Book Recommender

Braden Martin, Camille Williford, Braden Butler, Logan Tillman, Grant Anderson

Team Name: Book Recommendations

Team Roles

- Members collaborated, thus worked on multiple attributes of the project
- Main Roles:
  - Camille: AWS, Frontend, Ember set-up and integration
  - Braden M. and Logan: Backend (Google Books API and Lambda function)
  - Braden B. and Grant: Frontend
- All members worked on frontend to some degree

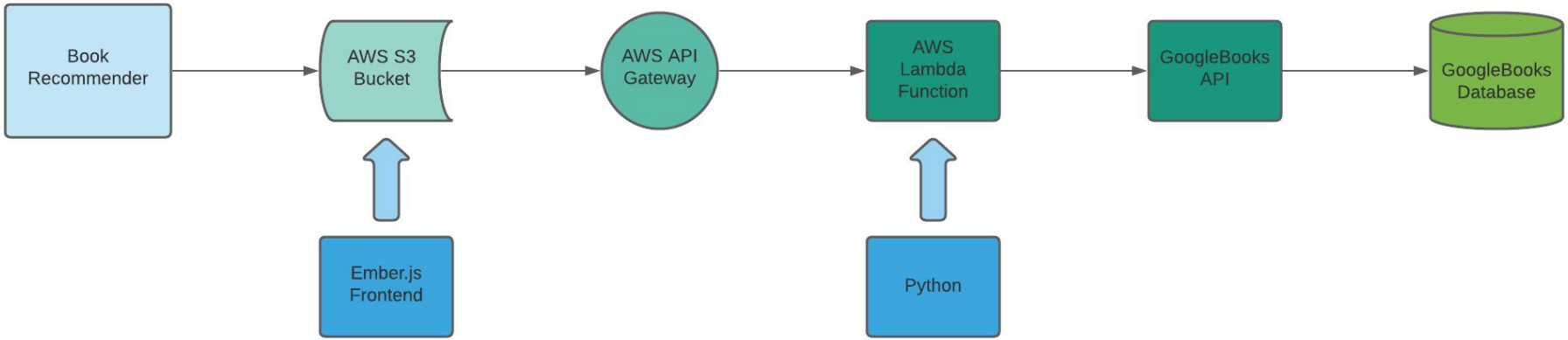
Introduction

- Book Recommender is an online application that provides book recommendations to the user based on search inquiries.
- Users can search by author or genre.
- Our main goal is to create a tool for readers that makes it easy to find their next book without collecting or using their personal information.
- Ember is used for the frontend and Google Books API is used to retrieve book information

Customer Value

- We wanted to target both avid and new readers who are struggling to find new works.
- Our application intends to narrow down a reader’s search by filtering based on genre or author.
- Book Recommender does not collect user data and relies entirely on the user’s search.
- New readers can use our application to find something that interests them while veteran readers can use it to find something exciting and new.

Software Architecture



This is an entirely serverless, cloud application that enables incremental development and flexibility. The Ember project defines a deployment pipeline that packages the frontend code and deploys it to an AWS S3 bucket. The code hosted on the S3 will then utilize the lambda function via API Gateway. The response from the API also had to be serialized into Ember understandable JSON.

Frontend: Ember

- Team members had prior experience
- Mirage, which fakes API data, enables decoupled development
- Fast and simple to spin up a skeleton project

Frontend Development

```
search: async function(){
    console.log(this.userInput);

    if (this.selectedOption == null) {
        this.set('filterError', "Please select a filter type");
        return;
    }

    let books = {};
    if (this.selectedOption == "inauthor") {
        books = await this.store.query('book', {
            inauthor: this.userInput,
        });
    } else {
        books = await this.store.query('book', {
            subject: this.userInput,
        });
    }
      books = books.map((book) => book.toJSON({ includeId: true }));
      this.set('model', books);
}
```

Technology: Google Books API and Lambda

- Google Books API as a repository for book metadata
- Lambda + API Gateway works as a backend for our project.
- Lambda makes Google Books API requests and reduces the response body to a more manageable format.
- Amazon API Gateway hosts our lambda function and allows us to invoke it from our front end

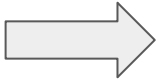
```
#parsing parameters
author = event['queryStringParameters'].get('inauthor', None)
genre = event['queryStringParameters'].get('subject', None)


#building api request string
reqlink='https://www.googleapis.com/books/v1/volumes?q='
if (author):
    reqlink=reqlink+'inauthor:'+ author + '+'
if (genre):
    reqlink=reqlink+'subject:'+ genre + '+'
reqlink=reqlink+'&maxResults=40'
print('reqlink: '+reqlink)

#making request to Books API
req=requests.get(reqlink)
```

Tech: Lambda Data shaving

- Take large response body from Google Books API and reduce it to more workable format
- Removes parsing responsibility from the front end

```
#building our response body with the attributes we care about, looping through all the books and extracting the info we want
for book in reqjson['items']:

    title=book['volumeInfo'].get('title', None)
    author=book['volumeInfo'].get('authors', None)

    if(author!=None):
        authorjoin=", ".join(author)
        author=authorjoin
    genre=book['volumeInfo'].get('categories', None)
    if(genre!=None):
        genrejoin=", ".join(genre)
        genre=genrejoin
    pageCount=book['volumeInfo'].get('pageCount', None)
    id=book.get('id', None)

    thumbnail = None
    if(book['volumeInfo'].get('imageLinks')):
        thumbnail=book['volumeInfo']['imageLinks'].get('thumbnail', None)

    MR=book['volumeInfo'].get('maturityRating', None)
    reqResponse.append({'type': 'book', 'id': id, 'title': title, 'author': author, 'genre': genre, 'pageCount': pageCount, 'thumbnail': thumbnail, 'maturityRating': MR})
```

```
{
  "kind": "books#volumes",
  "totalItems": 360,
  "items": [
    {
      "kind": "books#volume",
      "id": "vD83VP8MZvcC",
      "etag": "mFlOzuTJpiM",
      "selfLink": "https://www.googleapis.com/books/v1/volumes/vD83VP8MZvcC",
      "volumeInfo": {
        "title": "Apollo Beyond the Universe",
        "authors": [
          "Chuck Keyes"
        ],
        "publisher": "Larry Larson",
        "publishedDate": "2011-06-01",
        "industryIdentifiers": [
          {
            "type": "ISBN_13",
            "identifier": "9781452493084"
          },
          {
            "type": "ISBN_10",
            "identifier": "1452493081"
          }
        ],
        "readingModes": {
          "text": true,
          "image": true
        },
        "printType": "BOOK",
        "categories": [
          "Fiction"
        ],
        "averageRating": 5,
        "ratingsCount": 1,
        "maturityRating": "NOT_MATURE",
        "allowAnonLogging": false,
        "contentVersion": "0.0.2.0.preview.3",
        "panelizationSummary": {
          "containsEpubBubbles": false,
```


```
[{
  "type": "book",
  "id": "vD83VP8MZvcC",
  "title": "Apollo Beyond the Universe",
  "author": "Chuck Keyes",
  "genre": "Fiction",
  "pageCount": 312, "
  thumbnail": "http://books.google.com/books/content?
  id=vD83VP8MZvcC&printsec=frontcover&img=1&zoom=1&edge=curl&source=gbs_api",
  "maturityRating": "NOT_MATURE"
}]
```

Technology: Difficulties and What Didn’t Work

- Setting up Ember development environment with Mirage
- Google Books API limitations: attributes aren’t guaranteed to exist
- Serializing API response to Ember recognizable JSON.
- AWS S3 holds onto cached version of Ember app
- Limited reference material for setting up proxy integration in a Lambda function

Project Management

- Difficulty setting up Ember integration with Mirage delayed front end work
- Google Books API null attributes complicated data display on front end
- Major goals were still completed
  - Front end queries Lambda API based on user search parameters
  - Lambda API queries Google Books API and returns reduced data to front end
  - Front end displays query results to user

Reflection

- What went well:
  - Backend functionality is better than we had hoped
    - Still took quite a lot of time to implement it successfully
  - Frontend has the basic functionality we expected
  - Development environment set-up and integration with front end
    - Completed successfully with relative ease
- What went less well:
  - Frontend
    - Troubleshooting dev. environment to work with Mirage was a huge time sink
  - Scheduling meetings
    - Group members' varying schedules made meeting up much more difficult
- Overall, the project was a success: major goals were completed and expected functionality is present