CSCE A401/ CSCE A601
Software Engineering Project

Team Name: The UAA Sea Ferrets
App Name: Flashcard McFlashcardy Face

Pat, Jaaliyah, Eddie, and Boro

System and Architecture Requirements

1. Preface

This document was created to outline the system requirements of our flashcard app (catchy non-sarcastic name forthcoming). This provides the necessary content and context to understand **"why"** we're building this tool, **"what"** the tool is, and at a very high level **"how"** we're planning on getting it done. We'll describe the tooling, capabilities, and possible future use cases and provide links to internal documentation to help the reader further contextualize our product.

This is version 0.5.

2. Introduction

As we are all students in college, we constantly need to memorize new concepts and ideas all the time. To study effectively, most students create physical paper-based flashcards and carry them around all the time. Since carrying flashcards around all the time could be troublesome sometimes, we figured creating an app for e-flashcards could be more convenient for many than the good old-fashioned paper flashcards.

A flashcard is a small note card used for testing and improving memory through practiced information retrieval; it is typically two-sided, with the prompt on one side and the information about the prompt on the other. This may include names, vocabulary, concepts, or procedures. For this project, we will simulate the basic idea of flashcards in the web interface.

Since there's a desperate lack of "good" flashcard applications on the market today where most of the existing ones are old, the user interfaces are sub-optimal, and they're not really platform-independent, we intend to create a user-friendly efficient flashcard app.

We're going to create the best flashcard application you've ever seen. Initially, we're going to build a simple application that allows users to create and store flashcards in the cloud and review them as desired. Users should be able to:

- Create / View Flashcards
- Edit Flashcards
- Delete Flashcards

Further, time permitting, users should be able to:

- Share Flashcards with other users
- Use image recognition to categorize the flashcards from pictures captured on their phones.
- Create training datasets from labeled flashcards.
- Access the flashcards through a mobile device
- Set review schedules
- Export the flashcards to various formats
- View data about their recall rate, progress, etc.

Ultimately, as part of this project we plan to learn the following skills:

- UI/UX Design
- HTML, Javascript, CSS
- Cloud Infrastructure Design
- DevOps and CI/CD implementation
- Python and various front-end tools for the web
- Potentially rudimentary iOS development.

If we can get a working Minimum Viable Product (MVP) implemented there is no reason why we couldn't monetize the application. We don't need to charge users for it - we can monetize the flashcards as "training data" for companies working in the AI/ML space. We could provide a B2B solution where both students and AI can learn about a field.

3. Glossary

API - Application Programming Interface. The "back-end" interface allows various applications to access our data layer independent of the front-end technology supporting them.

AWS - Amazon Web Services. Amazon Web Services are a suite of software services and packages provided by amazon that allow for application hosting on the web.

AWS Lambda Function - Amazon's service for deploying code snippet's on demand. Once triggered it executes its code and potentially returns something useful to the location it was triggered from.

Amazon S3 - Amazon's AWS Simple Storage Service. Amazon S3, or often simply "S3" is a way to store static data in "the cloud" on Amazon's servers.

Back-end - The software tooling that performs data access and editing.

CI/CD - Continuous Integration / Continuous Deployment. The set of tools and practices we'll be using to be able to deploy code rapidly. We should be able to make major non-breaking changes to our code literally dozens of times per hour without causing an

outage - code testing and code deployment should be part of a single automated system.

Cloud - An often misunderstood term meaning “other people’s computers.”

CRUD - Create Read Update Delete.

DevOps - the set of practices combining IT operations and software development. In principle, all of the things that can be automated should be automated as part of this philosophy.

DynamoDB - AWS’s freely provided (up to 25 Gb) cloud storage platform stores objects as key-value pairs or documents.
Front-end - The user-facing portion of the application.

Git - The project will use a version control system to track code progress and manage to version.

Github - A cloud hosting service for Git repositories.

GHA - Github Action. A Github Action is a tool provided free by Github to allow you to spin up computing instances to perform automation on your git repository and more. This project is the primary tool for running automated tests and deploying code to AWS. They can be executed automatically or on command and are fully programmable.

HTML - Hypertext Markup Language, part of the front-end will be static components rendered from HTML.

HTTPS - Hypertext Transfer Protocol - Secure. (or sometimes HTTP over TLS)

JSON - Javascript Object Notation. An extremely common and useful way to store and package objects for distribution over the internet. In the context of this project, it is the primary medium we’ll use to transfer data between the front-end and back-end of our application.

Microservice - A small (and decoupled) service that defines a part of the entire application. The idea behind splitting portions of the product into “microservices” is that they are loosely coupled and independent of the other processes.

MVP - Minimum Viable Product. The smallest product that can be useful to customers.

Python - The programming language used for the back-end of this project.

Vue.js - A progressive Javascript framework for writing user interfaces.

Serverless - A cloud computing methodology where one doesn't have to run the servers and physical infrastructure of an application. That is managed by another provider, and your code is executed on demand.

TLS - Transport Layer Security, the modern protocol for securely delivering content over the web securely.

TOML - Tom's Obvious Minimal Language. A markup language for storing dictionary objects in a convenient and human-readable format.

TypeScript - The programming language (a superset of Javascript) used to control this project's front-end components.

WIP - Work in Progress. The catchall term is used to describe currently ongoing work.

YAML - YAML Ain't a Markup Language. YAML is a markup language for producing human-readable data serialization despite the nomenclature. It's the primary tool for writing Github actions.

4. User Requirements

1. We want to be able to create, retrieve, update, and delete user accounts.
    a. We need to be able to handle simple logins.
        i. We need to check if user logins are unique
        ii. Users must use an adequately secure password
        iii. We must be able to get the user email during initial sign-up to facilitate account password resets.
        iv. Other data may be collected later to facilitate monetization.
    b. Handle some sort of way to store the data
        i. We plan to store the vast majority of data as object files so that the users can retrieve their cards in human-readable formats.
    c. Password resets should be simple and efficient.
        i. Security Questions may be defined during the initial sign up
        ii. The ability to reset via email is industry standard - while we may not do this in the initial versioning, we would like to keep the option open.
    d. Security
        i. All data will be sent over TLS / HTTPS.
        ii. All stored passwords will be encrypted.
2. We want to be able to create, retrieve, update, and delete flashcard objects from online storage.
3. We want to be able to create, retrieve, update, and delete study plan objects from online storage.

5. System Architecture

Our project can be best described as having two architectural components—a website based on the *model-view-controller* (MVC) design pattern, and a backend based on the *client-server* design pattern. Broadly speaking, the back-end will CRUD user and flashcard data using AWS, while the front end will call the backend via its controller, so that users can interact with their flashcards.

The front-end, which GitHub will host, will serve as the entry point to the application. Users will enter their login info, which the website will send to the back-end. The back end will then hook up to the project's AWS storage to authenticate the user and return their stack of flashcards. Once returned, the cards will be rendered and manipulated by the front-end, with changes being periodically relayed back to AWS.

The technical details for the front-end are as follows.

1. Cards and UI are rendered in HTML/Javascript
2. Changes are sent to the server using AJAX requests.
3. Cards can be displayed for review by the front-end

Hypothetically, this would allow us to render the cards on any platform. Provided the platform can make a POST request, we should be able to view the cards. Pat is currently experimenting with Swift and iOS to determine the viability of implementing a second platform but he doesn't expect this to be anything other than a proof of concept by the end of the semester.

The backend will consist of the code that defines how users and their flashcards will be stored, and the code that interacts with AWS to CRUD them. At this time, the backend is simply storing all users and all flashcards in two separate JSON files. Interacting with AWS is hard, so we are keeping it simple for now. These interactions with AWS are accomplished using AWS *lambda functions*.

The overall flow of the the back-end is as follows:

1. A POST request containing JSON gets made to the AWS lambda function
2. The lambda function calls backend code to authenticate the JSON payload
3. The payload is then routed to the appropriate CRUD function
4. The function body runs and returns a JSON response from AWS and/or a status message.

At this stage of development, and with limited time to develop before the semester is complete, we don't plan on implementing offline access, or concurrency. Users should not be able to access other users' flashcards, Therefore, the source of truth for each user can simply be the latest POST request to the server. That is to say, the database will be LIFO. If a user initiates a post request to the server, and authentication is successful, then the response is their source of truth.

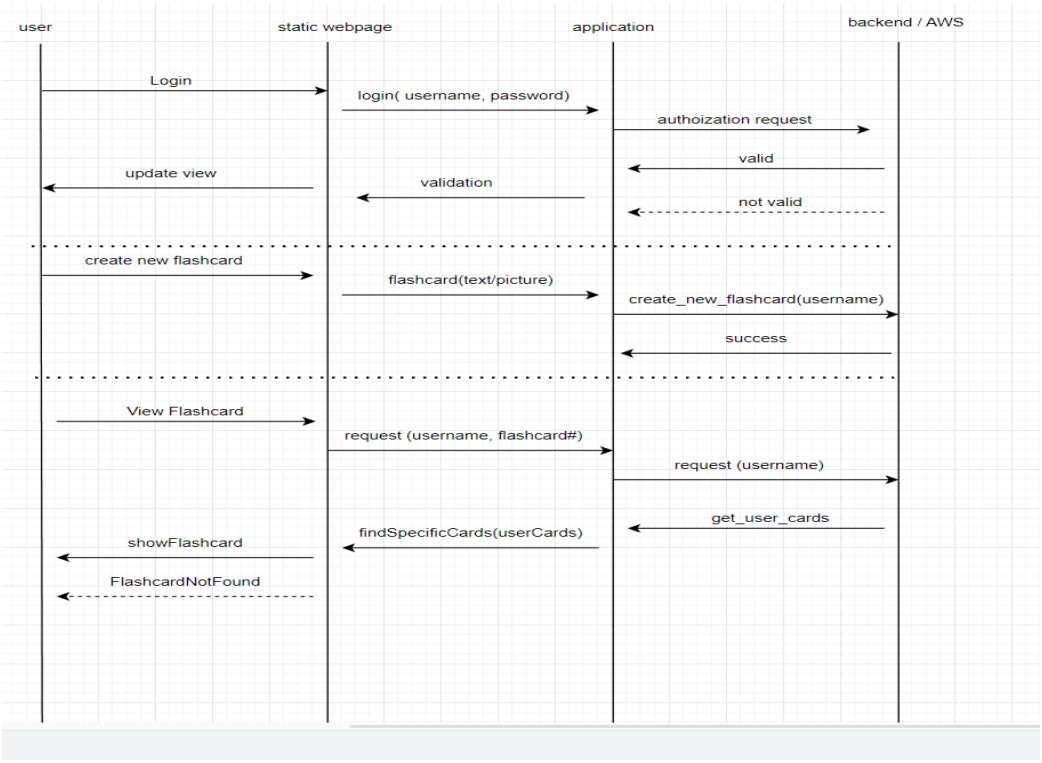
6. System Requirements

1. Front-End
    a. Browser / Vue.js
2. Backend
    a. Python
    b. AWS / Lambda / S3
        i. Microservices for each operational thing that we want to do
        ii. Currently get and post methods are supported (poorly)
        iii. Storage, TOML in S3 or dynamo DB
3. Something to connect them
    a. JSON between FE and BE via post and get requests as appropriate
    b. Serverless AWS lambda functions for all the maintenance of the code
    c. Connected over HTTPs (get/post)

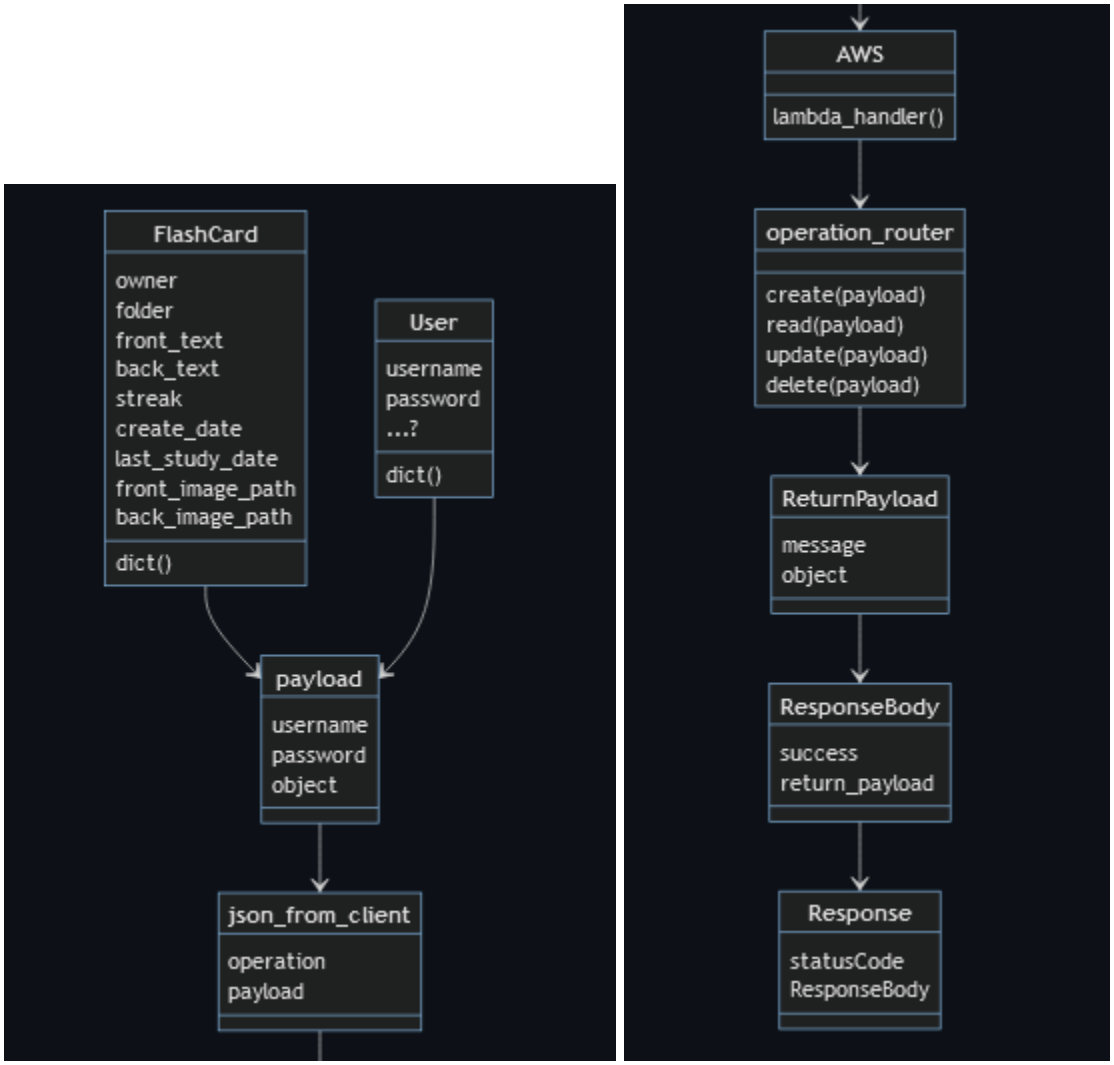
7. System Models

The following graphical system models are used to describe the overall structure, or flow, of our application. We'll begin with a sequence diagram describing the front-end, and then move to two diagrams that depict the operation of the backend.

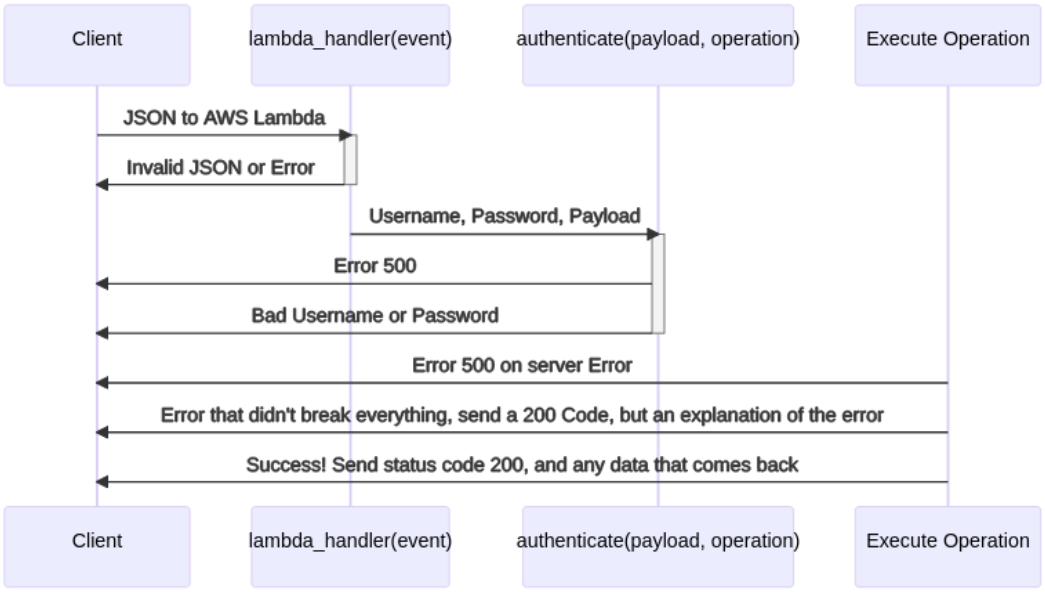
1. Sequence Diagram for Front-End



2. Flow/UML Diagram for Backend (top to bottom, left to right)



3. Sequence Diagram for Backend



To summarize the diagrams for the backend, the backend encodes objects as JSON payloads. These payloads are then made into JSON requests that are passed to the AWS lambda handler function. The lambda handler kicks the input back if it is a malformed request. Otherwise, the username, password, and payload are sent to the authentication function. Then, the operation in the JSON is performed, and the results are sent back from AWS (or an appropriate error message).

In addition to these diagrams, we also have a lovely workflow diagram made by Pat, which is accessible here. Also, these images are evolving and may change again soon. If you want to see the current latest version, follow this link. The document at the link (systems.md) contains the up-to-the-minute version, as well as example payloads and responses.

8. System Evolution

Fundamental assumptions for design, changing user needs and any predictions we have for future evolution. For example, we know that we will need to have two-factor authentication in the future; maybe our authentication solution is such that two-factor can be easily added later on without redoing the whole authentication framework.

1. Design Assumptions
    a. The most important assumption is that we can even do this first. Namely, we will be able to string together several pieces of technology loosely enough without any breaking or risky dependencies.
    b. The need for flashcards won't include a requirement for complex animations or toolings or other non-trivial additions.
    c. The API can be made general enough to be accessed from a wide variety of platforms and devices.
2. Changing User Needs

    Predictions for User Desires

    i. iOS / Android App
        1. It is plausible that a simple web interface will not be satisfactory to the vast majority of potential users. People hardly ever use the browser anymore. In the future, we'd expect to need an Android or iOS application that is able to integrate with the backend seamlessly. As such, our API needs to be designed to be front-end agnostic. Our application should function as long as the appropriate JSON gets sent to the API that follows the appropriate permissions structure.

    ii. Study Optimization
        1. Users may desire the ability to optimize their study sessions. If we could statistically model their behavior, we could programmatically change the rate certain cards are due to provide the best retention rate.

    iii. Classify the cards with Machine Learning
        1. Students may desire to be able to take a picture of a card and have it automatically classified instantaneously saving them time and effort.
        2. Mathematical formula character recognition could be applied to handwritten mathematical formulae to save the user the time and effort of manually inputting them.

    iv. Sharing Cards
        1. Users may desire to be able to share decks with other users or copy and edit those decks.

2. Users may desire to be able to share study schedules and data about their study habits with others as well.

9. Appendices

Pat's Continuous Integration / Continuous Deployment workflow (WIP)

Pat's Guide to Git and Github (WIP)

Pat's Guide to GitHub Actions (WIP)

CSCE A401/ CSCE A601 (Advanced) Software Engineering

Pat, Jaaliyah, Boro, Eddie

Flashcard McFlashcardy Face

October 24, 2022

Architecture Requirements

- Breakdown of each team member's contribution to work
    - Pat (25%):
        - Scribe Work (project report)
        - Completed system architecture
        - Work on AWS integration
    - Jaaliyah(25%):
        - Designed front-end UML diagram
        - Worked on backend development
    - Boro (25%):
        - Processed UI Design
        - Started to code web pages using Vue.js and vuetify
        - Helped to design front-end UML diagram
    - Eddie (25%):
        - Reworked/edited system architecture and models writeups
        - Worked on backend development
- Status of the work:
    - Planned tasks for the next two weeks
        - Boro - Complete "Home", "About", "Team", and login/sign up pages
        - Pat - Back-End development, show Boro XHR functionality in JS.
        - Jaaliyah - Back-End Development, connection tools
        - Eddie - Front-End and Back-End Development
    - Tasks in progress:
        - Initial Interface Design
        - API Design
        - Back-end development
        - Front-end development
        - Documentation
    - Tasks completed:
        - Initial Design Specification
        - DevOps Architecture
        - CI / CD Pipeline Design
        - Initial Research
        - Product Specfications
        - Mockups / Wire drawings of UI
        - API preliminary design
        - Basic UI design