please run `npm install` in both directories or the command in Listing 2 from the repository root directory.

```
cd frontend && npm install
cd ../server && npm install
```

Listing 2: Bash npm install

Once these are installed, there are several files that need to be created as shown in Table 1. These will be expanded upon later. For those working with Dr. Dunlap on the CS 3604 instance of the site, he will have the files with correct credentials. For security reasons, these will not be presented here in full.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authConfig.js</td>
<td>/frontend/src</td>
<td>configures authentication information</td>
</tr>
<tr>
<td>config.js</td>
<td>/frontend/src</td>
<td>configures API hostname to call</td>
</tr>
<tr>
<td>.env</td>
<td>/frontend</td>
<td>configures port to run service off of</td>
</tr>
<tr>
<td>.env</td>
<td>/server</td>
<td>configures port to run API and MongoDB URI</td>
</tr>
</tbody>
</table>

8.1.1  authConfig.js

The authConfig.js file is used as part of authentication through msal-react [12]. This is used to link your application to a Microsoft active directory. Future groups working with Dr. Dunlap should ask him about these details. Otherwise a template can be used from msal [12]. The critical part is to replace the clientId and authority as shown below.

```javascript
clientId: 'XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXX'
authority: 'https://login.microsoftonline.com/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXX'
```

Listing 3: Auth config snippet

To set up an active directory, login to Microsoft Azure and navigate to Entra ID. Navigate to App registrations. The clientId is the same as Application ID or AppId in the active directory. This is found on the App registrations page. Create a new registration if you haven’t already.

For the authority, the portion after microsoftonline.com/ is the same as the Tenant ID on the Entra ID landing page.
8.1.2 config.js

The `config.js` file simply assigns the variable `API_HOSTNAME` to the host of the API. In testing, this will probably be “http://localhost:4000”. In production, this will change, but will likely be https://caseslibrary.cs.vt.edu:4000.

Listing 4 is an example.

```javascript
const API_HOSTNAME = 'http://localhost:4000';
export default API_HOSTNAME;
```

Listing 4: Frontend configuration code snippet

8.1.3 .env (/frontend)

The `.env` file in the `frontend` folder defines the port from which the website will be accessible. This should be 80 in production and 3000 in development. The development version is in Listing 5

```
PORT=3000
```

Listing 5: Frontend configuration code snippet

8.1.4 .env (/server)

The `.env` file in the `server` folder defines the port of the API and the MongoDB to use. If MongoDB is running locally, as it should be in a production environment, the URI will be “http://localhost:27017”. Otherwise, it will be an online MongoDB URI. An example is in Listing 6

```
PORT=4000
MONGO_URI=http://localhost:27017
```

Listing 6: Backend configuration code snippet

8.2 Code Repository Organization

The GitHub repository has three main sections of code: frontend, server, and Azure invite. The Azure invite just contains some Python scripts to import users into the Entra ID directory. The global view can be seen in Figure 13 with annotations of purpose. The structure is based on Net Ninja’s MERN stack tutorial [22].
8.3 API Description

All current API details are shown below. The general format for describing them is:

- **Description**: Brief description about what the API does
- **Visibility**: Describes who can access and run this API (none means public); this is theoretical visibility
- **Route**: Relative route of the API; variable parts should be surrounded in `[]`
- **Request Type**: HTTPS request type (GET, POST, etc)
- **Input Data**: (If applicable) parameters, headers, body, etc. needed in HTTP request as a bulleted list (exclude if not needed)
- **Result (expected)**: Expected result and description, should likely include a JSON
- **Result (unexpected)**: Result from unexpected result (i.e., no such values found), should also include JSON

These will be broken up into sections similar to the ones seen in the repository organization, with separation at the request type level.
8.3.1 Profile API

Interacts with retrieving data regarding the user’s profile and thus their associated case study.

Create Profile

Description: Creates a new profile
Visibility: Private, no one can access it outside the backend; will probably be moved to “hidden” folder
Route: /api/profile/¡username¿
Request Type: POST
Input Data: Body should include JSON of the user with the following information

```
{
  "title": "Test Case Study",
  "username": "Test Username",
  "documentId": "Test Document ID",
  "tags": ["Tag1", "Tag2", "Tag3"]
}
```

Result (expected): JSON of the user created

```
{
  "_id": "661ed43fdcdca79b4f80c56",
  "title": "Test Tile",
  "username": "Test Username",
  "documentId": "Test Document ID",
  "tags": ["Tag1", "Tag2", "Tag3"],
  "createdAt": "2024-04-16T19:40:47.491Z",
  "updatedAt": "2024-04-16T20:13:46.775Z",
  "__v": 0
}
```

Result (unexpected): Will return some sort of error message (generated by MongoDB)

Get Profile

Description: Retrieves information about the profile
Route: /api/profile/¡username¿
Request Type: GET
Result (expected): JSON of user in following format

```
{
  "_id": "661ed43fdcdca79b4f80c56",
  "title": "Test Tile",
  "username": "Test Username",
  "documentId": "Test Document ID",
  "tags": ["Tag1", "Tag2", "Tag3"],
  "createdAt": "2024-04-16T19:40:47.491Z",
  "updatedAt": "2024-04-16T20:13:46.775Z",
  "__v": 0
}
```
Update Profile

Description: Updates the specified profile with some or all fields
Visibility: Only a given user can update their own fields
Route: /api/profile/{username}
Request Type: PATCH
Input Data: JSON of applicable parameters to change, for example a title change would look like this.

```
{
  "title": "New Title"
}
```

Result (expected): The OLD user data in JSON format

```
{
  "_id": "661ed43fdccfa79b4f80c56",
  "title": "Test Title",
  "username": "Test Username",
  "documentId": "Test Document ID",
  "tags": ["Tag1", "Tag2", "Tag3"],
  "createdAt": "2024-04-16T19:40:47.491Z",
  "updatedAt": "2024-04-16T20:13:46.775Z",
  "__v": 0
}
```

Result (unexpected): JSON with error tag

```
{
  "error": "No such profile"
}
```

8.3.2 Search API

Retrieves information relevant to searching, namely IDs and tags

Search by Title
Description: Searches case studies by a title via substring matching
“after” would match “Computer science after AI”
Route: /api/search/title/¡title¿
Request Type: GET
Result (expected): Array of JSON, with only an “id” tag
[
  {
    "id": "661ed95ae43003d785894774"
  }
]
Result (unexpected): Empty array if no results are found
[]

Search by Tag
Description: Search by a provided tag
Route: /api/search/tag/
Request Type: GET
Result (expected): Returns array of dicts with “id” tags
[
  {
    "id": "661ed95ae43003d785894774"
  }
]
Result (unexpected): Empty array if no results are found
[]

Get List of Tags
Description: Get list of all tags across all users
Route: /api/search/tags
Request Type: GET
Result (expected): Array of tags
[
  "Tag1",
  "Tag2",
  "Tag3"
]
Result (unexpected): Should only happen if Mongo server is down, unknown error

Get all IDs
Description: Gets list of all user ids
Route: /api/search/all
Request Type: GET
Result (expected): List of user ids
[
  "661ed95ae43003d785894774",
  "661ed95ae43003d785894774"
]
Result (unexpected): JSON or HTML with error
8.3.3 File API

Used for uploading and getting case files

**Retrieving a File**

Description: Gets the file attached to the case profile
Route: /api/file/fileName
Request Type: GET
Result: The file is retrieved

**Downloading a File**

Description: Downloads a file to the case account in the base file system
Route: /api/file/upload
Request Type: POST
Result: The file is stored in the file system (as long as it is .pptx, .pdf, or .docx. Can be changed in the code shown below for allowed types.)

```javascript
const allowedTypes = ['application/pdf',
'application/vnd.openxmlformats-officedocument.presentationml.presentation',
'application/vnd.openxmlformats-officedocument.wordprocessingml.document'];
```
9 Future work

Since the site was custom built from the ground up, there is plenty of room for additional features. Some proposed work for the future includes more fields for each case study, uploading of multiple documents and types, and increasing search functionality.

Any future student interaction features that appears on the frontend should be made as a component. For example, a like button or comment section. Creating a like button component will make it easier to reuse throughout the website. The same goes for the comment section. The styling should follow the VT color, and typography guidelines.

See also the discussion in Sections 3.3 and 3.4.
10 Lessons Learned

A major problem that plagued the group early on and is still ongoing is the AWS Amplify build errors. The last rebuild of the website prior to our group was in May 2023. Progress has been made on correcting this, however there is a persistent problem with the GraphQL Schema. It ultimately caused the late decision to switch to a custom website.

When building the custom site, the team underestimated how tightly linked the various components were. This is especially true for the presentation and application layers. Using React.js to call the Express.js endpoints requires significant knowledge of both.
11 Acknowledgements

First and foremost, we would like to thank the two previous CS 4624 Capstone groups who worked on this. Their work and documentation made our jobs significantly easier. We are indebted to these two groups for their work.

The Spring 2023 group with Theo Ouzhinski, William Denman, Katie Geibel, and Thrilok Shivaraman [3].

The Fall 2022 group with Trevor Bagbey, Matthew Betsill, Omar Elgeoushy, Daniel Setareh [2]

We would also like to thank Dr. Dunlap and Dr. Fox. Dr. Dunlap is the instructor for CS 3604. His optimism and guidance were crucial to the team. Dr. Fox is the instructor for the CS 4624 capstone class. He was instrumental in pointing us in the correct direction.

Finally, we would like to thank the developers of the VT Digital Library Platform, including Dr. Yinlin Chen and Lee Hunter, without whom this project would not have been possible.
12 References

References


34