Team DigiLearn

Software Test Plan v1.0:

The Digital Backpack

digiLearn

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

26 March, 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 serves to lay out the software testing plan that will be used to ensure the Digital Backpack is able to carry out it’s expected functionalities. Several methods of testing will be used for quality assurance including unit testing , integration testing, and usability testing. Unit Testing, described in Section 2.0, focusses on the individual units within the project that must be tested. Section 3.0 goes over integration testing, which is necessary for determining how well partner modules are able to work together. Finally Section 4.0, goes over usability testing. This document concludes with Section 5.0, which summarizes the testing strategies and overall goal of the test plan.

Team DigiLearn

2.0 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. A majority of the code base for this project is written in Python and Kotlin. As such, Python's built in library unittest and JUnit will be used to streamline the unit tests for these portions of the project. This section defines five major components of the Digital Backpack, and breaks each of these components into smaller user tests.

2.1 Google Authentication

Users are able to create accounts and sign in to the Digital Backpack application using a Google account through Google's Authentication system. The authentication flow consists of the following: if a user is not already logged in, bring them to the sign-in page; retrieve and store the user's credentials.

2.1.1 User Sign-In

Google offers a library that allows for the implementation of Sign-in with Google. Additionally, if a user is already signed into their account, they should bypass the sign-in screen and be taken directly to the application's main page. This sign-in functionality is handled client-side in Kotlin.

- *Boundary Values:* a valid Google Account, the correct password associated with the account, and GoogleSignInAccount object.
- *Expected Response:* If the user is already signed in, they will be brought to the main page, otherwise they will be taken to the Google Sign-in page. In the case that a user enters invalid account credentials, they will be rejected.

2.1.2 Server-Side Registration

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.

- *Equivalence Partitions::* Valid authorization code, Invalid authorization code, Malformed authorization code
- *Expected Response:* The server redeems the code and registers the user’s credentials in the database. The server then returns a success code to the calling module. In the case that the user is already registered in the database or the authorization code is invalid, the server returns a failure code instead.

2.1.3 Authenticating Requests

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.

- *Equivalence Partitions::* Valid Google ID token and user’s client secret, Invalid Google ID token and user’s client secret
- *Expected Response:* In the case of a valid ID token and associated client secret, the server will return to the calling function the user’s OAuth2Credentials object. In the case of invalid or malformed credentials, a failure code will be returned instead.

2.1.4 Retrieve & Store Credentials

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.

- *Boundary Values:* Google Credentials Object
- *Expected Response:* Server responds with an acknowledgement that the credentials were received.

2.2 File Upload/Download

Uploading and downloading files is an integral part of the Digital Backpack flow. The process is done on both the server and client sides.

2.2.1 Fetching File List

Before downloading any files, a file list is presented to the user. Presenting a file list saves on bandwidth for files that may be unnecessary for the user. When the server receives the user's information, it fetches a file list using the appropriate interpreter.

- *Boundary Values:* Web client secret, user data dictionary
- *Expected Response:* The server responds with a JSON containing the filedata, which consists of file names and file IDs.

2.2.2 Downloading Files to the Server

A file that is requested is downloaded and stored onto the server to be pushed to the client when queued. This is handled by the server using the Authentication Manager and appropriate interpreter.

- *Boundary Values:* Web client secret, user data dictionary, credentials, fileid
- *Expected Response:* The requested file is downloaded and stored on the server; The server responds with the metadata for that file

2.2.3 Pushing Files to the Client

Files that are stored on the server are ready to be pushed to the client. This is handled by the server with the DriveClientDownload method.

- *Boundary Values:* File name
- *Expected Response:* Using FileResponse, the server will return the appropriate file, using mimetypes to guess the file type.

2.3 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.

2.3.1 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.

- *Boundary Values:* Connection status
- *Expected Response:* A cloud image on the bottom part of the page.

2.4 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.

2.4.1 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 which 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 files 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.

2.4.2 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

2.4.3 Google Search Interface

A Custom Search Engine is used to provide a callable interface for Google Search. The Digital Backpack must be able to screen user generated searches and search results for inappropriate content and return relevant results for the user.

- *Equivalence Partitions*: a users search that contains restricted and/or non-restricted terms
- *Expected Results:* 1-1,000 results that do not include restricted sources or content.

2.5 Database

The Python 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.

2.5.1 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 is impossible and an appropriate error code is returned.

2.5.2 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.

3.0 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.

3.1 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 between each other inorder 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. When authenticating the user, the server should receive credentials from the client, and return a JSON response acknowledging the request. When initializing data, the server should receive metadata from the client, and return a JSON object containing the file data. When downloading data, the server should receive the file id of the requested data, and should respond with the metadata associated with that file.

3.2 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.

Boundary testing for the Storage Manager will include: ensuring that a user can only access files associated with them, as well as ensuring that a user can only upload files to their account. Attempting either of these flows without the correct key value pairs should throw an error to the server.

Boundary testing for the interpreter/Storage Manager interface includes: ensuring that files are stored locally in the correct user folder, as well as creating a folder for a user if one does not exist already.

3.3 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 are invalid or malformed, the REST interpreter returns a failure code instead.

4.0 Usability Testing

Observing a real-life user working with and testing the application is the purpose of usability testing. Users will be provided with the DigiPack application, being able to explore the functionalities our team has developed. The ultimate goal of usability testing is to view how the program functions on a user's end and is able to work properly for the final product. Our client, Dr.Vigil-Hayes has provided our team with participants, willing to allow the team to observe them working with the DigiPack application. These participants will check the UI of the application, software functionality and overall usability. With the usability testing, our team plans on observing a fully working application that is simple and enjoyable for the user to navigate through. If the usability testing results in a bad design or a non-functional application, team DigiLearn will take immediate action to respond to these errors. The goal of usability testing is not only to have a working application, but to discover the issues that need to be fixed in order to make improvements.

4.1 UI

Testing the UI for the DigiPack does not test the application's actual functionality but tests whether the presentation is desirable. As the user navigates through the DigiPack, we aspire for them to be pleased with the layout and design. Having a real-life user review the DigiPack UI will give our team a realistic opinion from our target users. If there are any bad reviews on the application's design, our team will immediately make improvements towards the front-end of the application.

4.2 Sign-In

The sign-in for the DigiPack is a crucial requirement as it allows the users to access the application. To ensure that there are no errors for users logging into the application, we want first to test two possibilities. First, we want the user to sign in using a Google account and be directed to the main page of the application. This tests the navigation of the application as well as the authentication process. The second scenario requires the user to leave the app and reopen it. This should lead the user directly to the main page instead of the sign-in page, recognizing that they have already been logged in. If the application performs these without error, then our sign-in is working correctly. If the user cannot be signed-in correctly, then there are errors in the REST interfaces and Google Authentication that needs to be fixed.

4.3 Upload and Download Files

Uploading and downloading files is another primary function of the DigiPack. A user should be able to upload assignments from the application to the internet and download files from the internet to the application. To test this function, the user will be given a file to upload to the application. The user will also retrieve a file from the internet and download it. If the program fails to execute one or both of these functions, issues in the software will be corrected.

4.4 Offline Queuing

The DigiPack needs to be able to queue the uploaded and downloaded requests while offline. To ensure that this function of the application is working correctly, we will have the user use the files previously used in the uploading and downloading process of the usability testing. Once the user can access these files with a reliable internet connection, we will have them disconnect from the WiFi. From there, we plan to see all requested queues able to be accessed. Along with making sure the queuing is functioning correctly, our team also wants to enable a smooth transition from online to offline.

4.5 Search Filtering

The Digital Backpack can service Google Search requests on the behalf of the user. The user will also be able to set blacklist parameters to prevent certain websites or categories of content from being returned to the user. This process will be validated through user testing. Testers will set their preferred parameters then make a search that may test these parameters. Testers will then confirm whether or not the Digital Backpack effectively filtered blacklisted content.

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, educational videos, 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 aimed to discuss DigiLearn’s thorough plan to test the software of the project. Grouped into three categories, unit, integration, and usability testing, each one being described into detail of the plan being used to carry out the testing, the expectations for the testing, and why this testing will be beneficial to the project.

Section 2.0 is an overview of how the unit testing will be carried out, Section 3.0 goes into detail about how the integration testing will be done, and section 4.0 describes the usability testing. Each of the categories are broken up by contributing parts of the project, allowing a plan for each to be discussed as well as the expected outcomes for each functionality.

This document has laid out the plan for the DigiLearn team to conduct viable testing to ensure that our product is functioning properly.