

Problem Description

As a prospective renter

Finding Short Term Housing as a student is,

- Overwhelming: there are several different housing platforms to consider, and most do not cater to short term housing.
- Time-Consuming: requires a lot of time going through irrelevant listings and manually tracking listings across websites to compare options.

As a prospective sublesser

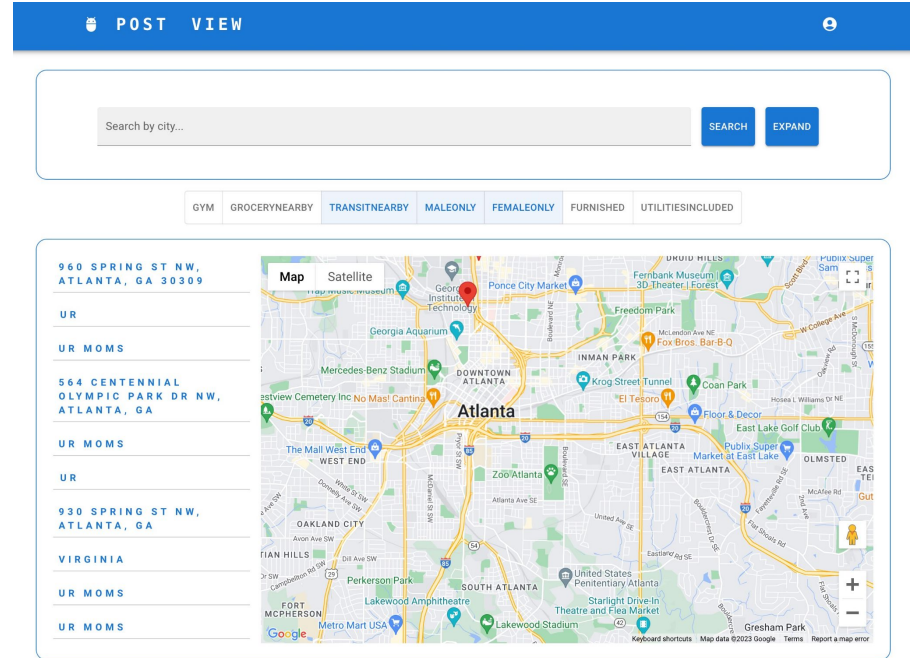
Finding someone to short-term sublease is,

- Unoptimized: the existence of so many general platforms requires duplicating listings across many sites to reach the most prospective short-term renters.
- Untargeted: there is not an existing well-supported platform to directly connect with prospective student short-term renters.

Our Solution

Through user interviews, we found the student sublease market is interested in a web platform that:

- Has extensive filtering capabilities for quickly finding desired listings
- Has a map centric interface to easily relate the location of listing to nearby places of interest
- Is optimized for creating detailed listings with minimal effort to ensure listings are as representative as possible without being burdensome to create.



Our Solution

We provide those moderating the site with a separate webpage for:

- Realtime statistics about user interactions with the site
- Client interface for conducting A/B testing
- Client interface for modifying the layout shown to users

Post Listings Page ^

☒ Tags

☐ Advanced Search

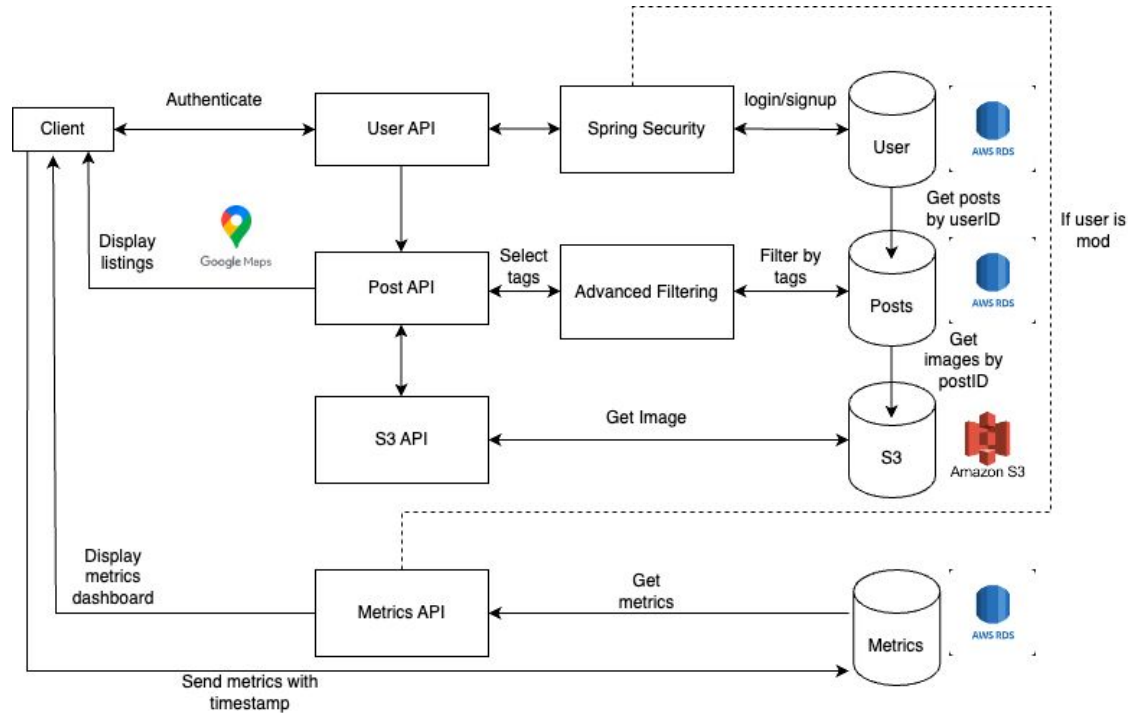
Post Listings Form ^

☒ Tags

SUBMIT NEW LAYOUT



Our Solution: Architecture Walkthrough



Learning Prototype Walkthrough

[wow look at our demo](#)

Questions to Answer in this Sprint

With the addition of our metrics page and feature flag toggling, what is the difference in user interaction for the following layouts,

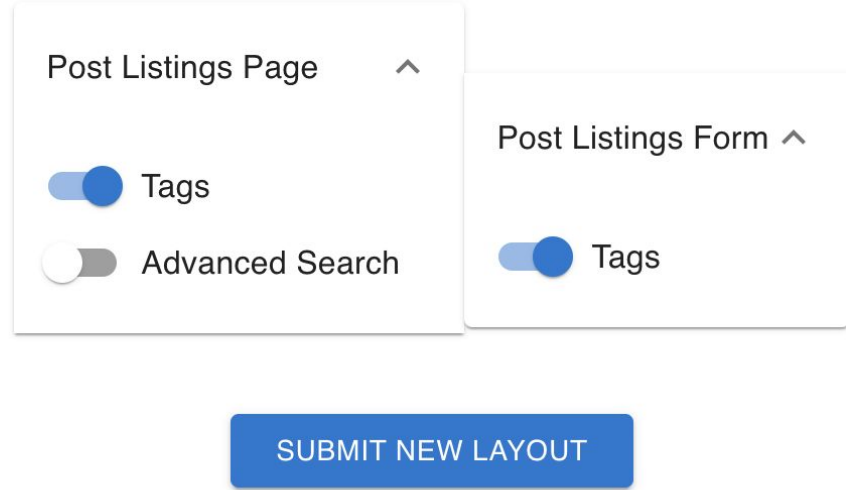
- Make a post page [Customer Segment: students looking to post a sublease
 - Tagging posts enabled (A), tagging posts disabled (B)
- View Listings Page [Customer Segment: students looking for a sublease]
 - Test Group 1
 - Tag set containing apartment external (ex. Grocery Nearby) attributes (A), tag set containing apartment internal (ex. Male Only) attributes (B)
 - Test Group 2
 - Advanced search features enabled (A), advanced search features disabled (B)

User Feedback on Learning Prototype

Gathering the data

We deployed the website, shared the link with several users and asked them to complete the tasks of finding a listing and posting a listing, and we controlled the layout presented to them.

We can then analyze the metrics in the metrics dashboard, all metrics are averages.



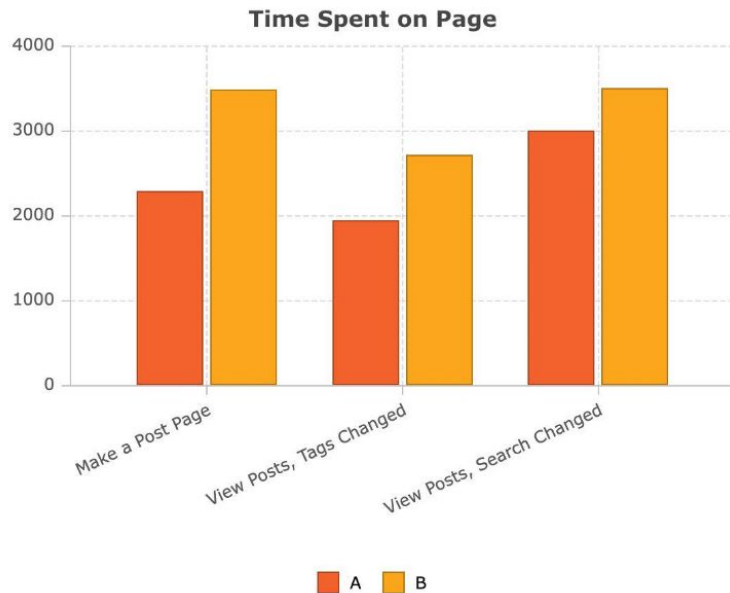
The image shows a screenshot of a web form with two panels. The left panel is titled 'Post Listings Page' and contains two toggle switches: 'Tags' (which is turned on, indicated by a blue circle) and 'Advanced Search' (which is turned off, indicated by a grey circle). The right panel is titled 'Post Listings Form' and contains one toggle switch: 'Tags' (which is turned on, indicated by a blue circle). Below these panels is a blue button with the text 'SUBMIT NEW LAYOUT'.

User Feedback on Learning Prototype and Mockups

Tags (A) vs No Tags (B) when making a post

As shown in the “make a post page” section, with tagging posts enabled, users spent less time on average than users who had the tagging posts feature disabled.

We can interpret this as the tagging posts feature saving users time when creating a post as opposed to when the feature is not available and they have to list these attributes in the description

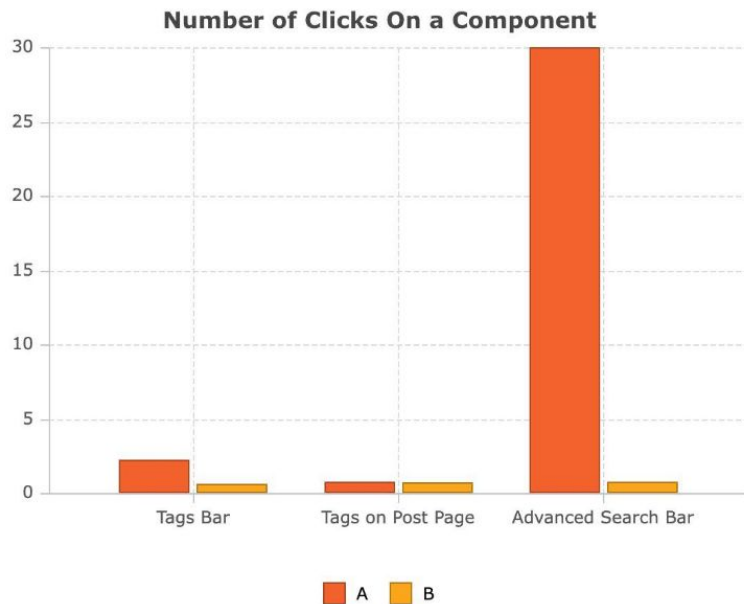


User Feedback on Learning Prototype and Mockups

Filtering with Internal Attribute Tags (A) vs External Attribute Tags (B)

As shown in the “tags on post page” section, the tags with external attributes (grocery nearby, transit nearby, gym, etc) were clicked slightly less on average than the tags with internal attributes (male only, female only, furnished, etc).

This can be interpreted as the internal attributes being more desirable to users for filtering listings than the external attributes, but not significantly more desirable.

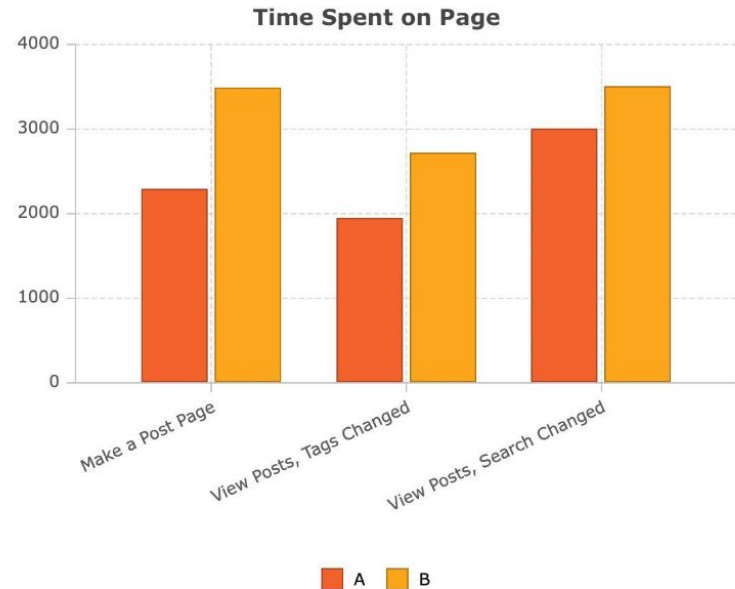


User Feedback on Learning Prototype and Mockups

Advanced Search Enabled (A) vs Disabled (B)

As shown in the “view posts, search changed” section, when the advanced search bar is enabled, users spend less time on average on the view listings page when compared to the advanced search bar being disabled.

This can be interpreted as the advanced search bar saving users time when searching for listings.



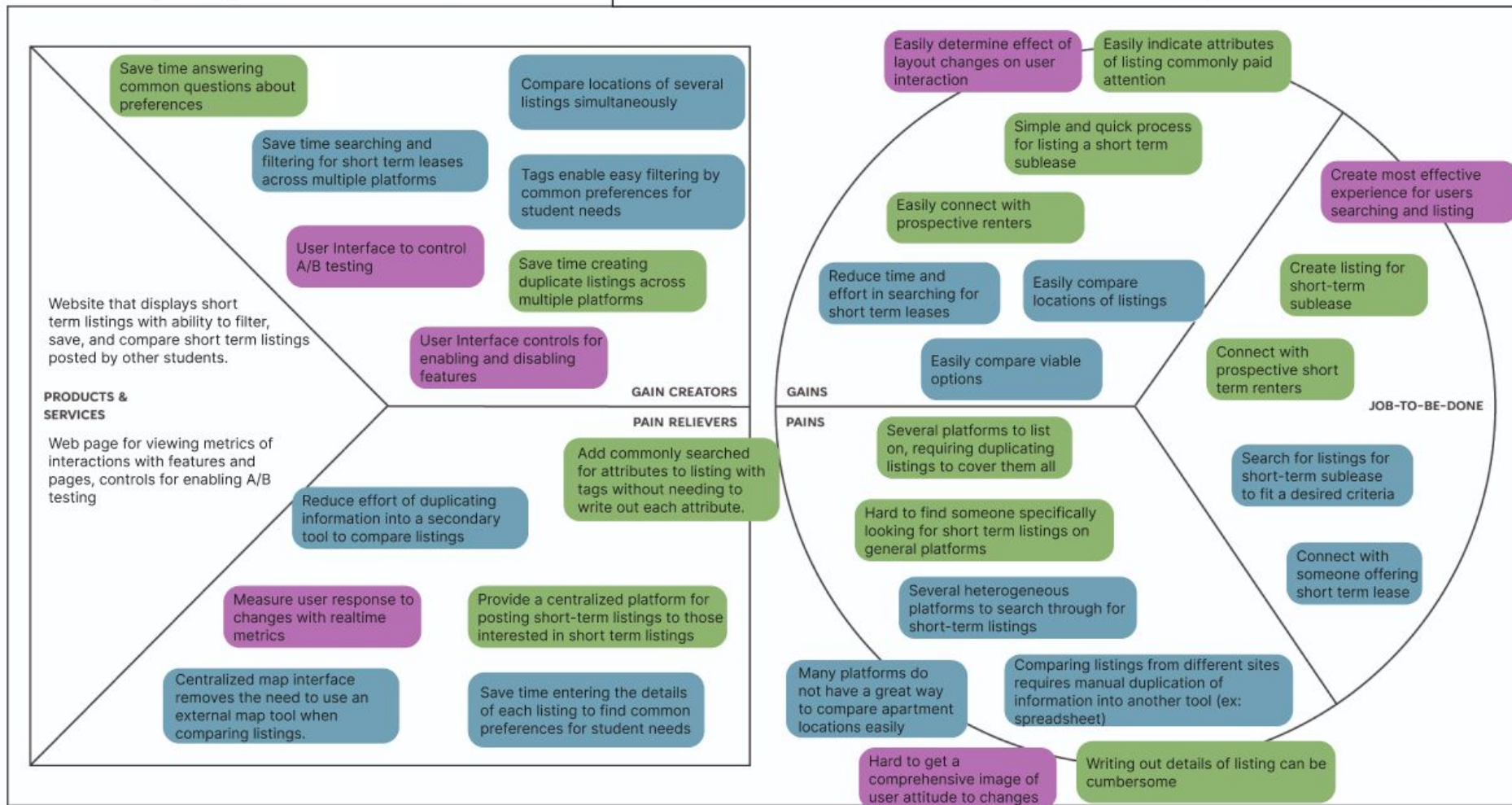
Value proposition canvas.

Key

Students looking **to** sublease

Students looking **for** subleases

Website Moderators / Devs



Business Model Canvas

Key Partners

- university housing groups and organizations
 - these organizations can enable us to gain more adoption of our solution
- university marketing events
 - these provide another avenue for us to connect with potential users
- AWS
 - for enabling components of our backend
- Google Maps
 - for displaying location of listings
- users
 - supply listings
 - supply site traffic
 - report potentially fraudulent or suspicious listings

Key Activities

- user centered design processes
- documentation
- scalable infrastructure choices
- review user feedback on listings and general site
- metrics review/deep dive

Key Resources

- S3 for image and object storage
- reliable cloud database hosting using AWS RDS
- Spring Boot Security for user authentication

Value Propositions

- Gain Creators
 - Save time searching and filtering for short term leases across multiple platforms
 - Save time creating duplicate listings across multiple platforms
 - Provide metric insights for site usage
- Pain Relievers
 - Reduce effort of duplicating information into a secondary tool to compare listings
 - Provide a centralized platform for posting short-term listings to those interested in short term listings

Customer Relationships

- communities
 - connect students looking for subleases with students looking to sublease
- co-creation
 - value of website driven by users creating listings

Channels

- Website that displays short term listings with ability to filter, save, and compare short term listings posted by other students.

Customer Segments

- students looking FOR subleases
 - students who are seeking to sign a short term sublease
- students looking TO sublease
 - students vacating their apartment temporarily hoping to sublease while they are away
- website moderators
 - those in charge of monitoring the website, making design decisions, interpreting site metrics

Cost Structures

- variable costs
 - Google Maps API calls
 - AWS S3 usage
 - Server hosting/usage
 - Interacting with payment processing services for premium offering purchases
- fixed costs
 - salaries

Revenue Streams

- premium offerings
 - listing boost: posters struggling to find a renter can pay a small amount to have their listing promoted and shown to more users
- advertising
 - advertise products relevant to student subleasing such as renter's insurance for those looking to sublease and those looking for subleases

Future Steps

- Improve API Performance
- More Storage
- Customized Ads
- UI/UX Research

Code Review

This example highlights the interaction of the different client-end files required to enable setting and loading the layout.

- In MetricsDashboard.js, a site moderator can update the layout by updating the layout database.
- When a user logs in, their layout is loaded and stored as a session variable in Login.js using functions from Auth.js.
- When they visit the ViewPage, PostingsList.js dynamically sets the tags using the layout loaded from the session variable.

```
MetricsDashboard.js
const submitNewLayout = () => {
  if (toUpdateLayout.advancedSearch && toUpdateLayout.tags) {
    setShowError(true)
  } else {
    setShowError(false)

    // upload valid config
    fetch("/layout/update", {
      method: 'GET',
      headers: {
        'Authorization': `Bearer ${data.accessToken}`,
      },
      body: JSON.stringify(toUpdateLayout),
    }).catch((e) => {
      console.log("failed to upload: {}", e)
    });
  }
}
```

```
Login.js
fetch("/layout/getLayout", {
  method: 'GET',
  headers: {
    'Authorization': `Bearer ${data.accessToken}`,
  },
}).then((response) => {
  return response.json();
}).then((responseData) => {
  -
  setLayout(responseData)
});
```

```
PostingsList.js
let toSet = getLayout().postListingTags ? internalTags : externalTags
```

```
Auth.js
export const setLayout = layout => {
  sessionStorage.setItem("layout", layout)
};
export const getLayout = () => {
  return sessionStorage.getItem("layout");
}
```


Code Review

This portion of code was especially important on our client-side implementation of submitting a post. This code portion:

- uploads the image files associated with a post one at a time
- properly sends the image data and the associated postUUID to the server, so that we can properly keep track of which images in our s3 bucket are associated with which post.

The code itself was not exceptionally challenging to write, but was a huge turning point in our project.

```
.then((responseData) => {  
  // try to upload the images  
  console.log("responseData: " + JSON.stringify(responseData));  
  var postUUID = responseData.postID;  
  console.log("post UUID: " + postUUID);  
  
  uploadedImages.forEach(image => {  
    console.log("image: " + image);  
  
    // Now, try to upload images to S3  
    const imgData = new FormData();  
    imgData.append('file', image);  
    imgData.append('post', postUUID);  
  
    const s3response = fetch('/file/upload', {  
      method: 'POST',  
      headers: {  
        'Content-Type': 'multipart/form-data',  
        'Authorization': 'Bearer ${token}'  
      },  
      body: imgData,  
    }).then((s3response) => {  
  
      if (s3response.ok) {  
        console.log('Post to /s3/posts successful!');  
        alert('Successfully Submitted');  
      } else {  
        console.error('Post to /s3/posts failed');  
        alert('Submission Failed');  
  
        //const errorData = s3response.  
        //console.error('Error Response Data:', errorData); // Print the error details  
      }  
    });  
  });  
});
```

Code Review

This bit of code was the magic we implemented in Sprint 5 to finally be able to send Images from our s3 bucket back to our client. It took a lot of trial and error. This is because Image files cannot be directly sent back to the user in a simple fashion. After multiple different methods, our team landed on sending streams of bytes back to the client so that we could piece the image(s) back together and display them on the viewing pages.

```
± Muchen Zhang +2 *
@GetMapping("/{download/images/{postId}")
public ResponseEntity<byte[]> downloadImageFiles(@PathVariable String postId) {
    List<String> postImages = postService.findAllImagesByPostId(postId);
    System.out.println(postImages);

    if (postImages.isEmpty()) {
        return ResponseEntity.notFound().build();
    }

    try {
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        for (String fileName : postImages) {
            byte[] data = service.downloadFile(fileName);
            outputStream.write(data);
        }

        byte[] combinedData = outputStream.toByteArray();

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.IMAGE_JPEG);
        headers.setContentLength(combinedData.length);

        return new ResponseEntity<>(combinedData, headers, HttpStatus.OK);
    } catch (Exception e) {
        System.out.println("Error combining files to bytes array.");
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }
}
```


Code Review

This bit of code was the magic we implemented in Sprint 5 to finally be able to send Images from our s3 bucket back to our client. It took a lot of trial and error. This is because Image files cannot be directly sent back to the user in a simple fashion. After multiple different methods, our team landed on sending streams of bytes back to the client so that we could piece the image(s) back together and display them on the viewing pages.

```
± Muchen Zhang +2 *
@GetMapping("/{download/images/{postId}")
public ResponseEntity<byte[]> downloadImageFiles(@PathVariable String postId) {
    List<String> postImages = postService.findAllImagesByPostId(postId);
    System.out.println(postImages);

    if (postImages.isEmpty()) {
        return ResponseEntity.notFound().build();
    }

    try {
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        for (String fileName : postImages) {
            byte[] data = service.downloadFile(fileName);
            outputStream.write(data);
        }

        byte[] combinedData = outputStream.toByteArray();

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.IMAGE_JPEG);
        headers.setContentLength(combinedData.length);

        return new ResponseEntity<>(combinedData, headers, HttpStatus.OK);
    } catch (Exception e) {
        System.out.println("Error combining files to bytes array.");
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }
}
```