Team DigiLearn **Final Report:**

The Digital Backpack

short line

Doctor Morgan Vigil-Hayes

Volodymyr Saruta

Caitlin Abuel

Israel Bermudes

Kristine Hermosado

Sebastian Kastrul

29 April, 2021

Team DigiLearn **Table of Contents**

short line

Team DigiLearn **1.0 Introduction**

short line

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 is 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 Google Services such as Google Drive, Google Search, and Google Classroom.

This document serves as a final report for the first beta prototype of The Digital Backpack. Section 2.0 details an overview of the process for development. This includes the definition of team roles as well as tools that were utilized during the process. Section 3.0 summarizes the main requirements outlined for the project by the client. The current architecture and implementation of the Digital Backpack are described in Section 4.0, which goes into detail on specific modules and technologies that were used in the software. A record of the testing plan is discussed in section 5.0, describing the various testing that was done during development. Section 6.0 outlines the overall project timeline, highlighting major milestones that were achieved throughout the year. Future work is discussed in Section 7.0, with ideas for expansions on the Digital Backpack. The document concludes with Section 8.0, followed by a glossary and appendix. Appendix A describes the development environments and toolchains utilized for this project.

Team DigiLearn **2.0 Process Overview**

short line

# 2.1 Development Life Cycle

Development started with gathering requirements from the client. Initially, the major requirements of the project were gleaned from the project description provided by the client. In the subsequent weeks of the first semester, regular meetings were held to clarify specifics of certain requirements or resolve issues the team ran into during research into possible avenues for implementation. Research topics were broken into five major topics: authentication, 3rd party resource compatibility, cloud storage, and support, user interface and experience design, and cross-platform development.

The process of implementation began in the second semester based on the five topics listed above. Atlassian products like JIRA and Confluence were used to keep track of tasks, features, and internal timelines. GitHub was used for version control and development storage. Internal communication was handled largely over Discord text and voice channels, external communications were handled exclusively over email or Zoom as per the client's request.

The team implemented a hybrid of SCRUM and Agile methodologies during the development process. Sprints were built one to two weeks in advance during team meetings through the JIRA board to ensure constant development progress, team accountability, and allow for flexibility where needed. Tasks within the sprints ranged from client outreach and document building to feature implementation and bug fixing.

Programming was conducted on a number of different platforms depending on the needs, use cases, and preference of the team member. Android Studio was the most widely used IDE used internally for the development of the mobile app. Built into Android Studio is git support, a diverse choice of mobile device hardware emulation, and options for testing with all current and previous versions of Android OS making it a very powerful tool. Other JetBrains-based IDE’s such as PyCharm and IntelliJ was used for the development of server-side systems for similar reasons to Android Studio. Systems administration and server testing were conducted through VSCode due to its support of the wide array of libraries and languages needed to create a functional service.

Meetings were held internally twice a week on a regular schedule as well as many impromptu online and in-person meetings when needed. Bi-weekly meetings were used for planning and discussion while the other meetings were mainly for development that necessitated interpersonal communication. Client meetings were held once a week when time permitted to keep up with the “constant input/constant output” ethos of Agile development.

# 2.2 Team Roles

Team member roles were assigned initially based on preference and past experience. Kristine Hermosado had the most experience with UI/UX design and was assigned with research and development of that aspect of the project. Caitlin Abuel had taken multiple classes on networks and remote services and was tasked with managing the Digital Backpack server setup and implementation. Israel Bermudes also had some experience with networks as well as professional experience with RESTful systems which allowed them to fit in well as the team’s 3rd party interface developer. Sebastian Kastrul had used Google Authentication services on a personal project as well as shown leadership skills during a software engineering course which made them the obvious choice as team lead and developer for authentication services.

As development started to pick up speed it was apparent that some portions of development had been overlooked and needed to be done. While the team had a UI, there was nothing to make it work and so Israel took the role of back-end functionality developer and client-side server interfacing. Sebastian took up the task of implementing authentication and storage on the client-side because of his preexisting knowledge of those libraries and flows from building them on the server. Kristine’s initial role was modified to include the development and implementation of the designs created the previous semester. Caitlin also became the lead developer for the PWA after having suggested the idea as a way to reach a larger audience with a singular solution instead of limiting the web-based platform to Google Chrome specifically.

Team DigiLearn **3.0 Requirements**

short line

As discussed in Section 2.1, the initial requirements for the Digital Backpack were derived from the project proposal provided by Dr. Morgan Vigil-Hayes. The Fall 2020 semester was devoted to understanding and refining these requirements. This process began with an investigation of the technological feasibility of the major components of the Digital Backpack. Namely, there were five areas of investigation: authentication and security, 3rd party resource compatibility, cloud storage and support, user interface and experience design, and cross platform development. The initial investigation found that all desired aspects of the Digital Backpack were feasible, so development proceeded towards creating more rigorous and specific requirements specifications.

Instrumental to this phase of the project was ongoing communication with the client. As issues arose and development continued, it was critical that the team and the client remained on the same page. To this end, meetings were held with the client on a weekly basis. During this time, a set of six domain-level requirements were developed to guide the requirements gathering process. These requirements have evolved over time. They are now as follows:

• The Digital Backpack will aid remote education.

• The Digital Backpack will provide seamless transitions between online and offline functionality.

• The Digital Backpack will utilize oCDN functionality in order to facilitate asynchronous connections and provide a delay-tolerant network.

• The Digital Backpack will support two types of end-users: students and teachers.

• The Digital Backpack will be highly accessible in both design and device requirements in order to accommodate a wide range of users.

• The Digital Backpack will securely handle all sensitive personal and scholastic user data to prevent unauthorized access.

These requirements were further investigated through the development of user stories. These stories were refined into use cases. Finally, a practical set of functional and environmental requirements were developed. These requirements are enumerated in their final form below. For more information on the requirements gathering process and the evolution of these requirements, please refer to the Team DigiLearn Requirements Specification document.

# 3.1 Functional Requirements

The functional requirements for the Digital Backpack describe its key features and how it meets the needs of the user. The functional requirements are listed below. Each requirement is followed briefly by more details.

1. The Digital Backpack will implement a proxy server to work on the user’s behalf.

The proxy server manages the user’s Google account and interfaces with Google services while the user is offline. Once a connection is established, the proxy server pushes useful data to the user’s device.

1. The Digital Backpack will display downloaded content for use offline.

All content provided by the proxy server is fully accessible offline; course and assignment information, downloaded files, and search results can all be accessed from inside the DigiPack app.

1. The Digital Backpack will interface with Google Classroom, Google Drive, and Google Search.

The DigiPack app provides an interface to interact with all of the above Google services. It is the proxy server that actually manages these services and then returns the results to the user’s device.

1. The Digital Backpack will require the user to log in or create a new profile.

Since the Digital Backpack works so closely with Google accounts and the Google API, the Digital Backpack handles user accounts and user sessions through the user’s extant Google account. This has a myriad of benefits including a quick sign-up process for the user.

1. The Digital Backpack will manage sensitive data and prevent unauthorized access.

The primary security vulnerability in the Digital Backpack system is the connection between the proxy server and the user’s application. To keep this connection secure, the DigiPack implements a standard, encrypted HTTPS connection. These connections are also validated using OAuth2 tokens, which are discussed under F.R.6.

1. The Digital Backpack will authenticate all user actions.

The Digital Backpack implements OAuth2 tokens to validate all connections to the DigiPack server. Built on top of Google’s own authentication framework, users can be confident that their data is secure.

1. The Digital Backpack will have an upload and download queue.

A primary use case for the Digital Backpack is obtaining files and submitting assignments. To make sure that this functionality is persistent across online/offline use, the DigiPack app implements an upload and download queue to store these actions until they are completed.

1. The Digital Backpack will have a domain filtering system for teachers and parents.

The Digital Backpack offers easy access to Google Search. Since Google search results can produce inappropriate content, the Digital Backpack proxy server can append search operators to student requests to whitelist and blacklist websites and terms. This ensures that students using the Digital Backpack are not server inappropriate content.

1. The Digital Backpack will use a uniform RESTful format for storage & transfer of data.

The Digital Backpack proxy server provides a REST interface to communicate seamlessly with Google APIs. Within the Digital Backpack system, content is stored in a uniform JSON format for consistent communication.

1. The Digital Backpack will be available on Android and for web browsers.

To maximize accessibility, the Digital Backpack has an interface available for Android and for web browsers. For the web, the Digital Backpack uses modern progressive web app technology to enable offline use.

1. The Digital Backpack will have a similar performance on each platform.

The Digital Backpack offers the same functionality for Android and the progressive web app.

1. The Digital Backpack will have multiple easy-to-use interfaces for different ages.

The Digital Backpack targets students in K-12 education. This covers a very wide range of ages. To best serve both young children and young adults with the same application, the Digital Backpack Android application implements two separate user interfaces. One is targeted specifically towards children and the other is targeted towards young adults. Both are built on top of the same application and so offer the exact same functionality.

1. The Digital Backpack will manage downloaded content stored on the user’s device.

The Digital Backpack offers an interface to the filesystem on the user’s device. Through this, users are able to directly access their content.

# 3.2 Environmental Requirements

The Digital Backpack must operate in a unique environment that imposes some unique constraints on how it can and can not function. The following environmental requirements shaped the development of the Digital Backpack.

**3.2.1 Federal Laws**

As the DigiPack is a mobile application for K-12 students, the constraints are laws that protect children privacy such as:

* COPPA (Children's Online Privacy Protection Act), which protects children's privacy under the age of 13. The act manages how the application would collect and store personal information about the user. Since the Digital Backpack would require account logins, it is essential that the system is aware of what is taking from the users and does not violate the law.
* FERPA (Family Educational Rights and Privacy Act), which gives parents the right to have access and revise their children's (who's under the age of 18) education records and have authority over what can be disclosed about the student regarding their education record. Since the Digital Backpack would be accessing the student's grade (which is under the protection of FERPA because it is directly related to the student, which is also maintained by an institution), the system must comply with the rules set by FERPA.

**3.2.2 Robust Network Usage**

Due to the nature of opportunistic connections to serve users with poor internet connections, it is likely and unavoidable that the user’s connections to the proxy server will be brief, sparse, and sporadic. For this reason, these connections must be used to the fullest possible extent:

* Transmission must begin as soon as the connection is established so that no time goes unused.
* Any possible work related to the transmission of data, such as queue said data or otherwise preparing it for transfer, should be handled while the user is offline.
* In the case that file transfer is interrupted, the best effort must be made to ensure that said transmission can be resumed, instead of restarted when the connection is reestablished.

Team DigiLearn **4.0 Architecture and Implementation** 

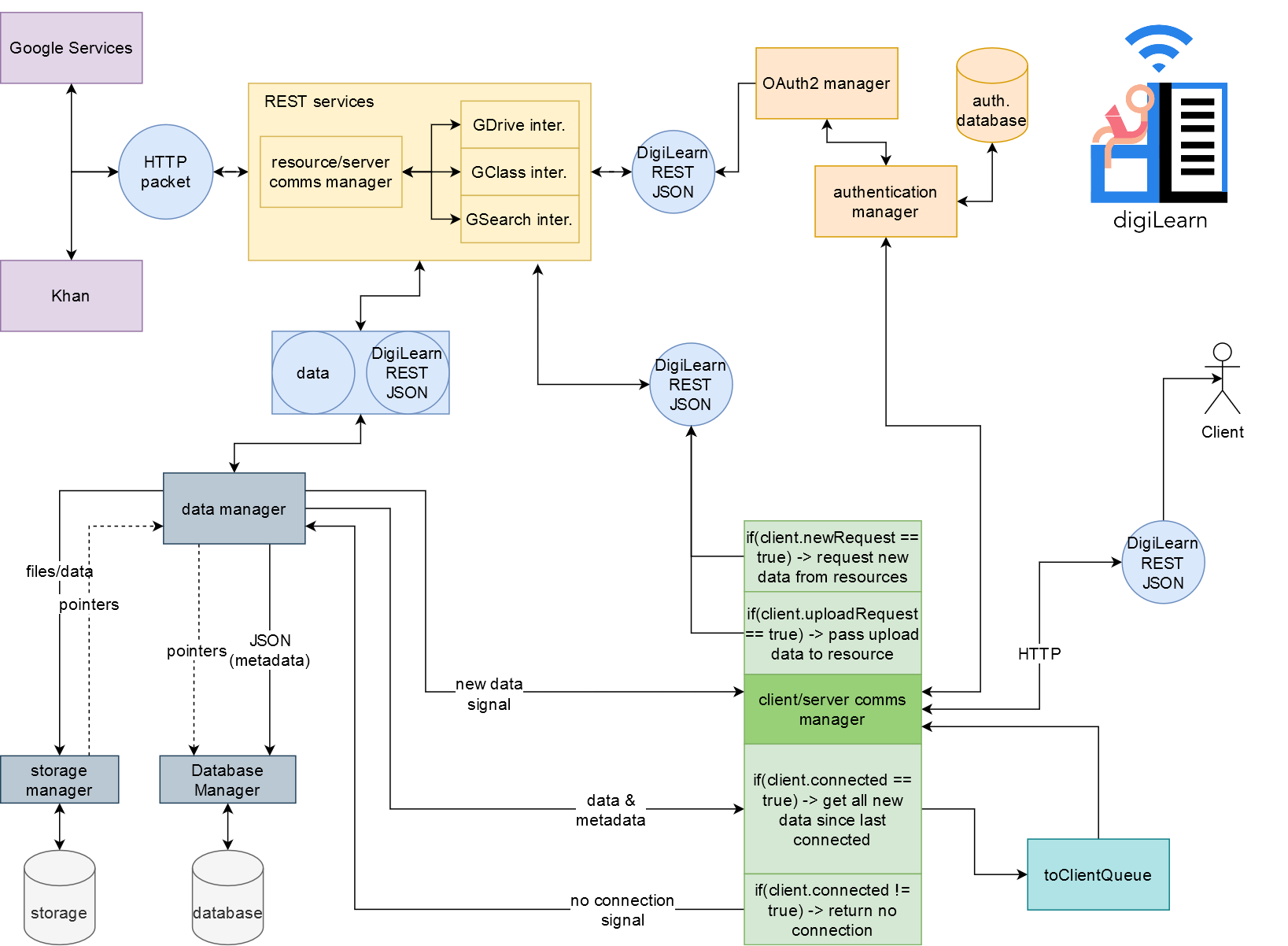
short line

# 4.1 Architecture Overview

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, Google Search, 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 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 its 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.

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. 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 4.1: 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). A user 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, 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 user’s 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.

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 are missing from the transmission. Users are able to make search queries, 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. Any new requests or uploads are sent to their respective REST interpreters to be passed to the requested resource and the process starts again.

# 4.2 Implementation of Major Components

This section details the implementation of major components in the Digital Backpack as they exist in the beta prototype.

**4.2.1 REST**

The REST interpreters were written in Python using Google’s Python Libraries. An interpreter is built for each resource and are used to convert the data passed back from requests into “DigiJson”. DigiJson at its core is simply a way for the Digital Backpack server to filter the data passed back from each resource to strictly what is needed by the server or user to function. This is done for two reasons; the first is to ensure that if values are missing from the response from a resource it is handled accordingly and second, to minimize the amount of data that needs to be passed between the user and server for any given operation.

* **Google Search** - The Google Search interpreter uses a “Custom Search Engine” provided by Google to serve requests. An array of string queries are passed into it along with any modifiers (currently specified by the view, see section 4.2.2 Retrieve Queries) builds a new string with all the modifiers and then makes the request. It passes back the maximum number of results that Google allows. The Custom Search Engine is currently limited to 100 queries per day.
* **Google Drive** - The Google Drive interpreter allows for the collection of any data available in the users Drive. Files, folders, other drives, etc.; for most cases, simply a user’s credentials and a pointer to a specific file need to be passed to each function to retrieve metadata about the file in question. A list of all files can be pulled using either the get\_file\_list() or get\_drive\_list() functions and passing in a user’s credentials.
* **Google Classroom** - The Google Search interpreter provides all of the basic functionality for Google Classroom that a user can do. Viewing courses and associated assignments, viewing announcements, and submitting assignments.

**4.2.2 Proxy Server**

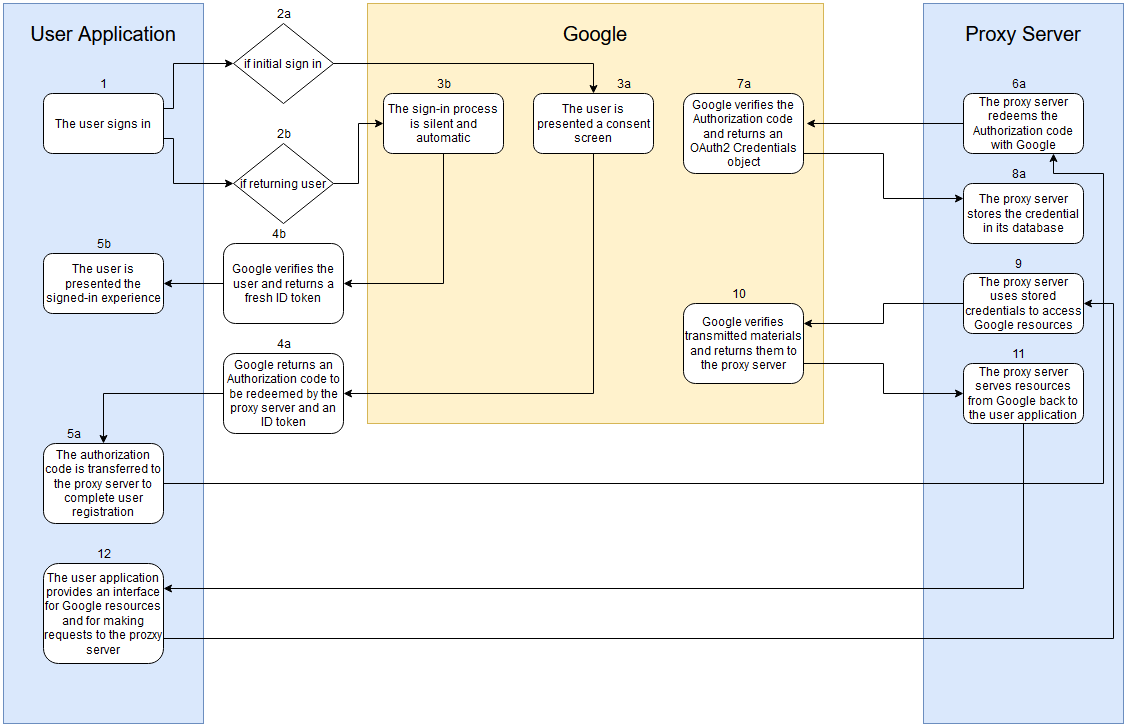
The proxy server is written using the Django Web framework and is hosted on a DigitalOcean Droplet. The server is secured with HTTPS using a certificate from the service LetsEncrypt. Communication between the clients and server is done using a series of GET and POST requests, that are handled using Django’s built-in views and URL routing. The specific backend operations that are performed are described in more detail in sections 4.2.1 (REST) and section 4.2.3 (Security). Listed below are the Django views that are used for server/client communication.

* **MobileAuth/WebAuth** - These views handle a POST request which sends authentication data to the server. The mobile application and progressive web app handle authentication slightly differently. Both utilize the AuthenticationManager, further discussed in section 4.2.3.
* **InitDrive** - The InitDrive view handles a GET request for a user’s Google Drive files. A user token is needed to pass to the GDriveInterpreter, which retrieves a user’s filelist. This view is used for both the mobile application and progressive web app.
* **InitGClass** - The InitGClass view handles a GET request for a user’s Google Class data. A user token is needed to pass to the GClassInterpreter, which retrieves a user’s class and assignment information. This view is used for both the mobile application and progressive web app.
* **DriveServerDownload** - The DriveServerDownload view downloads and stores a user requested Google Drive file onto the server. This file can then be served to the user at a later time. This view is used for the mobile application
* **DriveClientDownload** - The DriveClientDownload view downloads a user requested Google Drive file to the user’s device. This view is used for the mobile application
* **DrivePWADownload** - The DrivePWADownload view has the same functionalities as the DriveServerDownload and the DriveClientDownload, combined into one view for the progressive web app.
* **FileUpload** - The FileUpload view handles a POST request from the user of a file that they wish to upload.
* **RetrieveQueries** - The RetrieveQueries view handles a GET request for Google Search requests.

**4.2.3 Security**

The most significant security concern for the Digital Backpack is user authentication. To handle this, the Digital Backpack has users sign in with Google. Using an authorization code flow, the proxy server is able to act on the behalf of users while they are offline. Authentication flows are illustrated in Figure 4.2.3. Below, a few key details are explained.

* When students sign in to the app (position 1 in the figure) they either begin the initial sign-in flow (beginning with 2a) or the returning user flow (beginning with 2b).
* As soon as a student is registered, the proxy server begins to use student credentials to access Google resources (position 9).
* Positions 9-12 on the figure form a loop that illustrates the cycle of the proxy server distributing files, the user making requests, and the proxy server responding with the result.

****

*Figure 4.2.3: Diagram of Authentication Flows*

**4.2.4 Android Application**

As mentioned before, the Digital Backpack is available on mobile devices and it is only offered for Android devices. The application is built using the Android Studio software as it was the main software commonly used for Android development. The functions and classes described below are the functionality used to handle the functional requirements needed:

* **CacheUtility class** - The class stores and retrieves data from external app storage for caching app data.
* **networkDetectorTool class** - The class detects the user’s SDK build version for Android 9 and above or Android 8 and below to specify what function to use as each version can vary. After checking the SDK build version, it checks for types of connection (we specifically check for Wifi and Cellular) and should check if the connection is lost or available again.
* **popUp class** - Manages how the pop displays should when the user trigger the pop-up
* **signIn** - It starts the sign-in process of the user and passes the user result to another activity that will handle the user’s data.
* **onResume** - Part of the network detector tool, it detects if that specific page activity is on resume and if so it will check if there’s a connection.
* **onStop** - Part of the network detector tool, it detects if that specific page activity is on stop and if so it will check if there’s a connection.
* **serverAuth** - Handles initial sign-in authentication with the server.
* **connectToServer** - Retrieves and caches initial data from the server.
* **getSearchList** - Helper function for connectToServer handles acquisition of GClass data.
* **getClassList** - Helper function for connectToServer handles acquisition of GClass data.
* **getFileList** - Helper function for connectToServer handles the acquisition of GDrive data.
* buildActivitiesFromCache - If the network connection is off, it will try to retrieve the GClass, GDrive, and GSearch data from the cache, and uses that data to build intents for FileListViewActivity and gClassActivity.
* **getFile** - It uses the file ID assigned by Google to request that the server downloads the file.
* **downloadFile** - Once an acknowledgment is received by getFile, a second request is made to download the file off of the server.
* **DigiJson Classes** -
  + **DigiClass** - a representation of the DigiJson for a class built on the server.
  + **DigiDrive** - a representation of the DigiJson for a file list built on the server.
  + **DigiSearch** - a representation of the DigiJson for search results built on the server.
  + **DigiUser** - a representation of the users Google credentials to be passed to the server.
  + **GUserJson** - a representation of the users Google credentials to be used on the mobile app.

**4.2.5 Progressive Web Application**

The Digital Backpack is also available as a progressive web application (PWA). A progressive web app is a browser-accessible application that can also be used offline. This functionality is available on most modern Internet browsers, including Google Chrome and Firefox. Progressive web apps utilize service workers written in javascript in order to carry out offline functionalities.

The progressive web app is organized within the Django project files. The Django project has an assets folder and a templates folder. The assets folder contains a manifest.json file, as well as an icons and styles folder, containing static png logos and CSS files respectively. The templates folder contains the main HTML and Javascript files for the progressive web app.

The main contents of the progressive web app are found in the templates folder of Django. This is where the index file of the webpage is. The javascript functions utilized in the progressive web app are enumerated below:

* **addFiles** - Adds files retrieved from Google Drive to the database and creates HTML elements to display the file list.
* **newSearch** - Takes in user input from the search bar and stores the search in the local database. If the user is offline, the pending queries will stay in the database until they are able to be served.
* **getSearchResults** - Checks for an Internet connection and sends a POST request to retrieve search results from pending queries.
* **addResults -** Adds results from Google Search to database
* **displayResults** - Displays search results associated with a particular Google search.
* **getAssignments -** Gets Google Classroom data for the users.
* **displayAssignments -** Dynamically creates a table to display Google Classroom assignments. There are two minor helper functions for displaying assignments.
  + **getDueDate -** Unpacks a date JSON and returns a string with an American formatted date.
  + **showDetails -** Pulls up expanded details for a selected assignment.

# 4.3 As-Planned vs As-Built

Many changes have been made throughout the course of development for the Digital Backpack. This section highlights the major design changes for the Digital Backpack as well as the reasons behind these changes.

**4.3.1 Changes to Supported Services**

The original vision for the Digital Backpack would support Google Classroom, Google Search, Khan Academy, and YouTube. In an implementation, the Digital Backpack actually supports Google Classroom, Google Search, and Google Drive. Khan Academy and YouTube had to be dropped from the DigiPack.

* **Khan Academy -** Khan Academy previously supported a publicly available API for accessing the content hosted on their website. Unfortunately, this public API was discontinued in early 2020. Due to this, supporting Khan Academy with the Digital Backpack was not feasible.
* **YouTube -** YouTube does not provide a public API for accessing YouTube content. In fact, it is a violation of the YouTube terms of service to download and redistribute YouTube videos. It was not legal for the Digital Backpack to interface with YouTube in the desired way.

Google Drive was selected as an additional service to support in place of the above-removed services. Google Drive was selected for the following reasons:

* Google Drive and Google Classroom are closely coupled; the Digipack would better support Google Classroom with Google Drive support.
* Google Drive is a valuable resource for students. Viewing and interacting with documents is a common use of computers during the school year.

**4.3.2 Changes to Cross-Platform Development**

The Digital Backpack was originally envisioned as an application for Android and iOS. The team had initially planned to use Flutter as a cross-platform development framework to accomplish this goal. Unfortunately, a lack of access to Apple products made developing and testing a successful iOS application unfeasible for Team DigiLearn. Instead, the team pivoted to develop a progressive web application version of the Digital Backpack instead.

During this pivot, the team also decided to pivot away from using Flutter is a cross-platform development tool. This decision was made for the following reasons:

* Flutter only has beta support for web applications and was proving to be more of a hassle than a time-saving solution.
* Since the majority of backend logic for the DIgital Backpack takes place on the proxy server, independently developing two different application interfaces proved to be a feasible solution for cross-platform development.

Team DigiLearn **5.0 Testing**

short line

# 5.1 Unit Testing

Unit Testing is an important and necessary part of the software design process. By breaking down the source code into individual units, tests on these units can help to determine whether or not they are fit for their intended purpose. Unit testing was performed on parts of the Digital Backpack that were integral to the functionality of the entire system. This section defines five major components of the Digital Backpack and breaks each of these components into smaller user tests.

**5.1.1 Google Authentication**

The Google Authentication process is essential for two reasons. First, if authentication fails then the proxy server is unable to serve data to users and the entire system is useless. Second, robust authentication is necessary to keep the entire DigiPack system secure. The components of Google Authentication that were tested and the parameters for said tests are described below.

1. User Sign-in
   1. User sign-in was tested to ensure that only valid credentials were accepted and that users were given the correct flow after a sign-in attempt.
   2. *Equivalence Partitions:* Valid credentials, invalid credentials.
2. Server Side Registration
   1. After a successful sign-in on the client-facing application, an authorization code will be sent to the server. In this case, the authorization manager will redeem the code with Google’s Authentication servers. The returned credentials will be used to register the user in the server’s database. All of this is handled by the server’s authentication manager written in Python
3. Authenticating Requests
   1. Requests from the client-facing application will be paired with the user’s Google ID token and client secret. The authentication manager is responsible for verifying that these credentials are valid and thus authenticating access to a user’s data.
   2. *Equivalence Partitions::* Valid Google ID token and user’s client secret, Invalid Google ID token and user’s client secret
4. Retrieve & Store Credentials
   1. The user's login credentials must be retrieved and stored to access the user's files on services such as Google Drive or Google Classroom. This functionality is handled by the Authentication Manager server-side in Python.

**5.1.2 Network Connectivity**

The Digital Backpack is expected to operate with unreliable connectivity. An indicator for network connectivity will be used to signify whether or not the user has a connection. This is handled on the client-side in Kotlin.

**Handling Network Changes**

The network indicator must demonstrate the connectivity status as online or offline and be able to update this status in real-time.

* *Equivalence Partitions:* Some network connections available, no network connection available.

**5.1.3 Database**

The Django server includes an integrated database used to store credentials and other data associated with a user. The database comprises the database models themselves, implemented in Django's model's framework, and the database manager implemented in Python, which abstracts database access for the rest of the server.

**Database Models**

Database models consist of a series of fields that establish entities that can be stored in the database and the relationships between said entities. For each field in a given model, it must be ensured that the desired data type can be stored in the said field, and invalid data types are rejected with an appropriate error message.

* *Equivalence Partitions:* Valid data types, Invalid Data types
* *Boundary Values:* Valid data types of an incorrect size or that otherwise violate the field’s constraints.
* *Expected Results:* In the case of valid data, the data can be written and read from a given field. In the case of invalid or boundary data, reading and writing are impossible and an appropriate error code is returned.

**Database Manager**

The database manager abstracts the process of querying the database for the rest of the proxy server. Requests may be made to read or write data from any models or fields in the database.

* *Equivalence Partitions:* Valid requests, Invalid Requests
* *Expected Results:* In the case of valid request issues, a read operation's results are returned, or given data is written to the database, and a success code is returned. In the case of invalid requests, such as when data to be written already exists in the database or a search query returns no results, a failure code is returned, and the database is unchanged.

**5.1.4 REST Interfaces**

Modular interfaces within the Digital Backpack architecture allow for future scalability and minimal coupling. Google’s libraries offer a wide array of interactions with user accounts and each interface should reflect the possible actions the Digital Backpack application can do within an account. These tests assume constant uptime and internet connection for the Digital Backpack server.

**Google Drive Interface**

Many Google products use a user's Google Drive to store documents. The Google Drive interface must be able to generate a list of documents and files within a user’s account and return it to the server.

* *Equivalence Partitions:* All files in an individual “drive” (many users have ‘shared’ drives that can be accessed and modified by multiple users)
* *Expected Results:* A list of files and their associated metadata within the specified drive.

The interface must also be able to pull the metadata for an individual file given the file’s Google assigned id token.

* *Equivalence Partitions:* Singular files, folders
* *Expected Results*: The file name, size, parents (the file structure in which the file resides), mime type, id token, and a boolean value representing if the file has been moved to the “trash” folder.

**Google Classroom Interface**

Google Classroom depends heavily on Google Drive for storage capabilities. Student assignments are stored in a shared drive for users to access while things like due dates, what students are associated with what class and announcements for specific classes are created and stored within Google Classroom. A user can be in multiple classes at the same time.

* *Equivalence Partitions:* A singular user’s account, different users account
* *Expected Results:* All classes (specifically the classroom IDs) associated with the user

The Google Classroom interface must be able to pull metadata about a specific class including current and future assignments, due dates of said assignments, the locations of associated documents, and announcements made by the instructor.

* *Equivalence Partitions:* A singular class
* *Expected Results:* A list of all assignments associated with the class and metadata including due dates, associated document IDs, and announcements

**Google Search Interface**

The Google Search API allows the Digital Backpack to serve search results to users. Students can submit requests to the Google Search Interface and, when network conditions permit, will have results delivered back to them.

* *Equivalence Partitions:* A Google Search request
* *Expected Results:* A list of results including links, titles, and summative text.

# 5.2 Integration Testing

Integration testing ensures the efficacy of systems within a project. It is essential to ensure that significant modules work together as expected. The scope of the Digital Backpack project encapsulates many components that rely on each other. The most notable of these interactions include the client/server communication and the REST interfaces & authentication. These modules were tested together throughout the development process. The subsections below detail the relationships between the modules.

**5.2.1 Client/Server Communication**

Communication between the client and server involves many distinct modules. The mobile application is developed using Android Studio and written in Kotlin. The server is hosted on a digital ocean droplet, using the Django framework written in Python. The client and server must be able to send messages to each other in order to perform the necessary tasks for the Digital Backpack, such as requesting and retrieving files. This communication is handled using standard HTTP requests. The main flows for communication include authenticating, initializing user data, and downloading files.

**5.2.2 Resource/Server Communication**

While the Google Interpreters handle the actual communication between the current resources and the Digital Backpack server, storing and accessing files on the server is handled by the Storage Manager. The Storage Manager serves as an interface for both the interpreters and server to access storage within the server. The interpreters interact with it by passing downloaded File objects along with the associated user and metadata to be stored in a local storage location. The server uses the Storage Manager to pass files from to the associated user’s device as well as storing new files from the user to then pass to an interpreter to be uploaded to a resource.

**5.2.3 REST Interpreters & Authentication**

The REST Interpreters rely heavily on the authentication manager, as the relevant tokens are necessary for accessing a user’s authenticated Google profiles. The primary interface between the authentication manager and the REST interpreter is the retrieval of user credentials, which are stored in an OAuth2Credentials object. The REST interpreter passes the user’s ID token to the authentication manager which then returns either a failure code or the needed credentials to act on the user’s behalf. In the case of a failure code, the REST interpreter must take no action and the request in question must not be serviced.

The REST interpreters also interface with the server/client communication module. The SCC passes requests from users, along with their ID token, to the REST interface. In the case that both the ID token and the request are valid, the REST interpreter replies with the data associated with the request, such as a file from the user’s Drive. If a connection cannot be established with the Client at that time, the data is instead deferred to the storage manager and the response is added to the server’s queue. In the case that either the ID token or the request is invalid or malformed, the REST interpreter returns a failure code instead.

# 5.3 Usability Testing

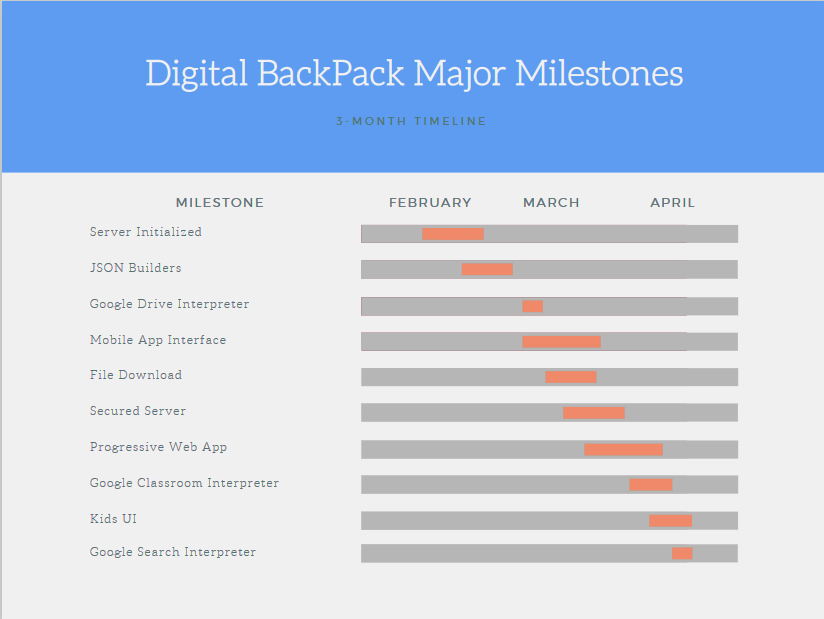
Usability testing is an important part of the development process. The Digital Backpack has an intended audience of K-12 students. By allowing users to test the application, they can provide invaluable feedback that can allow for a better user experience. While the application did not reach a stage where thorough user testing was possible, our client was able to provide us with feedback during the team’s weekly meetings that were informed by user studies done by our client. The front-end interface is arguably one of the most important factors that influence user experience. Our client informed us of design decisions that would be useful for the target audience according to the research done. Some such details included ensuring accessible design with clear and readable text as well as high contrast.

Team DigiLearn **6.0 Project Timeline**

short line

This section goes over the project timeline of the Digital Backpack, including major milestones that were reached. The implementation of the Digital Backpack was broken into two distinct phases. Phase one was the software design and project Initiation phase. This involved a planning phase in which research was done on the necessary modules and libraries that would be used. The second phase was the implementation and prototyping phase. The Gantt chart below highlights the Major milestones that were accomplished during this phase.

The first major milestone was on February 25th when the proxy server was initialized and hosted on the digital ocean. The next big milestone was on March 4th when the google drive interpreter was completed. This served as the basis for the rest of the interpreters needed for this project. The development of the mobile app began on March 5th and evolved throughout the development process. Another milestone was establishing HTTPS on the proxy server, which was accomplished on March 22. From there, the development of the progressive web app began, and its functionalities were caught up to the mobile app by March 30th The google classroom interpreter was completed on April 1st. The Google Search Interpreter and Kids UI were both completed during the same sprint on April 12th.

****

*Figure 6.0: Significant Milestones Gantt Chart*

Team DigiLearn **7.0 Future Work**

short line

The necessary functionalities of the Digital Backpack have been implemented, including a seamless transition between online and offline use, availability on multiple platforms, user authentication, and integration with Google services. Moving forward there are a few features that could be implemented to improve the digital backpack, especially pertaining to the user experience.

Currently, there are no indicators for when the network is downloading new content for the users. Implementation of some sort of loading bar can enhance the user’s experience, making it clearer when they should be expecting new content. The google class assignments list displays assignments in the order they are retrieved from the google classroom API. One thing that may be helpful for users is the ability to choose how they want the assignments list sorted, such as by due date, point value, or class.

Other improvements to the Digital Backpack include the expansion of services that the application interfaces with. Currently, the Digital Backpack does not serve cached webpages when utilizing the offline Google Search. One useful resource could be to serve cached Wikipedia pages to the user.

For production, as more users test the application, some server upgrades may be necessary to accommodate increased traffic. Additionally, applications that use Google APIs have some limitations while they are in testing. For example, the API that allows the Digital Backpack to interface with Google Search limits the number of queries that can be made from a non-enterprise account.

Team DigiLearn **8.0 Conclusion**

short line

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 provides a complete overview of the Digital Backpack project as built by Team DigiLearn. The process, requirements, architecture, implementation, testing plan, schedule, and future work of this project have been enumerated in the above sections. The team has worked incredibly hard to bring to fruition this application that has the potential to positively impact many students’ lives. Though there were many challenges throughout the development process, this project has provided the team with an invaluable learning experience.

Team DigiLearn **Glossary**

short line

This Glossary defines common terms found throughout this document.

**Cross-platform development:** Development of a single application that can function on different operating systems.

**API:** Application programming interface.

**IDE:** Integrated Development Environment

**UI/UX:** User Interface/ User experience

**PWA:** Progressive Web Application

**oCDN:** Opportunistic Content delivery network

**HTTPS:** Hypertext Transfer Protocol Secure

**URL:** Uniform Resource Locator

**SCC:** Server/Client Communication

**JSON**: JavaScript Object Notation

Team DigiLearn **Appendix A:** Development Environment and Toolchain

short line

1. **Hardware**

The platforms the team developed are Linux and Windows. Figure 7 below lists the computer hardware the team used.

| Operating Systems | Processor | Memory |
| --- | --- | --- |
| Linux | i7 | 12GB |
| Windows | i5 | 12GB |
| Linux | 3800xt | 32GB |

Figure 7: System Environment

There should not be any minimum hardware requirements for effective development aside from having a decent hardware machine.

1. **Toolchain**

The development environment was in Python and Kotlin. The team used Visual Studio Code as the IDE for Python and Android Studio for Kotlin. The list below outlines the IDEs and their additional plug-ins:

* Visual Studio Code
  + Purpose: It helped the development of the server.
  + Plug-ins: None
* Android Studio
  + Purpose: The main application that was used for the front and backend of the mobile application.
  + Plug-ins: Kotlin and Git
* Snowflake SSH Client:
  + Purpose: This client was used to SSH into the Django droplet and easily transfer between server and local files.
  + Plug-ins: None

1. **Setup**

The team’s work is mainly around Android Studio and Visual Studio Code. Since the mobile application is Android-based, Android Studio is the main software used for development. It can be downloaded here, <https://developer.android.com/studio>. Visual Studio Code is the editor used when developing the server functionality and as well as developing the progressive web application. It can be downloaded here, <https://code.visualstudio.com/>.

1. **Production Cycle**

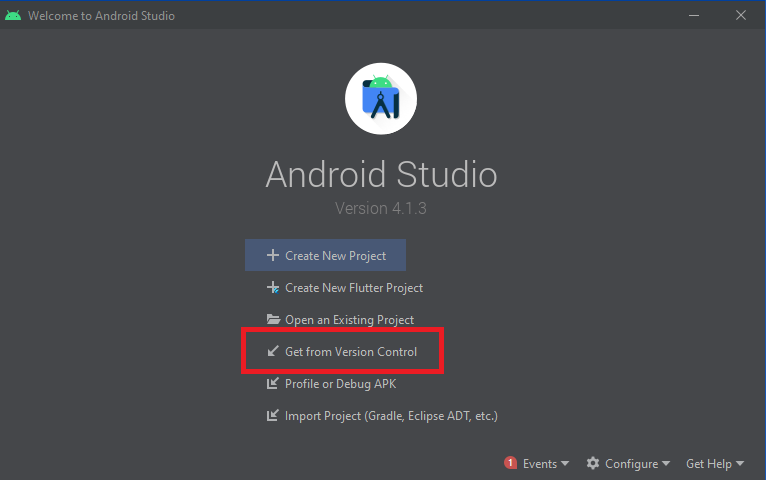
This section will walk through the process of editing, compiling, and deploying the mobile application and the web app. The source code is located here to install/setup the system: <https://github.com/SMKastrul/digiLearn>

Production cycle of the mobile application:

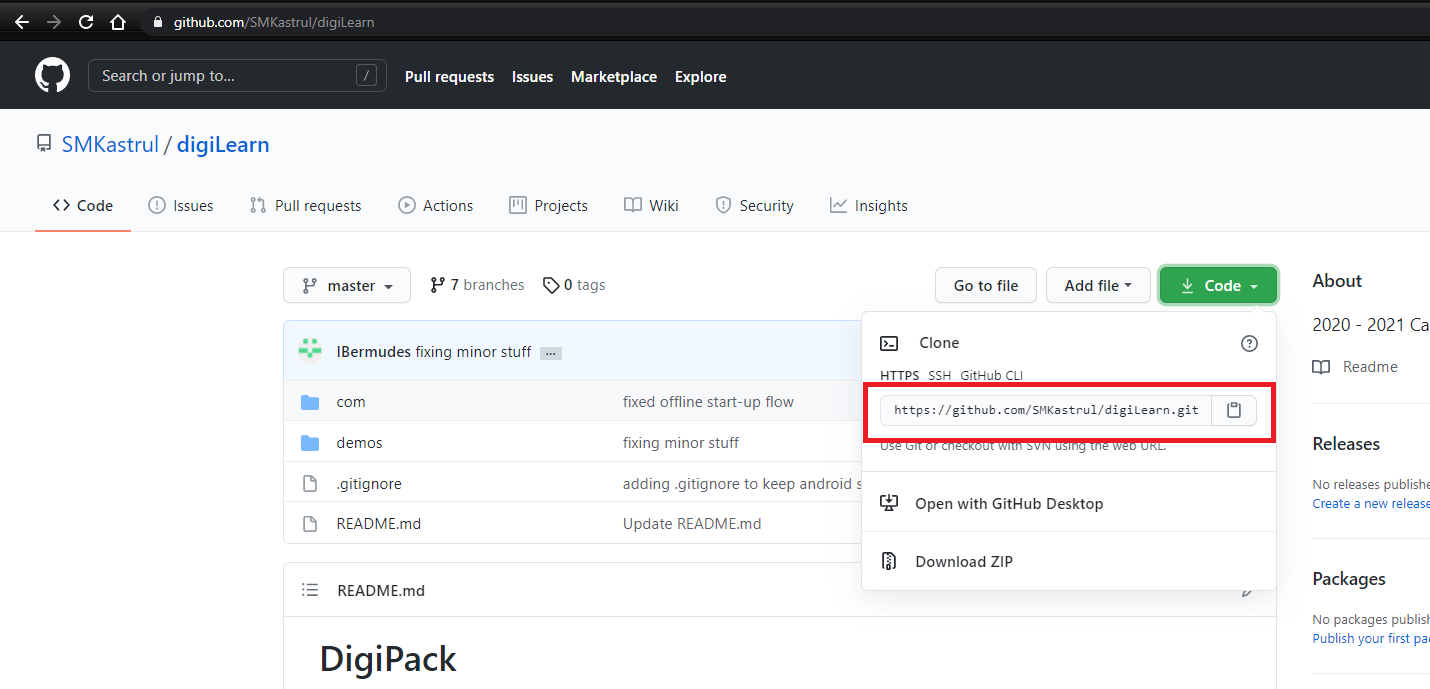
Step 1: Install Android Studio

Step 2: Install the Git plugin in Android Studio

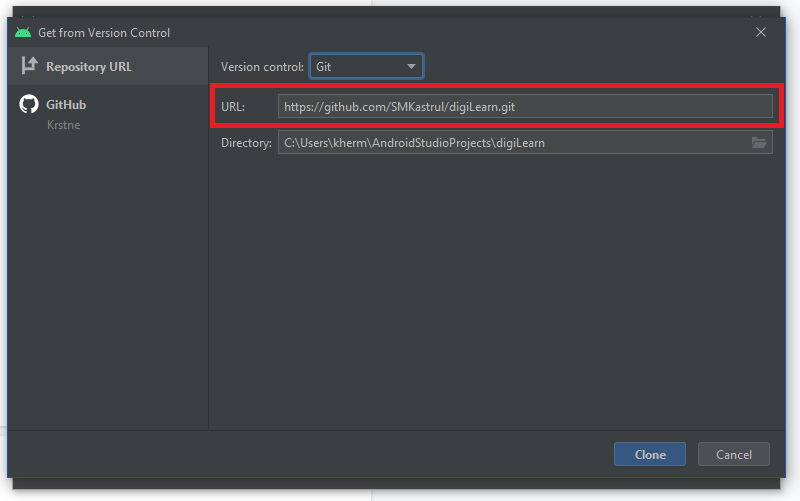
Step 3: Get the project from GitHub to your Android Studio. When first opening the Android Studio, pick “Get from Version Control”



Step 4: There is two option for obtaining the source code either via Repository URL or signing in with Github. If you already signed in with Github, only contributors can have direct access to the repository. For the Repository URL, obtain the source code by grabbing the HTTPS link.



After obtaining the URL, paste it on the Android Studio and its directory should be in the AndroidStudioProject older that was automatically generated when Android Studio was installed. After cloning the repository, you should be set.



Step 5: To edit the application, open Android Studio and open the project folder, navigate to digiLearn/com/DigiLearn/DigitalBackpack/App/Mobile App/DigiPack and set that as the project.

Step 6: Run the gradle build to ensure that all dependencies are installed.

Step 7: Before running the application, ensure that the server is on. As well as creating OAuth credentials in the Google Console API.

Step 8: After making some edits, you can run the program by either running Android Studio’s emulator (the recommended emulator is Google Pixel 2) or by connecting to a physical device via USB.

Step 9: To push new changes to the repo, open the git tab in Android Studio and click “commit” and then “push”.

The progressive web application is integrated within the Django server project files. The process for development on the progressive web application begins with starting the Django server:

Step 1: SSH into the DigitalOcean droplet session

Step 2: Change directories into the folder with the server files.

**$ cd [path to server files]**

Step 3: Start the virtual environment

**$ source [name of virtualenv]/bin/activate**

Step 4: Run the start server command

**$ python3 manage.py runsslserver 0.0.0.0:8000**

Changes to the progressive webapp are made directly on the droplet files. Snowflake SSH client is used to quickly transfer between the local and server files. The progressive web app is organized within the Django project files. The Django project has an assets folder and a templates folder. The assets folder contains a manifest.json file, as well as an icons and styles folder, containing static png logos and CSS files respectively. The templates folder contains the main HTML and Javascript files for the progressive web app.