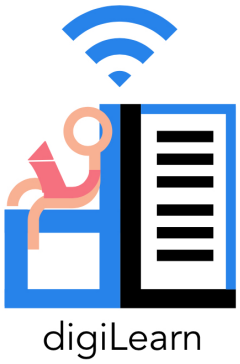
Team DigiLearn

Software Design v1.0:

The Digital Backpack



Doctor Morgan Vigil-Hayes
Volodymyr Saruta
Caitlin Abuel
Grave Shirey
Israel Bermudes
Kristine Hermosado
Sebastian Kastrul

5 February, 2021

Team DigiLearn

Table of Contents

Team DigiLearn

1.0 Introduction

The COVID-19 pandemic has led to a sudden shift to remote learning. Unfortunately, many students across America don't have access to a reliable Internet connection. The phenomenon known as “the homework gap” affects nearly 12 million students that are unable to fully participate in their coursework due to a lack of sufficient Internet access. Such a situation disproportionately affects disenfranchised communities. These students must rely on public hotspots to complete their assignments.

Dr. Vigil-Hayes runs the Community aware Networks & Information Systems Laboratory (CANIS) in the School of Informatics, Computing, and Cyber Systems at Northern Arizona University. CANIS Lab focuses on network analysis and community-centered design. Team DigiLearn is working with Dr. Vigil Hayes and CANIS labs to bring to life The Digital Backpack. The Digital Backpack or DigiPack is an app that will allow a fluid transition between online and offline learning. When a user comes into range of a wifi connection, the DigiPack will automatically download the requested content for offline use later. The app will also automatically upload completed assignments for the user. These upload and download requests can be queued offline to be performed when a network connection is available. The app will interface with popular Learning Management Systems such as Google Classroom.

This document outlines the overall software design plan for the Digital Backpack. Section 2.0 gives an overview of the project, broken down between an implementation overview as well as an architectural overview. The components of the software architecture are broken down into separate modules in Section 3.0. The first subsection, 3.1, focuses on the front-end components, while section 3.2 goes into detail about the back-end. An implementation plan is discussed in section 4.0, accompanied by a Gantt chart which outlines the weekly workflow goals for the project. This document concludes with section 5.0, which provides a review of the software design plan.

Team DigiLearn

2.0 Project Overview

This section gives a general overview of the project. Section 2.1 focuses on the implementation overview, which discusses the bigger picture of the product and the certain technologies and approaches that will be used to complete the project. Section 2.2 is the architectural overview, which will go into detail about the higher level aspects of the project. The architectural overview is broken down into two parts: an architectural diagram, and a discussion of the architecture.

2.1 Implementation Overview

The team solution is to create a mobile application (need to fix to just android and plugin) that will enable students to interact with educational materials seamlessly between online and offline environments. This solution will contain two main components: the user-interface application and the proxy server.

2.1.1 Tools and Technologies

To implement the DigiPack, we will use the tools and technologies listed below:

Front-End:

- Adobe XD: User-interface design
    - Adobe XD is a vector-based design for mobile and web applications. The tool's main goal is to make it easier for UI and EX designers to create designs, prototypes, and wireframes. We chose this tool for designing our mobile and web application development as it will help with multiple designs and can quickly create wireframe prototypes.
- Flutter: Cross-platform
    - Flutter is a cross-platform app development software created by Google. It allows both Android and iOS implementation under one codebase. The software uses an

object-oriented language called Dart, which has a similar C-style syntax. Since the team wants to build the application under one codebase to work on different devices, Flutter is the best option. The team chose this tool as the best approach to deal with our mobile and web application between Android, iOS, and chrome extension.

Back-End:

- Django: Web Framework
    - Django is an open-source web framework written in Python. It helps secure and maintain websites as it reduces the amount of work for web development. As the project promises a web application aside from the mobile application, Django is the best option for rapid and continuous development.
- Digital Ocean: Cloud service
    - DigitalOcean is a cloud hosting service built by developers with a simple interface to provide for easy configuration. With the benefit of being an easily configurable service, DigitalOcean is the best tool fit for the project.
- RESTful Service: Data handling
    - RESTful services are stated to an architectural style that contains fixed constraints. With the structure style, the server and the users can exchange resources using a standardized interface and protocols. The service also provides a structure to requesting web content and local standardization of content for ease of use and modularity. The project needs to push and pull resources from the services such as Google Classroom, Google Docs, Khan Academy, and Youtube, and RESTful service can help fulfill that need.
- MySQL: Databases
    - MySQL is a standard service for creating and managing databases. As MySQL is standard, familiar, and compatible with Django, it is a fit for this project.
- OAuth2: Security
    - OAuth2 is an industry-standard authentication framework. The framework allows sharing the same resources from one site to another without entering credentials again. OAuth2 is the tool chosen to resolve the issue of ensuring that all end devices in our system only communicate with authorized devices.

2.2 Architectural Overview

The core architecture of the DigiLearn service is largely unchanged from the Requirements Specifications Document. The diagram below shows a more granular depiction of each component, describing the flows of data and control within the system.

The majority of communications will be managed by the “client/server communications manager” (henceforth referred to as the CCM) and “resource/server communications manager” (RCM). These two managers form the two ends of communication between the client and the resources that DigiLearn provides access to. The resource/server communications manager processes data sent and received between Google Drive, Google Classroom, Khan Academy and the DigiLearn service. Requests, both from clients and to resources, are in the form of HTTP packets; the RCM and CCM, in the most basic sense, are web sockets used for sending and listening for HTTP packets.

The RCM when receiving data from a resource will send the data to its respective REST interpreter to be converted into the DigiLearn JSON format. The REST interpreters, described in section 3.2.3, are as their name implies: interpreters. Each interpreter will be responsible for converting data received from a resource into a format that the DigiLearn architecture can handle and vice versa. Since each resource has their own requirements for communication these interpreters are key to ensure that communication meets the standards for each resource, to create modularity allowing for new resources to be supported in the future, and to create homogenous data within the DigiLearn system. Homogeneity is achieved through the DigiLearn JSON structure providing a consistent format and ample context to any data transfer both within and outside of the DigiLearn architecture.

This JSON file will be the main mode of communication within the DigiLearn architecture. Any file transfer, request, authentication, or other communication will be done with the JSON file structure described in Section 3.2.4. Metadata for file transfers and requests: upload/download dates, file sizes, what resource is involved in the transfer or request, along with metadata describing the communication itself: user identification, permissions, associations, asynchronous communication tags, and more will be stored in each JSON file. This will allow any part of the system to use the same file and structure to serve the user and create an architecture with relatively low coupling.

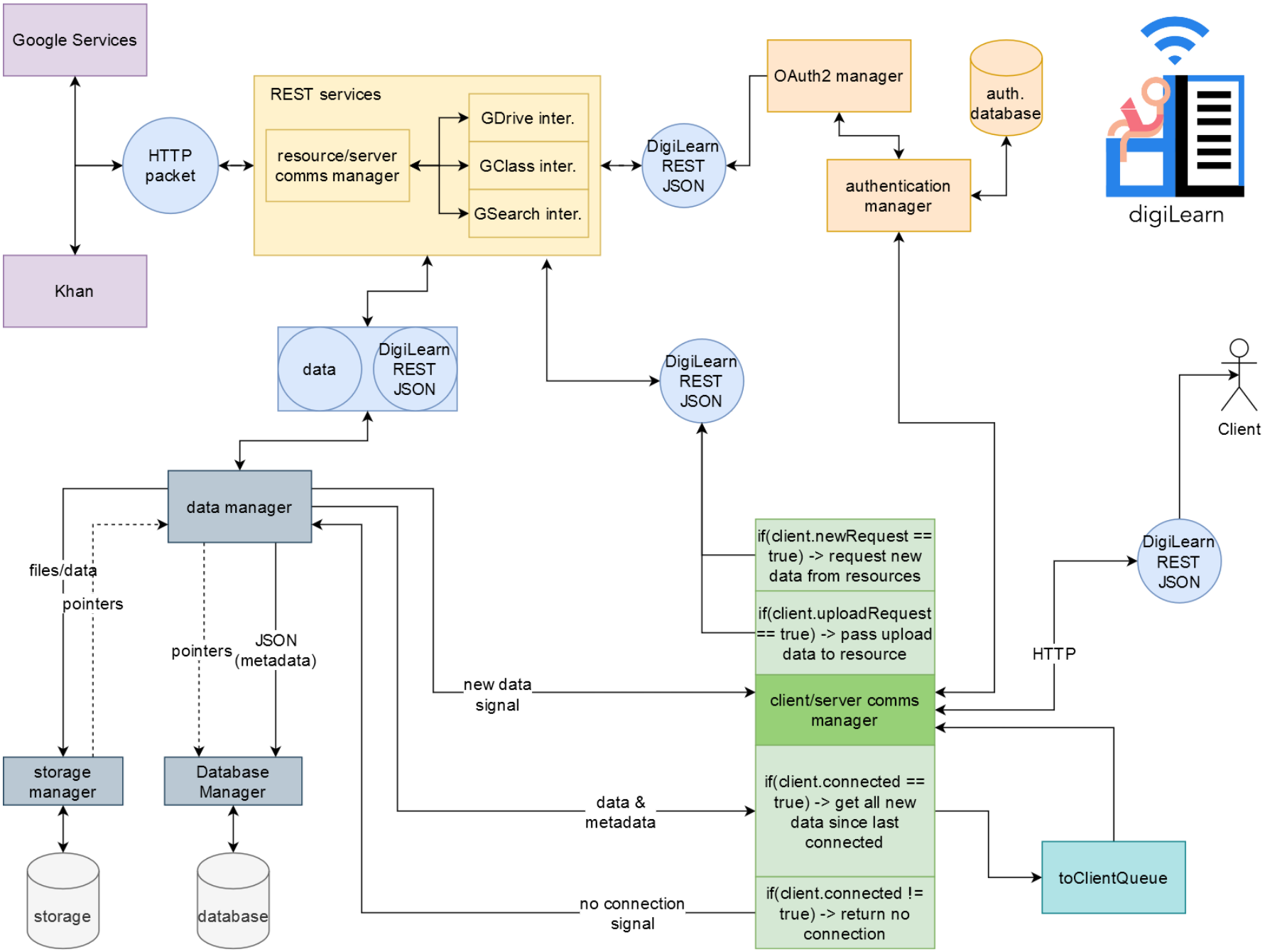


Figure 2.2.1: DigiLearn Architectural Overview Diagram.

Internally, the Data Manager (DM) will be the main source and sink for communication with the client(s). Documents requested by users will be sent to the DM by an interpreter, which will then pass the document to the storage manager to be held until the user is able to download it to their device and/or it expires. The storage location will be passed back to the DM and added to the JSON file for later use. The JSON file is then parsed for the metadata required by the DigiLearn Database and passed to the Database Manager.

The metadata will be used to create database entries allowing the DigiLearn service to track file and user associations, storage locations, request entries, and more. Storing this data in a database allows for JSON files to be created and destroyed dynamically, freeing up space on the server and providing much needed search and sort functionalities to the DigiLearn architecture.

Before the JSON file is destroyed after being added to the database, the DM sends the file to the CCM. The CCM will then check if any of the associated users are available to receive the data associated with the file and if not simply returns the file. If a user is available the CCM already has all of the information necessary to send the user their data by requesting the document from the Data Manager and passing it to the user.

User availability in the context above is determined by the CCM and the Authentication Manager (AM) described in Section 3.2.6. A user[1] connects to the DigiLearn server initially by sending an HTTP request to the server stating their user ID, their password, the time they sent the request, and the last time they connected. The CCM passes the required information to the AM where it is referenced against the Authentication Database. The user password is compared against a hashed password stored in the Authentication Database. If the user is who they say they are, the connection is opened.

This open connection, or “session”, starts a series of events to serve the user as much as possible while the connection persists. First a confirmation of connection is sent to the user; this confirmation signals the user to send any pending requests or missing file markers back to the server to be filled. At the same time, the CCM sends the users information to the DM which in turn compiles a list of all of the files that have been acquired for the user since they last connected. Once this list is compiled it is sent back to the CCM to be processed by the Client Queue (CQ) and sent to the user.

Due to the possibility of connection loss in the middle of a transfer, the first JSON file sent to the user contains all metadata about the files they are about to receive. The user's device uses this to determine if the whole file is received during the transfer and if not, can mark which pieces of files or whole files that are missing from the transmission. Users are able to make search queries,

[1] User in this context refers to the DigiLearn application on a users device, most of the process described here is not shown to the actual user.

mark documents to upload or submit as assignments, and request specific documents from their accounts while offline through the DigiLearn application. These requests are stored on their device until they connect to the DigiLearn service next and they are uploaded to the server along with the missing file markers.

The CCM begins sending the user all of the files that the server has requested for them since their last connection and at the same time begins populating a new query to the DM. This query gets all of the pieces or whole files that are missing from the previous transmission from the DM and adds them to the beginning of the queue to send to the user. Any new requests or uploads are sent to their respective REST interpreters to be passed to the requested resource and the process starts again.

Team DigiLearn

3.0 Module and Interface Descriptions

This section breaks down the overall system of the Digital Backpack into individual modules and goes into detail about how each of these sections fit into the bigger picture. The main components can be separated by the front-end functionalities of the app, and the back-end functionalities.

3.1 Front-End

The front-end of the Digital backpack allows the user to experience a high quality app, working with the user interface and platform development.

3.1.1 Mobile application for users

The mobile application module focuses on the user interface which the users will be interacting with. The application allows users to download, submit, and search educational materials. The UML below shown in Figure [some number] is the mobile application components.

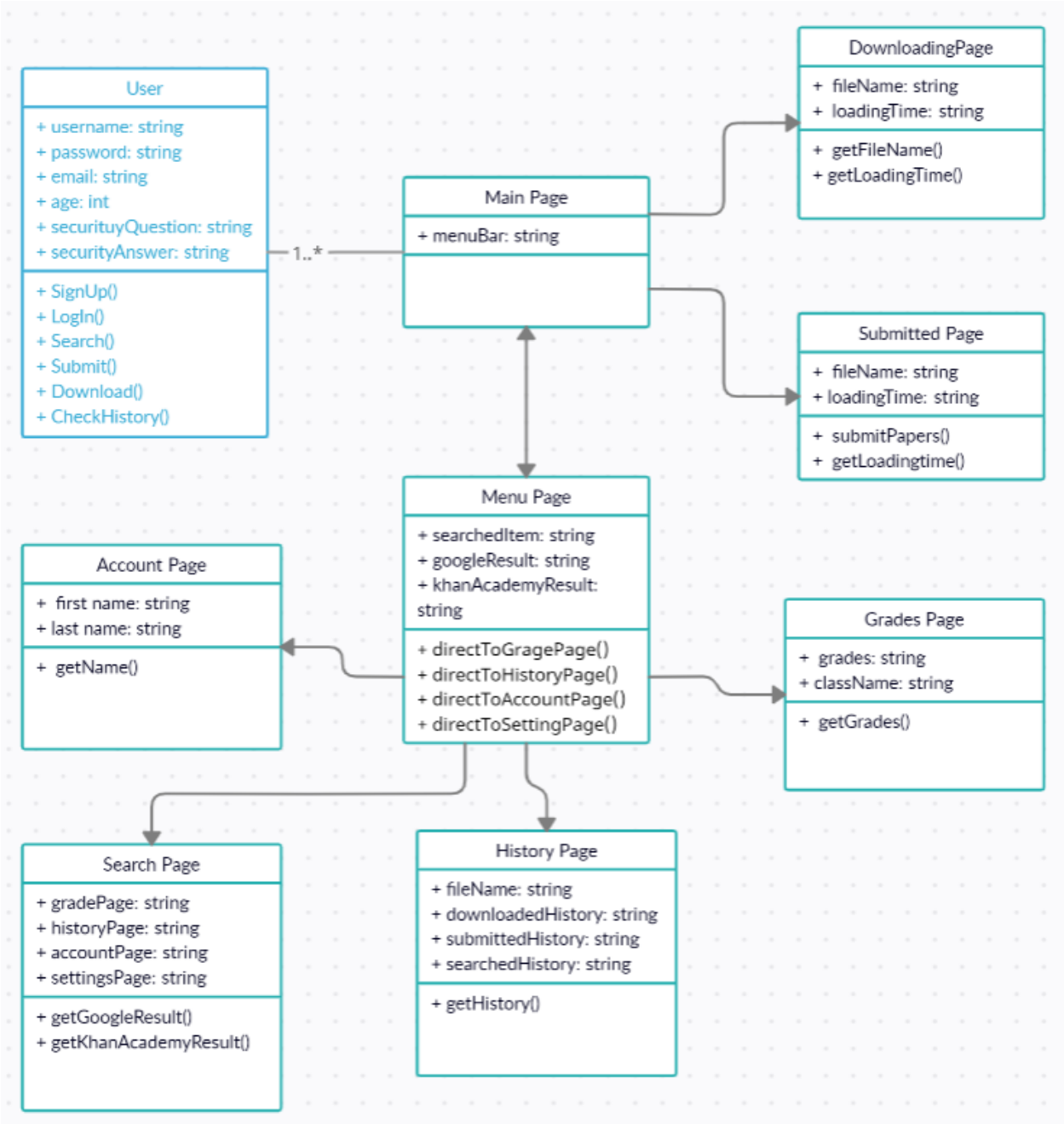


Figure 3.1.1.1 : Mobile Application UML (ROUGH DRAFT UML)

Further explanation about the components of the UML is discussed below:

Users

The users are the students in the UML design. The students are the one interacting with other classes within the UML. They are able to log in, view download progress, view history, view grades and lastly, do Google and KhanAcademy search. The user has the attributes and behaviors listed below:

- username - The user will create a username and it will be used to uniquely identify them.
- password - A form of security that gives authorization to access the account. Only the user who created the account can enter.
- email - The email will be saved to the database after account creation for security purposes.
- age - The user's age is an essential aspect of the project because certain risks pertaining to age and legal limitations could arise if care is not taken. It will also determine what kind of UI the user will get.
- securityQuestion - The purpose of this attribute is for security purposes, in the event of the user forgetting their account information, the user can regain their account through this questionnaire.
- securityAnswer - This is connected to the securityQuestion attribute as it will determine if the user is the right account owner.
- SignUp() - The first thing users have to do is create an account.
- LogIn() - After the user has established their own account, the users can easily log in.
- Search() - One of the features of the application is allowing users to be able to search through the given platform such as Google search.
- Submit() - The users will be able to submit their educational documents.
- Download() - After doing a search, the user can download the materials they wish to.
- CheckHistory() - The user will be able to check their history logs of their downloaded and searched documents.

Main Page

The main page is where the users are going to be redirected after signing up or logging in. The main page would have a relationship with the menu bar page and the account page. From the main page, the users have the option to view the downloads, submission and search page. The main page also contains the menu bar that will give more options to do. The main page has the attributes and behaviors listed below:

- menuBar - The user would be able to access other pages through the menu bar.

Downloading and Submitted Page

The downloading and submitted page is part of the main page as Both pages have similar attributes and behaviors but the downloading page requires getting the time left on the certain file. The attributes and behaviors for both pages are listed below:

- fileName - The name of the file the user is going to download.
- loadingTime - This should display the time it takes for a certain file to download or submit.
- getFileName() - The method should grab the name of the file downloading.
- submitPapers() - The method should submit the files that the user has chosen.
- getLoadingTime() - The method should be able to calculate the amount of time the file is taking.

Menu Page

The menu page contains optional operations such as going to the account, history, and setting pages. The menu page is accessed through the main page on the top left corner of the interface. The menu page has the attributes and behaviors listed below:

- gradePage - The users will be able to check their grade through this attribute.
- historyPage - The users will be able to check their history through this attribute.
- accountPage - The users will be able to check their profile through this attribute.
- settingsPage - The user will be able to go to the settings through this attribute.
- directToGradePage() - The method should redirect the users to the grades page.
- directToHistoryPage() - The method should redirect the users to the history page.
- directToAccountPage() - The method should redirect the users to the account page.
- directToSettingPage() - The method should redirect the users to the setting page.

Account Page

The account page is a basic page that would only contain the user's first and last name. The account page has the attributes and behaviours listed below:

- first name - The user's first name is displayed.
- last name - The user's last is displayed.
- getName() - The method should get the user's first and last name from the database.

Search Page

The search page is where the user can use the search feature. The search page has the attribute and behaviours listed below:

- searchedItem - The name of the searched item.
- googleResult - The result the google search would pop out.
- getGoogleResult() - The method should grab the result that the Google search produces.

History Page

The history page is where the downloaded and searched files are displayed. The history page has the attribute and behaviours listed below:

- fileName - The name of the file that the user has previously downloaded.
- downloadedHistory - The section where the downloaded files can be viewed.
- submittedHistory - The section where the submitted files can be viewed.
- searchedHistory - The section where the search files can be viewed.
- getHistory() - The method should be able to grab the history files for each section.

Grades Page

The grade page is where the student's grades are going to be displayed. The grades page has the attributes and behaviors listed below:

- grades - The grade of the user for that certain class would be displayed.
- className - The class names of the user would be displayed.
- getGrades() - The method should grab the user's grades with the class name.

3.1.2 Web application for admins

The web application focuses on the user and admin side of the application. The web application will be able to use Google plug-in's to allow multiple functions for the admin. These functions include access to a login page that directs the admin to the main page and allows the admin to view the users search history, and downloaded history, as well as be able to filter what can be viewed by the user. All of the functions and connections of the web application are illustrated in the figure below.

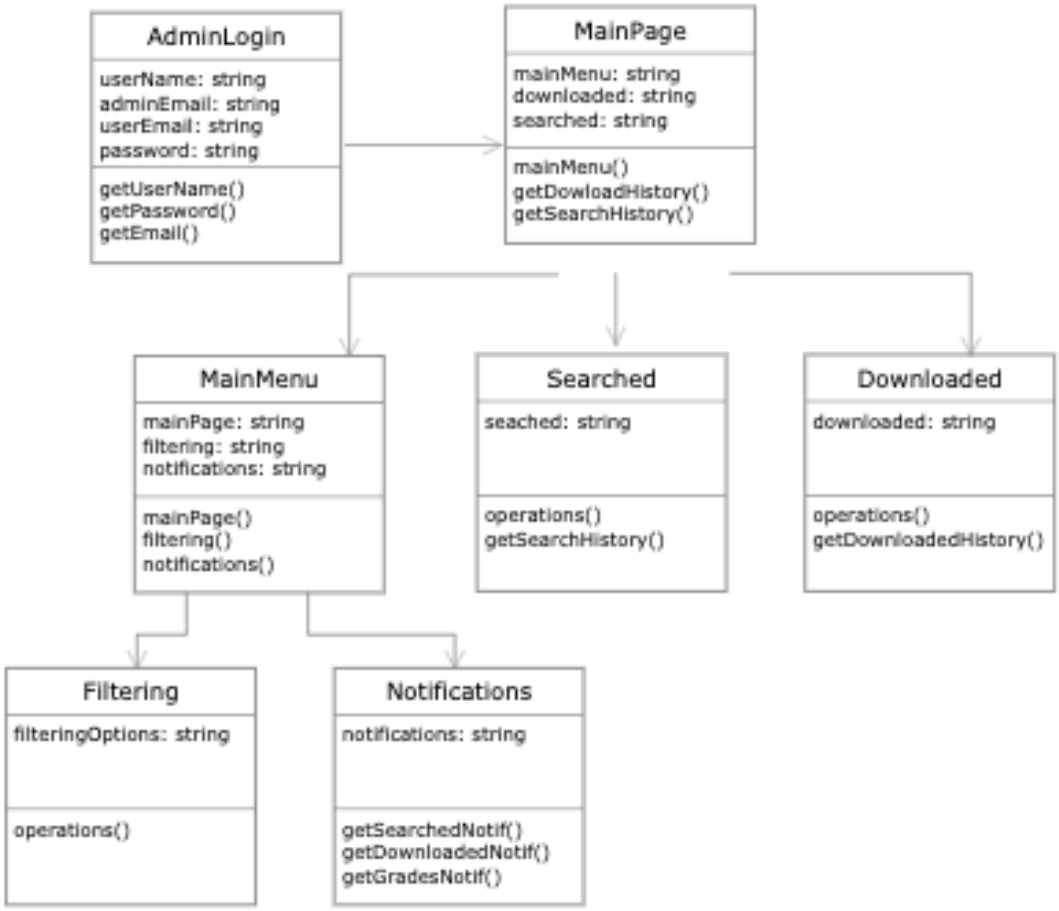


Figure 3.1.2.1 : Web Application UML Diagram

3.1.2.1 Admin Login

The administrator in the diagram above is the parent of the child or user. The administrator will need identification to verify their accessibility to view the users account. Key attributes for the admin login are:

- userName: The administrator will use the username of the user they are granted access to in able to be verified to log in.
- adminEmail: The administrator will need to provide their email address for identification and authorization.

- userEmail: The administrator will need the user's email address in order to access views to their accounts.
- password: The administrator will need the user's password in order to access the view to their accounts.

Key operations for the administrator are:

- getEmail( ): Necessitates the administrator to enter their email address and the user's email address in order to create an account.
- getUserName( ): Necessitates the administrator to enter the username of the user they are viewing in order to gain access to their account.
- getPassword( ): Necessitates the administrator to create a password when creating an account and using that password to login to their account.

3.1.2.2 Main Page

After the administrator is logged in, they will be redirected to the main page, which includes the primary features of the application as well as a main menu bar that is able to direct them to different pages. Key attributes of the main page are:

- mainMenu: The administrator will be able to view a main menu bar that gives them access to click to the other pages of the application.
- downloaded: The administrator will be able to view the history of items downloaded by the user.
- searched: The administrator will be able to view the search history of the user.

Key operations of the main page are:

- mainMenu( ): Provides the main menu bar on each page, and includes links to the other pages.
- getDownloadedHistory( ): Receives and allows the administrator to view the user's downloaded history.
- getSearchedHistory( ): Receives and allows the administrator to view the user's search history.

3.1.2.3 Main Menu

The main menu will be located on the side of each page on the application. The main menu will include links to all other pages on the web application. Key attributes of the main menu are:

- filtering: The administrator will be able to click a link that routes them to a page that provides them filtering options.
- notifications: The administrator will be able to view notifications of recent downloaded and searched items from the user.
- mainPage: The administrator will be able to click a link that routes them to the Main Page of the application.

Key operations of the main menu are:

- mainPage( ): Allows administrators to link back to the main page.
- filtering( ): Allows the administrator to link to a filtering page.
- getNotif( ): Receives recent searched and downloaded history and allows the administrator to view notifications of the history.

3.1.2.4 Searched Page

The searched page is located on the main page. This page allows the administrators to view what the user has searched. Key attributes for the searched page are:

- searched: The administrator will be able to view the search history of the user

Key attributes for the searched page are:

- getSearchedHistory( ): Receives the recent user search history and allows the administrator to view the user's search history

3.1.2.5 Downloaded Page

The downloaded page is located on the main page. This page allows the administrators to view what items the user has downloaded. Key attributes of the downloaded page include:

- downloaded: The administrator will be able to view the history of items downloaded by the user

Key operations of the downloaded page include:

- getDownloadedHistory( ): Receives the recent downloaded history allows the administrator to view the user's downloaded history

3.1.2.5 Filtering

The filtering page can be accessed through the main menu and is used to allow the administrator access to filter the content that the user is searching and downloading. Key attributes and operations for filtering include:

- filteringOptions: The administrator will be able to filter search and download settings for the user

3.1.2.5 Notifications

The notifications allow the administrator to view the most recent searches and downloads from the user. Notifications can be viewed through the main menu. Key attributes for the notifications include:

- notifications: Administrators will be able to view notifications of recent downloaded and searched items from the user.

Key operations for the notifications include:

- getNotif( ): Allows the administrator to view notifications of the recent searched and downloaded history.
- getSearchedHistory( ): Receives the recent user search history and allows the administrator to view the history
- getDownloadedHistory( ): Receives the recent user downloaded history and allows the administrator to view the history

3.2 Back-End

The back-end of the Digital Backpack handles the functionalities necessary for opportunistic content delivery.

3.2.1 Client/Server Communication Manager

Server-Client Communication is a significant part of the Digital Backpack back-end. Metadata will be sent between the client and the server to communicate important information such as: the connection date, requested data, and requested data size. This information can then be used to determine what data is missing when connections are disrupted. This server-client communication will be implemented by utilizing Websockets, which allow for bi-directional communication.

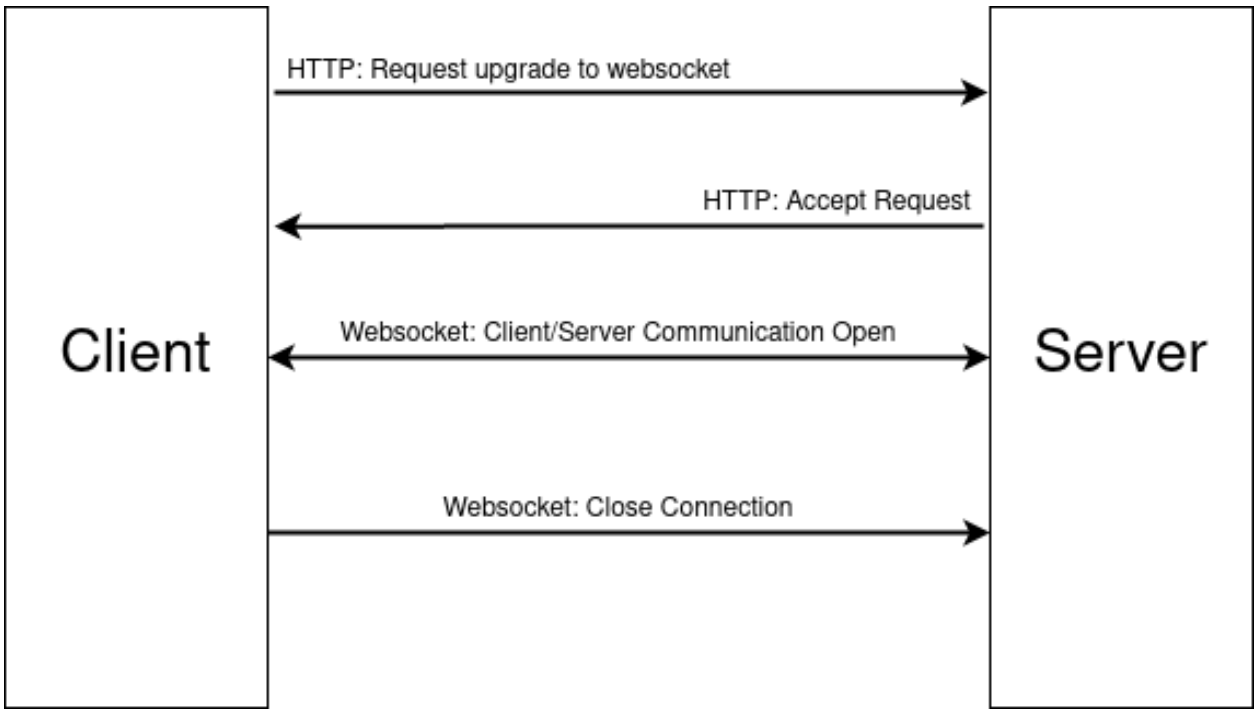


Figure 3.2.1A: Websocket Protocol Between the Client and Server

Figure 3.2.1A shows a visual representation of the Websockets protocol that will be used for the server-client communication. First, the client makes an initial HTTP request to be upgraded to websockets. The server responds to this request and opens a persistent websocket connection between the server and client. Metadata will be exchanged during the bi-directional communication.

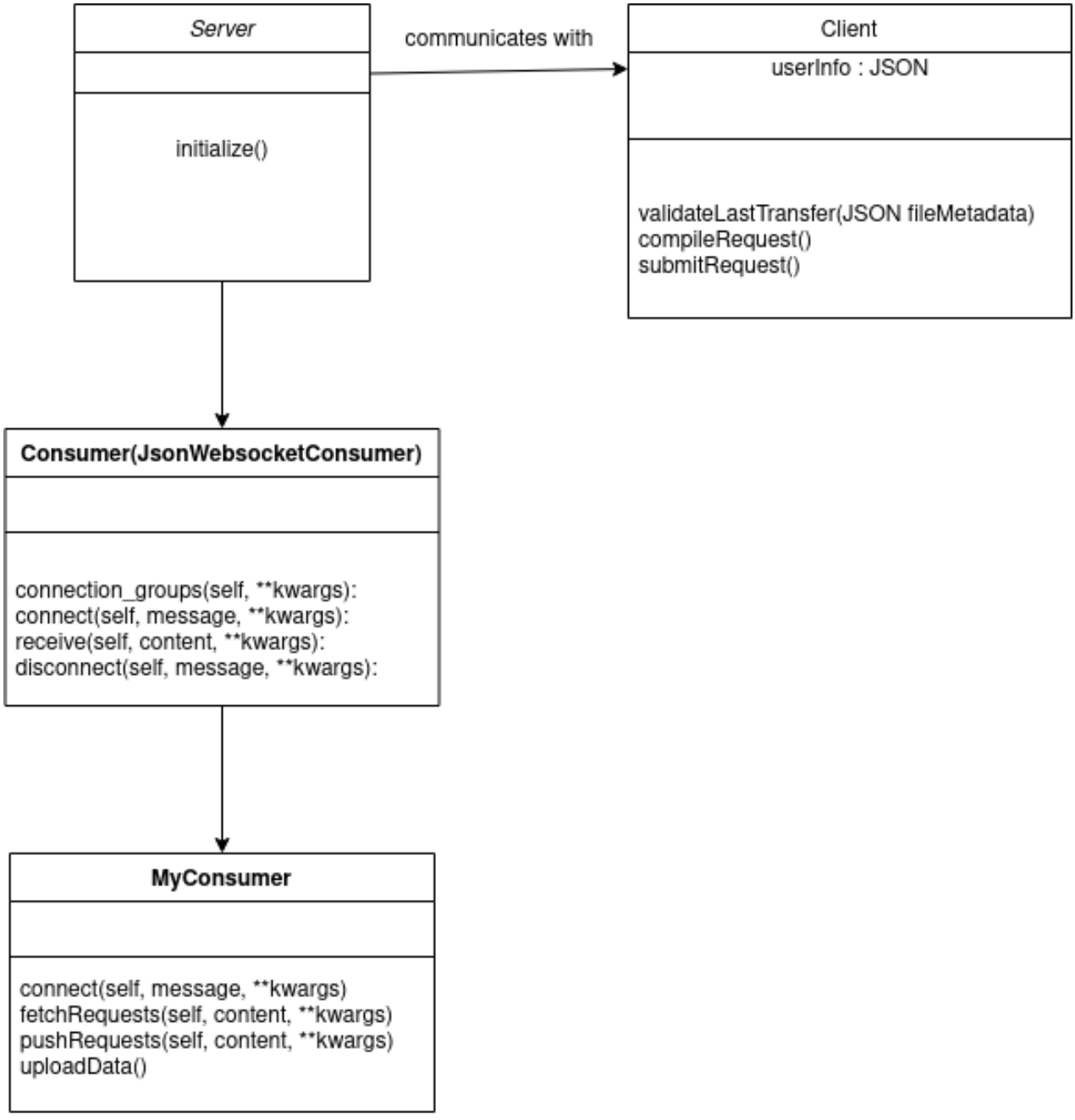


Figure 3.2.1B: Server/Client Communication UML

The client/server communication manager consists of the server, client, and consumers.

Server

The server represents the central unit for processing user requests. Most of the work required by the server is deferred to consumers that handle requests. The server has the following method:

- **initialize() -** Initializes the server.

Client

The client represents the user that interacts with the server to make upload or download requests. The client has the following fields and methods:

- **userInfo** - A JSON file that contains information about the user as well as metadata pertaining to requested file transfers. Authentication information is also stored in this JSON, which will be used by the server to authenticate the user's identity.
- **validateLastTransfer(JSON fileMetadata) -** This function takes in a JSON file that contains metadata about a file transfer. Using this, the client device will determine whether or not the file transfer was complete and successful.
- **compileRequest() -** While offline, any requests for uploads or search queries that the user makes will be compiled on the user's device until a connection is established and the requests can be uploaded to the server.
- **submitRequest() -** When a connection is achieved, all compiled requests on the user's device will be uploaded to the server.

Consumer

The consumer handles tasks that are designated by the server. The consumer functions are based on Django's built-in class-based websocket consumers. Operations done by the consumer are handled by the asynchronous task queue. (See Section 3.2.2 for further details about the Task Queue)

- **connect(self, message, **kwargs) -** Accepts the connection and attempts to authenticate the user. If the user is authenticated, they will have access to the application's functionalities.
- **authenticateUser(self, content, **kwargs) -** This function takes a user info JSON file and passes the necessary information to the Authentication Manager. (See Section 3.2.7 for further details about the Authentication Manager)
- **fetchRequests(self, content, **kwargs) -** Using the user info JSON file containing file requests, the consumer passes the relevant information to the Database Manager, which will compile the requested files that are available for the user. (See Section 3.2.5 for further details about the Database Manager)

- **pushRequests(self, content, **kwargs) -** When the consumer receives the compiled list of requested files from the Database Manager, it will begin pushing these files to the user's device.
- **uploadData() -** Upload requests from the user will be uploaded to the DigiLearn server.

3.2.2 Task Queue Module

The task queue will handle asynchronous processing as requests are made by the user.

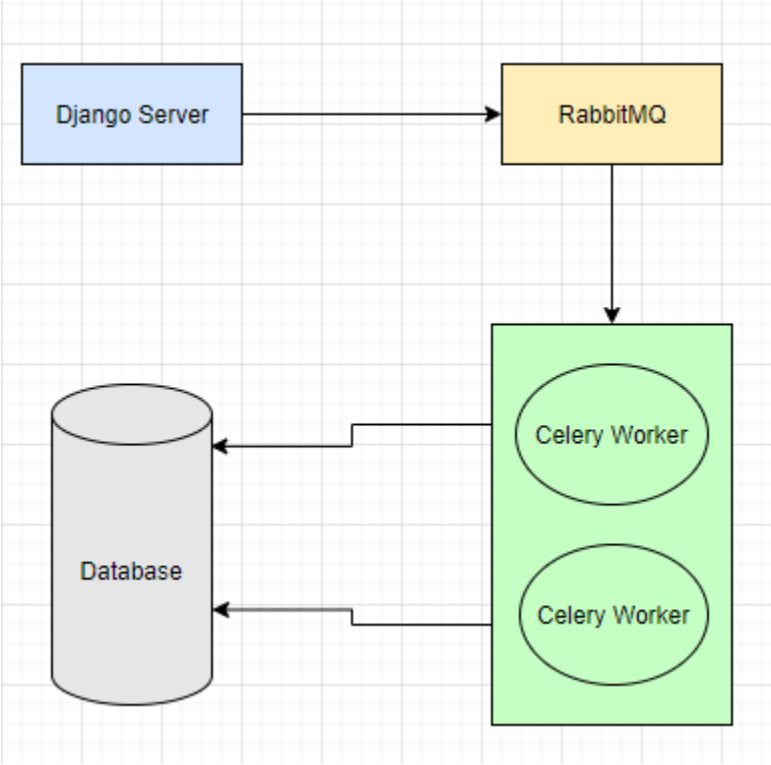


Figure 3.2.2: Celery Queue Workflow

The overall Task Queue Module consists of the Django server, a message broker, workers, and a results storage. Celery, the task queue manager, is made up of the message broker, workers, and results storage.

Module Components

- **Django Server** - The Django server defers tasks to Celery for background processing.
- **RabbitMQ** - RabbitMQ serves as the designated message broker for the system. It handles exchanges between messages and task distribution between workers.

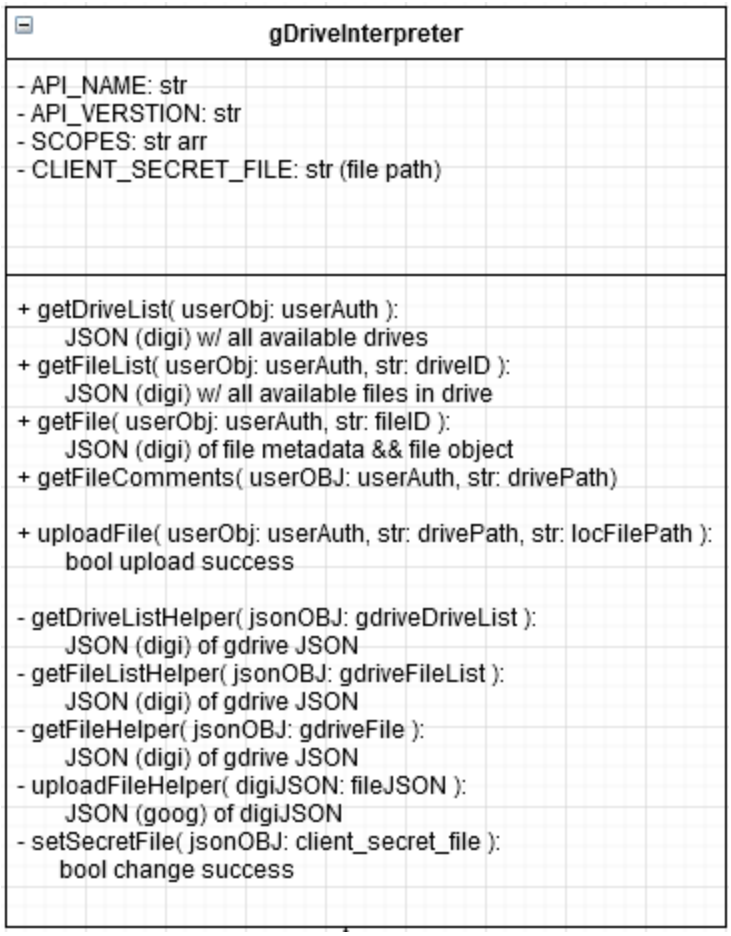
- **Celery Worker** - Each Celery worker will execute the task it is designated
- **Results Storage** - A basic data structure that can be used to store the results of an executed task.

3.2.3 REST Interpreters

The DigiLearn service will support **three** resources: Google Drive, Google Search, and Google Classroom. Each resource is able to send and receive different data with a multitude of unique formats, data, and values; therefore each resource will need its own interpreter to convert the data received from each resource into a format that is usable by the DigiLearn server and conversely convert the DigiLearn format into something usable by the resource. These interpreters, in tandem with the Server-Client Communication Module described in Section 3.2.1 form the inlet and outlet for data between the DigiLearn server and supported resources.

3.2.3.1 Google Drive

The Google Drive interpreter converts JSON objects between the DigiLearn format and Google Drive format.



- **Public getters**

  Take in user authentication values needed for proper connection to the resource and necessary data to serve the request. All return a DigiLearn JSON object containing all values received by the server in response to the request.

- **Private getterHelpers**

  Take in DigiLearn format JSON objects and return Google Drive JSON format objects.

- **uploadFile**

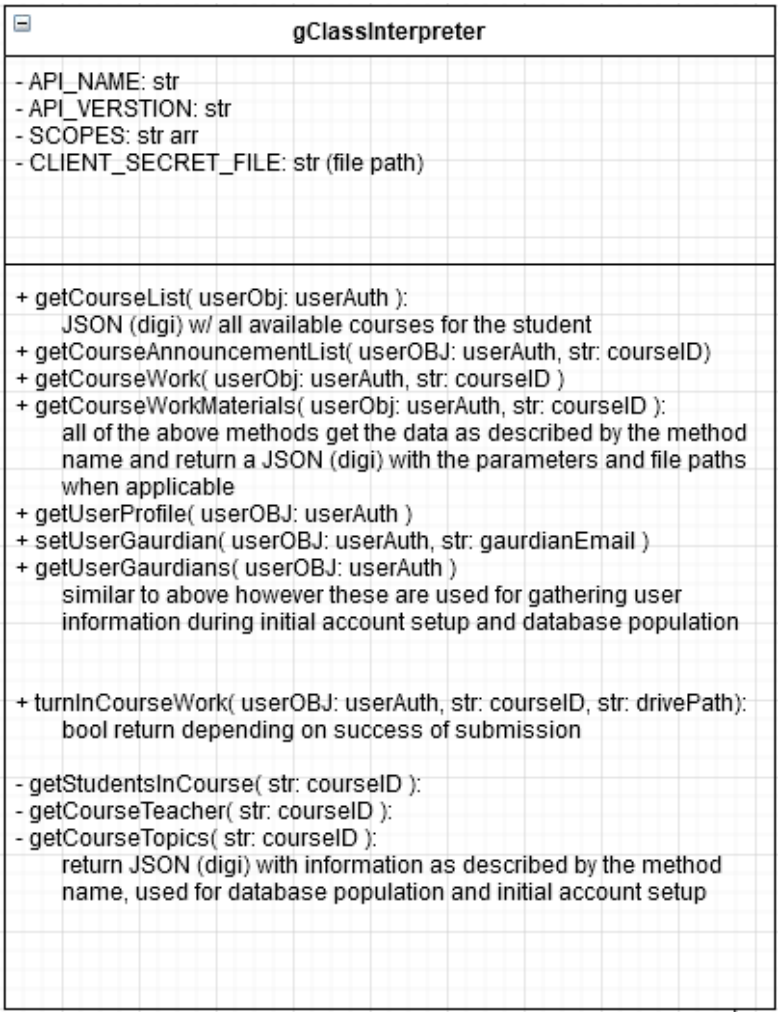
  Takes in DigiLearn format JSON file object and converts it with the help of uploadFileHelper to Google Drive JSON format object and receivable data

- **setSecretFile**

This method is used for switching between user accounts based on the user object passed into any of the other methods.

3.2.3.2 Google Classroom

The Google Classroom interpreter converts JSON objects between the DigiLearn format and Google Classroom format.



- **Public getters**

  Take in user authentication values needed for proper connection to the resource and necessary data to serve the request. All return a DigiLearn JSON object containing all values received by the server in response to the request.

- **Private getters**

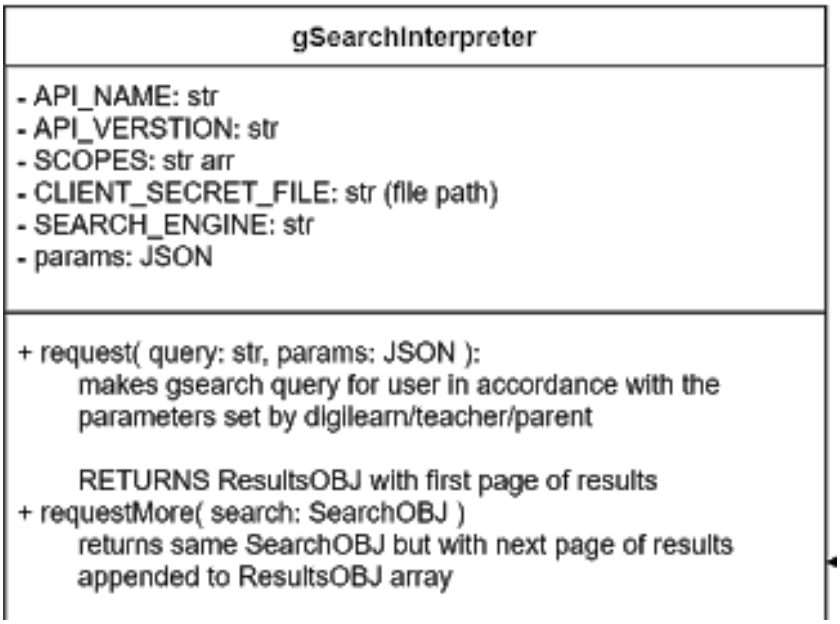
These methods are used to fill in information to better describe the JSON objects returned by the public getters. They are private because they are used by the public getters to create more complete data for later use in the database described in Section 3.2.5.2

- **turnInCourseWork**

  This method is used to convert DigiLearn JSON file objects into Google Classroom JSON objects with the necessary parameters for submission for a specific assignment.

3.2.3.3 Google Search

The Google Search interpreter converts JSON objects between the DigiLearn format and Google Search format.



- **Request**

  Makes a request on behalf of the user to the Google Search engine with the restrictions set by DigiLearn, teachers, and parents through Google Classroom. Returns JSON object with the first page of results. Queries can be made for both text and image searches through the Google Search engine.

- **requestMore**

  If requested by the user, up to ten (10) pages of results can be returned to the user. This limit is put in place by the Google Search API.

3.2.4 DigiLearn JSON

The DigiLearn JSON format is created dynamically by the system, adding objects and values as available or necessary. At the core are nine (9) main types of objects with two “superclasses” to organize them. The superclasses, denoted “Authentication” and “Session”, are used to describe the user and their connection status respectively. In the Authentication superclass user ID’s, hash keys, authentication tokens, and other information necessary to ensure that the users connection is secure and the DigiLearn server can properly and legally connect to their account(s). The Session superclass is used to describe the users status with the server. Information such as the date and time they last connected to the server, if they are currently connected to the server, when they connected to the server for this session, what data has been pulled for them since the last time they connected, and any requests they may have made for the DigiLearn server to get for them this session.

This JSON file is used exclusively for transfer between the DigiLearn server and Digital Backpack users. Within the DigiLearn server architecture the data represented in the JSON file is stored either in the database and storage described in Section 3.2.5 or as Python objects.

Due to the physical size of this file due to formatting for readability it is attached to this document, the objects that are used to build the JSON file are shown below:

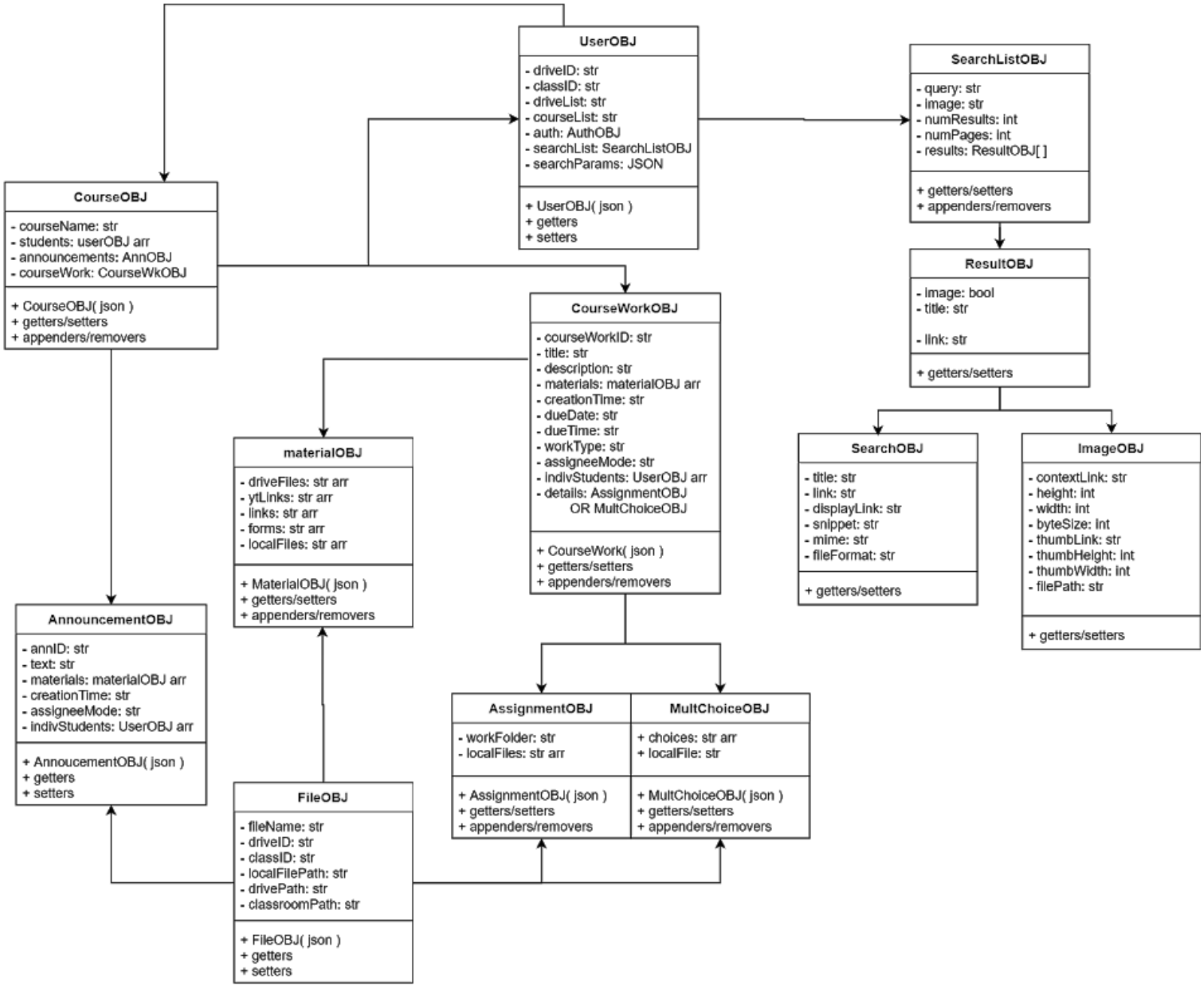
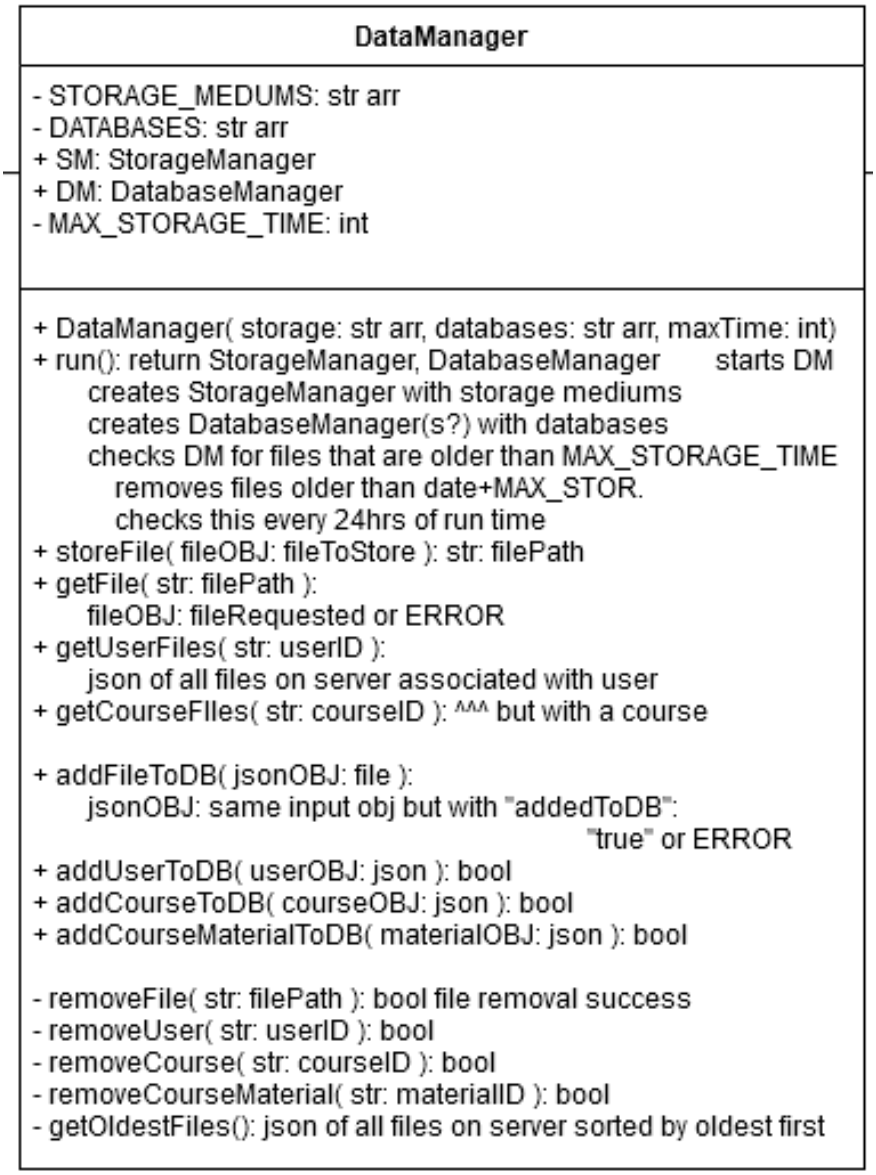


Figure 3.2.4.1: Digital Backpack JSON Diagram

3.2.5 Data Management

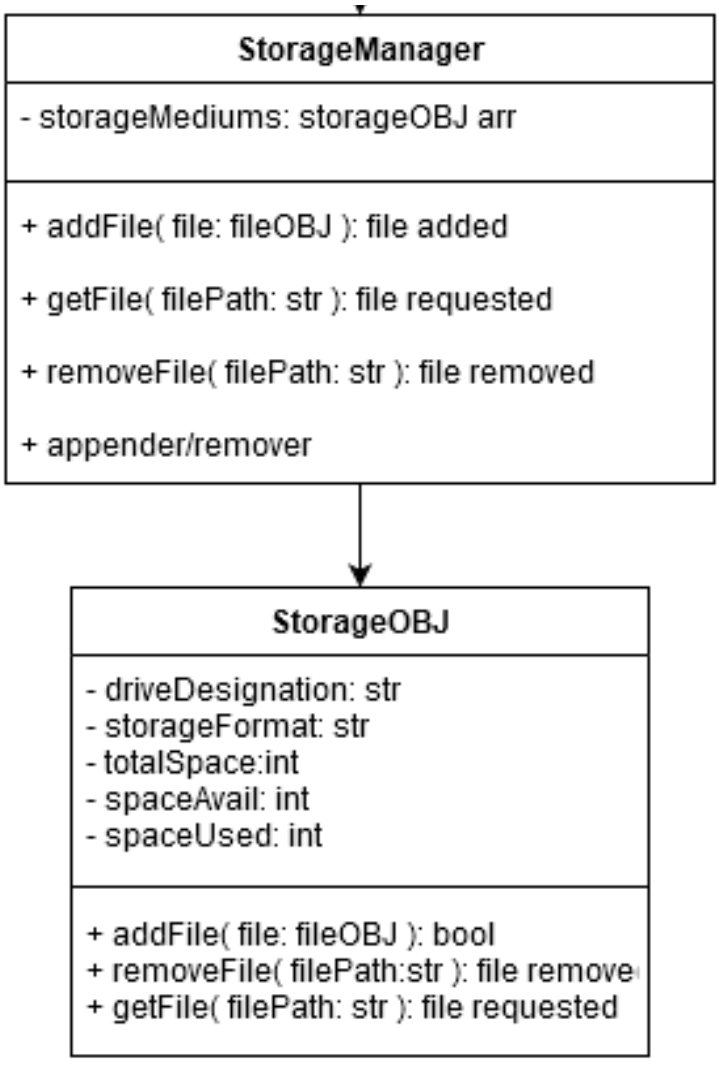
Storing data and user associations with said data on the DigiLearn server is crucial to ensuring that users can be served efficiently. Associations between files, users, teachers, classrooms, assignments, and more will be handled by the Database Manager. Files and documents that need to be stored will be controlled by the Storage Manager. The Data Manager itself will be the main source and sink for data within the DigiLearn architecture and provides a multitude of functionalities to facilitate this.



Adding and returning (or “getting”) files from the Storage Manager as well as adding users, files, and courses to the database via the Database Manager are available for use by any part of the architecture with permissions to access the Data Manager. Removing things however, is only done by the Data Manager. The Data Manager automatically removes files and their entries in the database after a specified amount of time when the Data Manager is initialized. This ensures that data isn’t accidentally deleted by another program, and that used space on attached storage mediums is kept to a minimum.

3.2.5.1 Storage Manager

The Storage Manager will be used to control the flow and storage of files and documents within the DigiLearn server. The Storage Manager represents a set of methods and controls to be used by the Data Manager to create, store, delete, and transfer data to and from supported resources and users.



3.2.5.2 General-Storage Database Manager

The Database Manager acts as an interface between the general-storage database and the rest of the DigiLearn architecture. The General-Storage database is described below in section 3.2.5.3. The database manager is implemented using Django's database API. The database itself is implemented using MySQL.

3.2.5.3 General-Storage Database

The general-storage database stores minimal user data such as a unique student identifier, class enrollment, and assignment submissions. The general-storage database is implemented in accordance with the entity-relationship diagram below in figure 3.2.6.1.1.

The following subsections clarify the entities, their attributes, and their relationships with each other.

- **student**

  The student entity refers to a student served by the Digital Backpack. The studentId is a unique identifier throughout the Digital Backpack system. This entity is associated with the classes it is enrolled in, resources transferred to the student, assignments submitted by the student, and Google searches submitted by the student.

- **class**

  The class entity refers to scholastic classes served by the Digital Backpack. Classes are associated with assignments and additional resources which are provided to the students. Classes are also associated with the students that are enrolled in them.

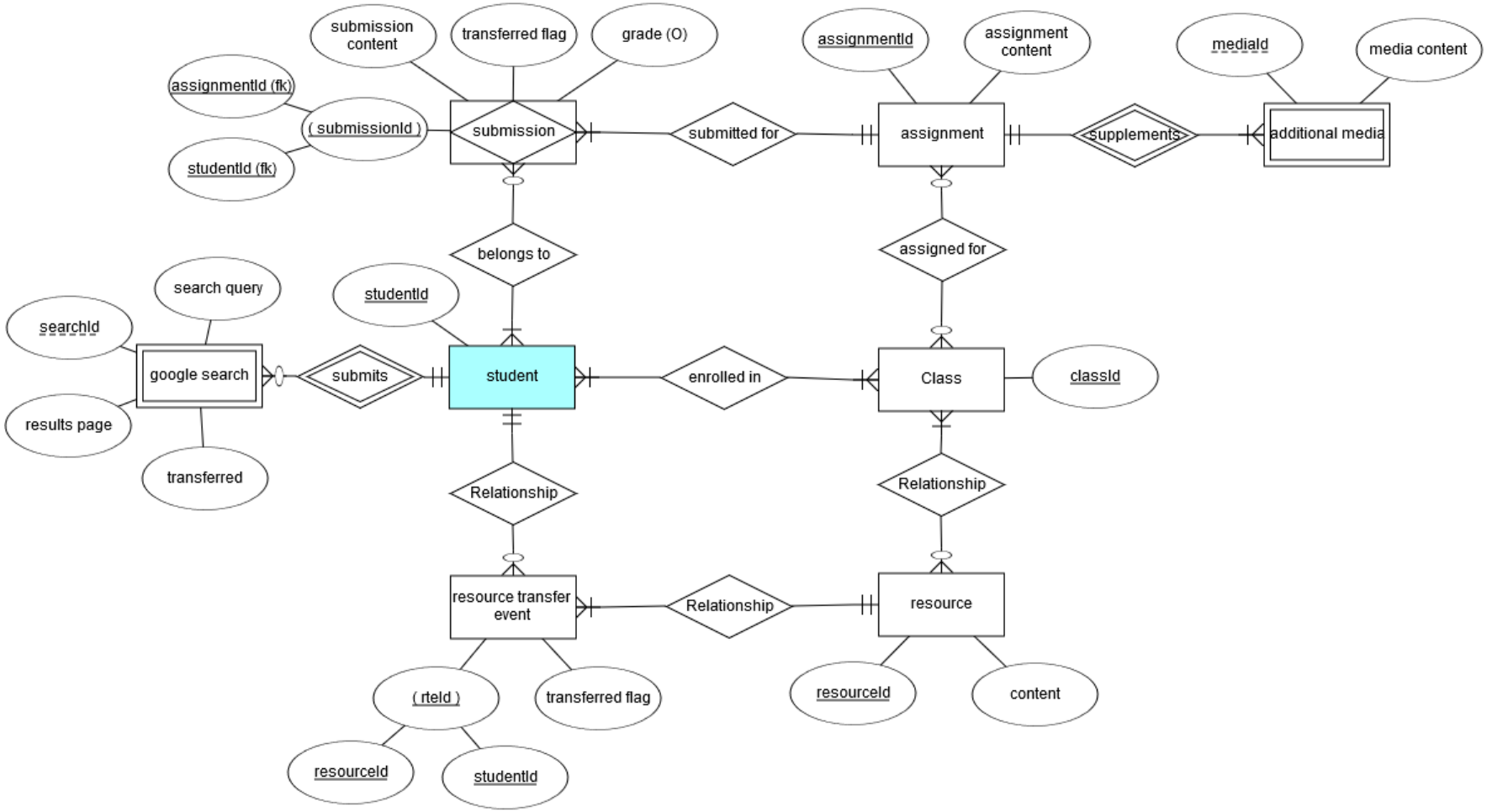


Figure 3.2.6.1.1: General-Storage Database Entity-Relationship Diagram.

- **assignment**

  The assignment entity refers to any assignment that has been assigned to the entire class. The assignment entity contains the content of the assignment. The assignment is associated with the submission entity, which defines the relationship between a student and the assignment. The assignment entity is also associated with the additional media entity which defines any supplemental resources that are to be included with the assignment.

- **submission**

  The submission entity defines the relationship between a student and an assignment. The submission entity contains the student's submission for the assignment, their grade, and a transferred flag. The transferred flag informs the Digital Backpack on whether assignment information needs to be transferred to or from the student.

- **additional media**

  The additional media entity defines any additional resources that are to be distributed with a given assignment. This may include supplemental readings, videos, and so on. Additional media is associated only with the assignment it belongs to.

- **resource**

  Similar to additional media, the resource entity defines any additional resource that is to be distributed to the members of a given class. This may include supplemental readings, videos, and so on. Additional media is associated only with the class it belongs to.

- **resource transfer event**

  The resource transfer event defines the relationship between a student and a resource. The transferred flag informs the Digital Backpack on whether a resource needs to be transferred to the student.

- **google search**

  The google search entity defines Google search requests submitted by a student. As above, the transferred flag informs the Digital Backpack on whether information needs to be transferred to or from the student.

3.2.5.4 Authentication Database

The authentication database stores only student IDs and the associated authentication and refresh tokens. The authentication database is implemented in accordance with the entity-relationship diagram below in figure 3.2.6.2.1

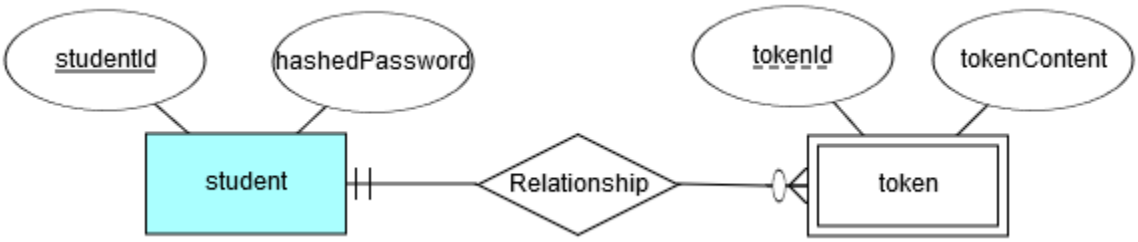


Figure 3.2.6.2.1: Authentication Database Entity-Relationship Diagram.

The following subsections clarify the entities, their attributes, and their relationships with each other.

- **student**

  The student entity defines a student served by the Digital Backpack. The studentId is a unique identifier throughout the Digital Backpack system. Student passwords are stored in a hashed form here for authentication purposes.

- **token**

  The token entity refers to any authentication or refresh token. The tokenId is described in section 3.2.7.1. The tokenContent attribute contains the token itself.

3.2.6 Authentication Management

The management of OAuth2 tokens is a cooperative effort between the proxy server and the client application.

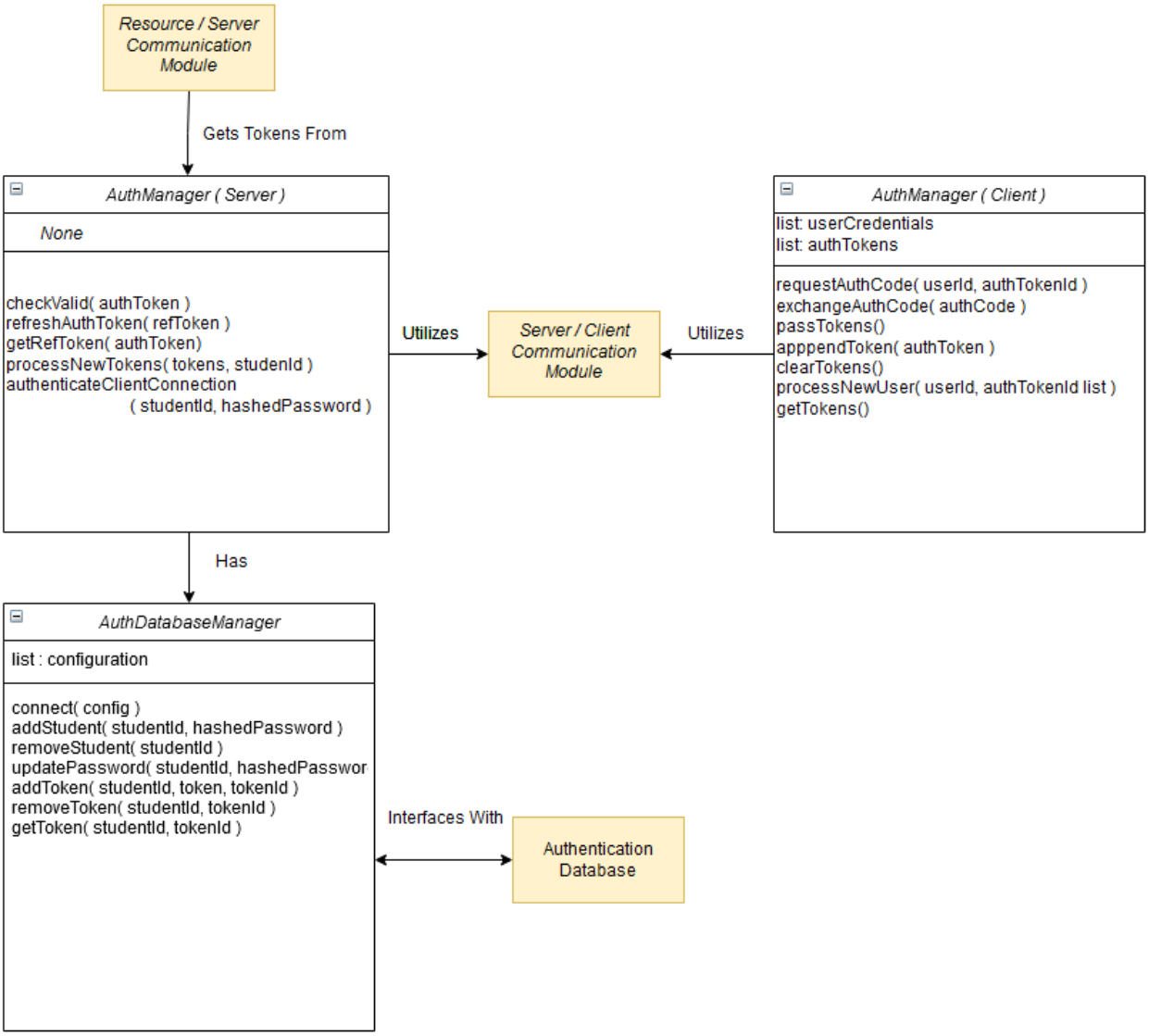


Figure 3.2.7.1: The OAuth2 Manger Module

The OAuth2 Manager handles the process of acquiring and managing authentication tokens. These tokens will allow the server to operate on the behalf of the user even when the user is offline.

Module Components

- **AuthManager (Client)** - The AuthManager on the client side handles the initial acquisition of authorization tokens.
- **AuthManager (Server)** - Authentication and refresh tokens in the Authentication database are manages by this component

- **AuthDatabaseManager** - The AuthDatabaseManager acts as an interface between the server-side AuthManager and the authentication database.

3.2.6.1 Token IDs

Authentication tokens and refresh tokens will be identified by their token ID. The token ID for these tokens will include the following components:

- The service for which the token is used
- The scope of authentication granted by the token
- Whether the token is an authentication token or a refresh token

Token IDs do not contain information associating them to a user. Token association is determined by the relations established in the authentication database.

3.2.6.2 AuthManager (Client)

The client-side AuthManager is responsible for handling the process of acquiring user authentication for the various services that the Digital Backpack interacts with. This is a process that requires internet connection and should only be performed when the app is initially set up and when new services become relevant to a particular student. This AuthManager also communicates with the client-side storage manager for the purpose of retrieving the user's password.

The following is an explanation of the fields and operations relevant to this AuthManager.

- **List: AuthTokens**

  As the AuthManager obtains permission to operate on the behalf of the user for various different services and on various different scopes, Oauth2 authorization tokens are issued to the DigitalBackpack. Said tokens are stored in this list.

- **requestAuthCode(userId, authTokenId)**

  Requesting an authorization code is the first step in the process of obtaining an authorization token. The AuthManager redirects the user to a webpage where they grant the Digital Backpack permission to act on the user's behalf.

- **exchangeAuthCode(authCode)**

The AuthManager contacts the authorization server relevant to a given authorization code. The AuthManager exchanges the authorization code for an authorization token.

- **appendToken(authToken, authTokenId)**

  Helper function appends an authToken and its associated ID to the authTokenList.

- **clearTokens()**

  Helper function clears the authTokenList.

- **processNewUser(userId, authTokenIdList)**

  This function manages the process of taking a new user, their list of needed authorization tokens, and obtaining authorization for each token. These tokens are then stored in the authTokenList.

- **getTokens()**

  This public-facing interface function returns the list of authTokens

3.2.6.2 AuthManager (Server)

The server-side AuthManager is responsible for managing authorization and refresh tokens to enable the server to act autonomously and indefinitely on the behalf of the user. This AuthManager has an AuthDatabaseManager that handles the storage of tokens. This AuthManager is also responsible for serving tokens to the Server / Resource Communicator to authorize external operations.

The following is an explanation of the methods used by this AuthManager.

- **checkValid(authToken)**

  The checkValid function takes in an authentication token and checks if the token has expired. If it has, it refreshes the token and stores the updated token.

- **refreshAuthToken(refToken)**

  The refreshAuthToken function takes in a refresh token and connects to the associated authentication server to retrieve a refreshed authentication token.

- **getRefToken(authToken)**

The getRefToken function transmits a valid authentication token with the authentication server and receives a refresh token in return. The refresh token is then stored in the authentication database.

- **processNewTokens(tokens, studentId)**

  This function handles the processing of new tokens received from a client. Refresh tokens are obtained and all tokens are stored in the authentication database

- **authenticateClientConnection(studentId, hashedPassword)**

  This function takes a student's hashed password and compares it to the hashed password stored in the authentication database.

3.2.6.2 AuthDatabaseManager

The AuthDatabaseManager acts as an interface between the proxy server and the authentication database.

The following is an explanation of the fields and methods used by this AuthManager.

- **list : config**

  The config list contains the constant information necessary to connect to the Authentication database, such as the address of the database and login information.

- **connect(config)**

  The connect function opens a connection to the authentication database.

- **addStudent(studentId, hashedPassword)**

  This function adds a new student to the authentication database.

- **removeStudent(studentId)**

  This function removes a student from the authentication database.

- **updatePassword(studentId, hashedPassword)**

  This function updates a student's password in the authentication database.

- **addToken(studentId, token, tokenId)**

  This function adds a token associated with a student to the authentication database.

- **removeToken(studentId, tokenId)**

  This function removes the specified token from a given student in the authentication database.

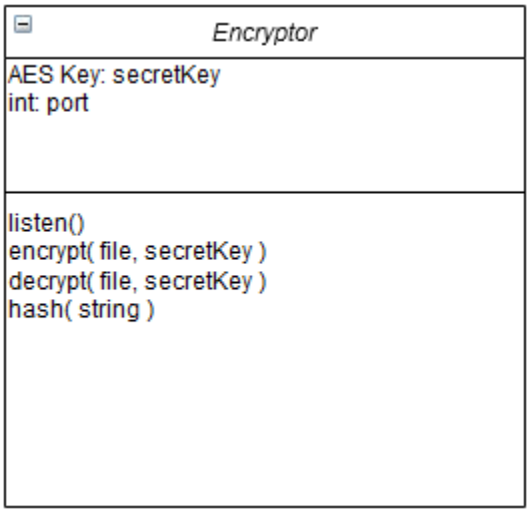
- **getToken(studentId, tokenId)**

  This function retrieves the specified token that is associated with the specified student from the authentication database

3.2.7 Encryption

Any sensitive data stored locally in the Digital Backpack system or transferred over the client/server connection will be encrypted using AES encryption. As such, both the client and the server will have an Encryptor component that will handle both the process of encrypting and decrypting and the safe handling of encryption keys.

3.2.7.1 Encryptor (server)



The proxy server has no hardware to ensure the secure storage of keys and is hosted on third-party hardware. For these reasons, the server-side encryptor does not store it's key.

Instead, the key must be transferred to the server over a secure TCP connection whenever the server is restarted.

The following is an explanation of the fields and methods used by this encryptor.

- **secretKey**

  This is the AES encryption key used to perform encryption operations.

- **port**

  This is the port on which the encryptor listens for it's AES key on startup.

- **listen()**

  This function starts a server that waits for a TCP connection over which to receive it's encryption key.

- **encrypt(file, secretKey)**

  This function encrypts the given file through a standard AES encryption algorithm.

- **decrypt(file, secretKey)**

  This function decrypts the given file through a standard AES decryption algorithm.

- **hash(string)**

  This function hashes the given string using a standard salted hash algorithm.

3.2.7.2 Encryptor (client)

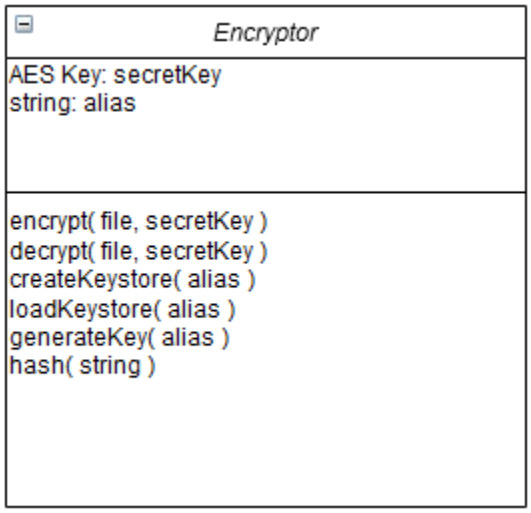


Figure 3.2.7.2.1: Encryptor (Client) UML Diagram

The client-side encryptor utilizes Android's secure storage hardware to safely store the AES encryption key.

The following is an explanation of the fields and methods used by this encryptor.

- **secretKey**

  This is the AES encryption key used to perform encryption operations.

- **alias**

  This alias is a string identifier for the Digital Backpack entry in the Android keystore environment.

- **encrypt(file, secretKey)**

  This function encrypts the given file through a standard AES encryption algorithm.

- **decrypt(file, secretKey)**

  This function decrypts the given file through a standard AES decryption algorithm.

- **createKeystore(alias)**

  This function creates an entry for the Digital Backpack in the Android keystore environment.

- **loadKeystore(alias)**

  This function loads the Digital Backpack's entry from the Android keystore environment and retrieves the secretKey stored there.

- **generateKey(alias)**

  This function uses the Android keystore library to generate an AES encryption key.

- **hash(string)**

  This function hashes the given string using a standard salted hash algorithm.

Team DigiLearn

4.0 Implementation Plan

The implementation of the Digital Backpack has been broken up into four phases. The first two phases are summarized by the following Gannt charts. Figure 4.0 covers phases one and two while Figure 4.1 covers phases three and four.

The first phase of the project was the Software Design and Project Initiation phase. This phase was dedicated to planning the project for the semester and defining the implementation details of the project. The final product of this phase was this document.

The second phase is the Full Prototype Initiation phase, which is the first and primary implementation phase. During this phase, a fully functional prototype of the Digital Backpack will be created. Development of the Digital Backpack will be organized into week-long sprints. These sprints will begin and end on Tuesday. During the Tuesday team meeting, team members are expected to report their progress and address any issues from the previous week. On the following day's Wednesday meeting, tasks will be updated and the next week's sprint will be planned.

The third phase of the project is the Software Refinement and Testing phase. The development process will continue to be organized into week-long sprints in the same format as described above. During the phase, the full prototype of the Digital Backpack is complete and development shifts towards refining the final product through the modification of features and bug-fixing. During this time, the team will conduct experimental and user-based testing to confirm that the project meets the performance requirements outlined in the Requirements Specifications document.

WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
1	Software Design and Project Initiation					
1.1	Communication Strategy Memo	Sebastian	1/11/21	1/20/21	9	100%
1.2	Software Design Document - Intro	Caitlin	1/12/21	2/1/21	19	100%
1.2.1	Software Design Document - Conclusion	Caitlin	1/12/21	2/1/21	19	100%
	Software Design Document - Arch. Overview	Israel	1/12/21	2/1/21	19	50%
1.2.2	Software Design Document - Imp. Overview	Israel	1/12/21	2/1/21	19	90%
1.2.3	SDD Modules - Proxy Server	Caitlin	1/12/21	2/1/21	19	50%
1.2.4	SDD Modules - Security	Sebastian	1/12/21	2/1/21	19	60%
1.2.5	SDD Modules - REST Interface	Israel	1/12/21	2/1/21	19	50%
1.2.6	SDD Modules - Application Interface	Grace	1/12/21	2/1/21	19	50%
1.2.7	SDD Modules - Application Logic	Kristine	1/12/21	2/1/21	19	50%
1.2.8	SDD - Implementation Plan	Sebastian	1/12/21	2/1/21	19	75%
1.2.9	SDD - Rough Draft	Team	3/23/18	3/23/18	0	60%
1.2.9	SDD - Final	Team	3/23/18	3/23/18	0	0%
2	Full Prototype Initiation					
2.1	First Sprint	Team	2/2/21	2/9/21	7	0%
	Peer Evaluation 1	Team	2/11/21	2/12/21	1	0%
2.2	Second Sprint	Team	2/9/21	2/16/21	7	0%
	Register for UGRAD Symposium	Unassigned	2/8/21	2/12/21	7	0%
	Design Review Script Draft	Team	2/15/21	2/19/21	4	0%
	DR2 - Script Final	Israel, Sebastian	2/15/21	2/16/21	1	0%
	DR2 - Slides	Kristine	2/16/21	2/17/21	1	0%
	DR2 - Voiceover	Grace	2/16/21	2/17/21	1	0%
	DR2 - Final Editing	Caitlin	2/17/21	2/19/21	2	0%
2.3	Third Sprint	Team	2/16/21	2/23/21	7	0%
	Forth Sprint	Team	2/23/21	3/2/21	7	0%
	Fifth Sprint	Team	3/2/21	3/9/21	7	0%
	Tech Demo Flight Plan	Sebastian	3/1/21	3/5/21	5	0%
	Schedule Tech Demo with Mentor	Team	3/2/21	3/2/21	1	0%
	Sixth Sprint - Begin Prototype Refinement	Team	3/9/21	3/16/21		
	Peer Evaluation 2	Team	3/11/21	3/12/21	1	0%
	STP - Introduction	Unassigned	3/8/21	3/16/21	8	0%
	STP - Conclusion	Unassigned	3/8/21	3/16/21	8	0%
	STP - Unit Testing	Unassigned	3/8/21	3/16/21	8	0%
	STP - Integration Testing	Unassigned	3/8/21	3/16/21	8	0%

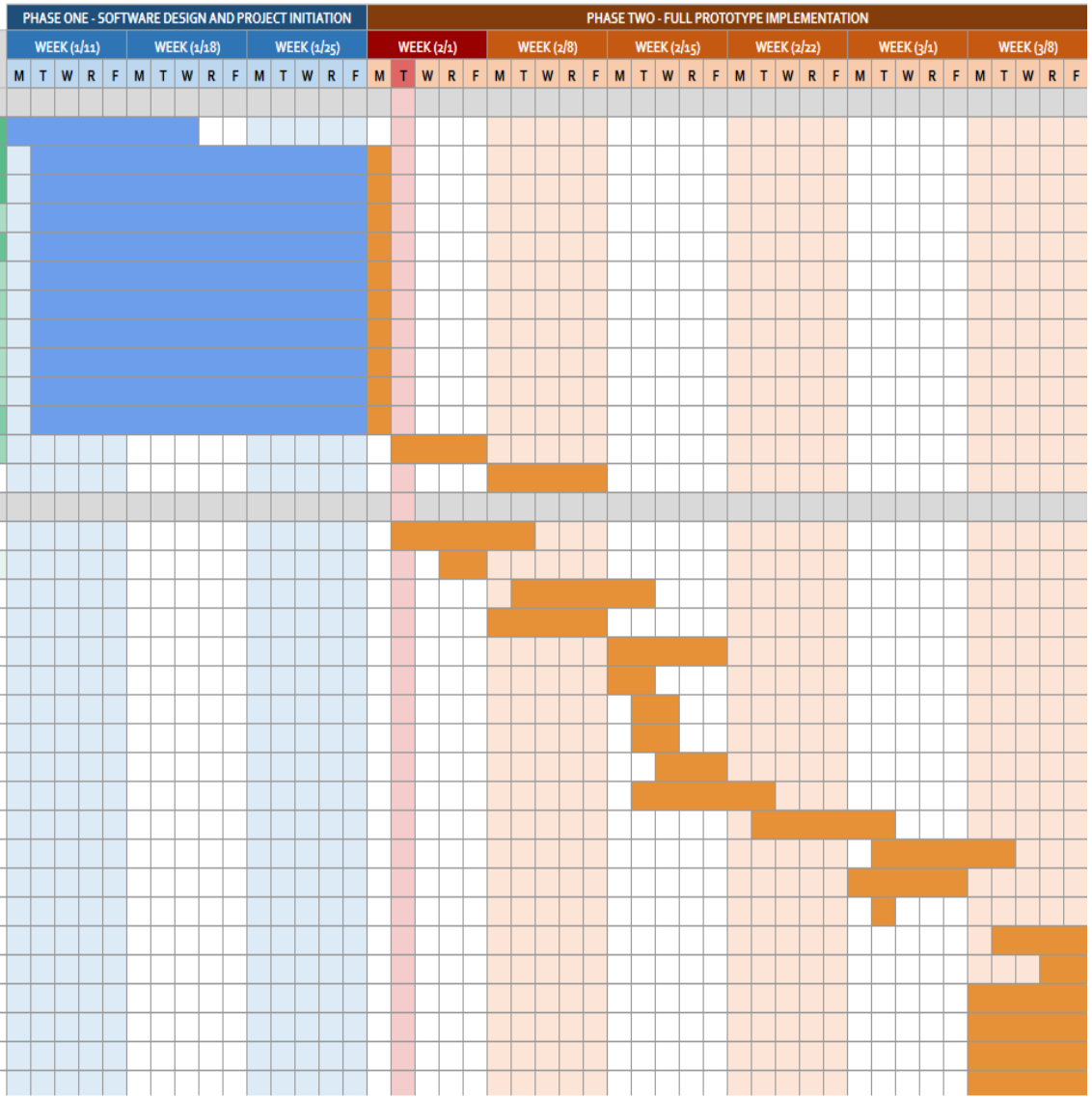


Figure 4.1 Gantt Chart of Phases One and Two

TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	PCT OF TASK COMPLETE
Software Refinement and Testing					
STP - Introduction Final	Unassigned	3/16/21	3/23/21	8	0%
STP - Conclusion Final	Unassigned	3/16/21	3/23/21	8	0%
STP - Unit Testing Final	Unassigned	3/16/21	3/23/21	8	0%
STP - Integration Testing Final	Unassigned	3/16/21	3/23/21	8	0%
STP - Usability Testing Final	Unassigned	3/16/21	3/23/21	8	0%
Software Testing Plan Final	Team	3/23/21	3/26/21	8	0%
Peer Eval 3	Team	3/25/21	3/26/21	1	0%
DR3 Script Draft	Unassigned	3/9/21	3/16/21	7	0%
DR3 Final	Unassigned	3/16/21	3/23/21	7	0%
Seventh Sprint - Continue refinement	Team	3/16/21	3/23/21	7	0%
Eight Sprint - Continue Refinement	Team	3/23/21	3/30/21	7	0%
Ninth Sprint - Begin testing	Team	3/30/21	4/6/21	7	0%
Completed Team Website	Unassigned	3/30/21	4/13/21	14	0%
Capstone Poster	Unassigned	3/30/21	4/13/21	14	0%
Capstone Presentation Planning	Unassigned	3/30/21	4/13/21	14	0%
Schedule Capstone Presentation Dry Run	Team	4/6/21	4/6/21	0	0%
UGRADS Presentation	Team	4/16/21	4/16/21	0	0%
Peer Eval 4	Team	4/15/21	4/16/21	1	0%
Project Completion					
Acceptance Test Demo Plan	Unassigned	4/13/21	4/20/21	7	0%
Team Reflection Document	Team	4/20/21	4/27/21	7	0%
Final as-built report	Unassigned	4/20/21	4/27/21	7	0%
Final Project Delivery	Team	4/20/21	4/29/21	9	0%
Peer Eval 5	Team	4/29/21	4/20/21	1	0%

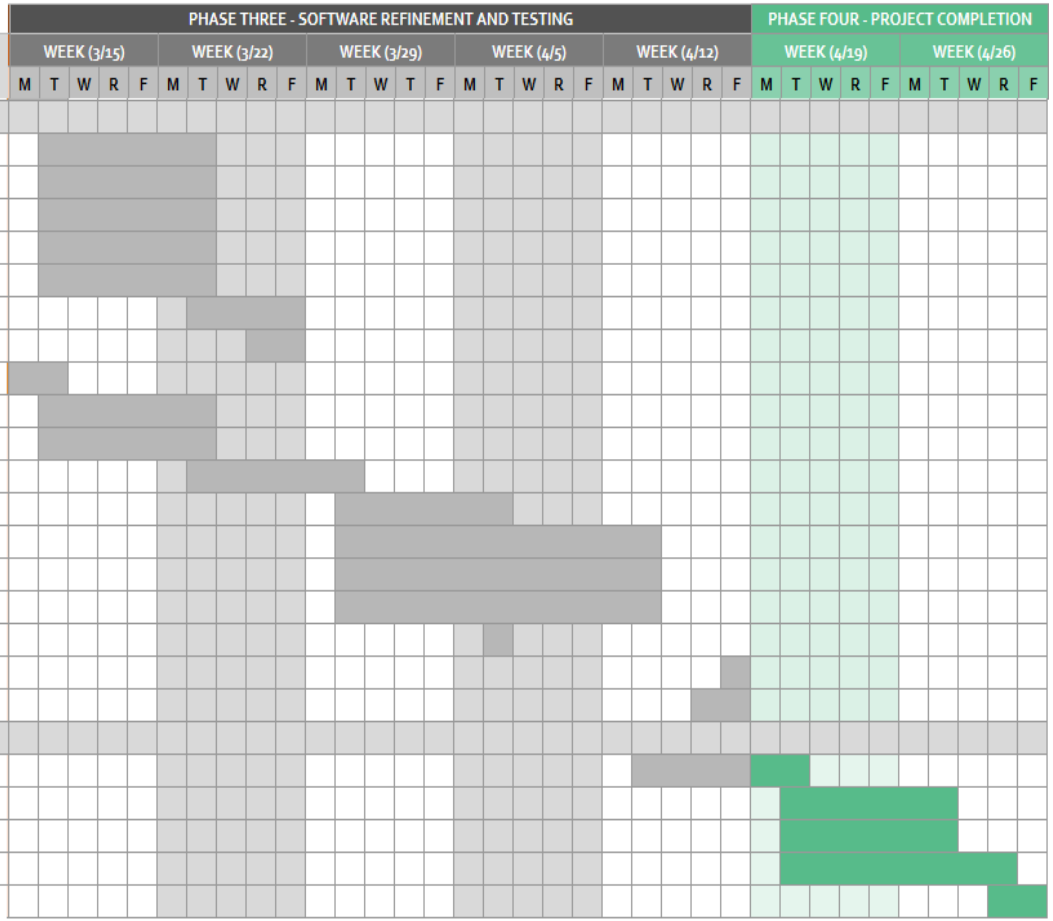


Figure 4.2: Gantt Chart of Phases Three and Four

Team DigiLearn

5.0 Conclusion

As the internet has become more and more of a necessity for students of all levels of education the “digital gap” has grown considerably. Students without regular or reliable internet access are at a significant disadvantage compared to others that are able to access the internet consistently. The Digital Backpack project aims to aid students struggling with this issue by offering an opportunistic Content Delivery Network or oCDN for educational content. The Digital Backpack application will give users the ability to download and upload homework assignments, tests, and even search for resources to support their learning in the background, any time they are able to connect to the internet. By storing these things on the user’s device they will be able to take educational content home and still be able to participate similarly to students with constant internet access while being completely offline.

This document served to define the software design plans of the Digital Backpack. The project overview, in Section 2.0, described the technologies that will be used for the project, as well as a general plan for the architecture of the system. Section 3.0 discussed in details the most important components of the Digital Backpack to go in depth on the requirements for each of these components. Section 3.1 focuses on the front-end of the system, discussing the views of the application as designed for an ideal user experience. Section 3.2 describes the back-end that brings the project to life and allows for a smooth online and offline transition. The implementation plan for the project is expressed through a Gantt chart in Section 4.0.

Enumerating the details of this project provides a clear guideline for the future of the Digital Backpack. The specifications and technologies described in this document were chosen according to their compatibility with the requirements requested by the client. This exploration of compatible technologies has further refined the overall final project. Previous requirements, such as Khan Academy support, have been abandoned due to incompatibility errors that were found during research. These issues have been discussed and acknowledged by the client. Still, the

overall goal of the project has been determined to be attainable. The success of this project will bring about the expansion of educational opportunities for disenfranchised communities.