Team digiLearn  
Technological Feasibility

short line

Doctor Morgan Vigil-Hayes

Volodymyr Saruta

Caitlin Abuel

Grave Shirey

Israel Bermudes

Kristine Hermosado

Sebastian Kastrul

8 October, 2020

## 

Team digiLearn  
1.0 Introductionhorizontal line

short line

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. The COVID-19 pandemic has led to a sudden shift to remote learning, which disproportionately affects disenfranchised communities, further widening the education gap. These students rely on public hotspots to complete their assignments.

We are Team digiLearn, our members are Sebastian Kastrul, Israel Bermudes, Caitlin Abuel, Kristine Hermosado, and Gracy Shirey. Our team mentor is Volodymyr Saruta. We are working with our sponsor, 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 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 and Khan Academy.

This document serves to examine technological challenges our team may face during the development of our software. We begin in section 2.0 by introducing the main technological challenges for our project. Section 3.0 expands on these challenges with a detailed analysis of the technologies related to each specialized area. We aggregate these ideas in section 4.0, wherein each individual technology comes together to form our planned system. The final section, 5.0, concludes our feasibility document with a review of our goals and a summary of our system.

Team digiLearn  
2.0 Technological Challengeshorizontal linehorizontal line

short line

This section outlines the main technical challenges we will face for this project. These challenges relate to the major design aspects necessary to create a working product for our client. Each subsection discusses a technology, it’s purpose for the project, and the requirements related to that technology.

# 2.1 User Interface

## **2.1.1 Purpose**

The mobile application's goal is to provide offline service to K-12 students who have limited internet access. The application serves as a place where students can pull and work on online assignments without an internet connection.

## **2.1.2 Requirement**

Considering that the users of the application would be K-12 students, it is essential to have different layouts for each age group. We have come up with the following design challenges that we might face while completing the project:

* **Easy to Use** - With the different age groups, elementary students' cognitive ability is not as prominent as high school students. We will need to make a UI design that must be able to satisfy elementary students and high school students' expectations. With this in mind, we would need to be able to use a tool that allows team members to navigate through design features efficiently.
* **Collaboration Friendly** - Since the application will be based on K-12 students (users), the team will need to develop multiple visually appealing and informative designs to elementary/middle school and high school students. So we would use an application that lets multiple team members collaborate on the same design file.
* **Compatibility** - The application will be implemented in both Android and iOS operating systems, which would mean we will need to design based on Android and iOS. We would want to use a tool that allows cross-platform designs and be able to make prototypes as we go on with the development.

# 2.2 Cross Platform Development

## **2.2.1 Purpose**

The Digital Backpack is a mobile application used to service students while online and offline, mainly students from K-12. Since our project is targeting a wide audience, the cross platform development tool should be able to be accessed by multiple platforms. Our solution is further discussed in 2.3.3.

## **2.2.2 Requirements**

* **Simple Architecture** - Since the Digital Backpack is going to be used by students and teachers, it is important that our UI architecture is nice and simple. The application should be easy to use for both sides. We want a tool that allows our team to create a mobile application that the users can easily read and follow.
* **Reaches Across Multiple Platforms** - There is a wide variety of people that will be using this application. There are many platforms that are being used these days. Our goal is to be able to help students and in order to do that the Digital Backpack needs a tool that is able to reach across multiple platforms. The main platforms our team is looking at is IOS, Android, and Web Applications. Our mobile application requires a Framework that is flexible with all of these platforms.
* Runs Quickly and Smoothly
* Easy to Fix Bugs Fast

# 2.3 Networking

## **2.3.1 Purpose**

The Digipack must be able to network with educational services and retrieve data for the user. Content will be downloaded opportunistically, so it is essential that this transaction can occur as quickly and seamlessly as possible. Proxy servers can be used for our application to aggregate data related to school content while the user is offline. This pre-fetched data will ensure fast download speeds when a user reaches an opportunistic connection. We will also need a way to host this server. Our solution is further discussed in section 3.3.

## **2.3.2 Requirements**

We will create a proxy server that will aggregate data related to school content while the user is offline. A constraint from our client is that the proxy server should be linux-based, and the service should be hosted using cloud services. Our working prototype will be a cloud-based server that can serve 25-50 students. The feature requirements of the proxy server are as follows:

* **Integrates with Google Classroom APIs**
  + Fetch announcements, assignments, and grades
* **Integrates with Google Search**
  + Store search queries to be posted when a connection is available
  + Store top query responses for offline viewing
* **Manages OAuth credentials**
  + Further information about security is discussed in sections 2.5 and 3.5.
* **Provides a REST API**
  + Further information about REST services is discussed in sections 2.4 and 3.4.
* **Pre-fetches content requested by a user**
  + Allows content to immediately be pushed/pulled when opportunistic connectivity is established,

# 2.4 Multi-resource integration and storage

**2.4.1 Purpose**

The end user(s) of our product need to be able to access resources such as the ones described in section 3.4 in a quick and easy fashion while still conforming to the limitations of a mobile device. Student users will need to be able to access multiple web based educational resources and outlets all from our mobile application. Teacher users most likely would like to be able to provide lists of resources that students can access, assignments, and possibly even curate the sources available to their students on a class by class basis. Being able to integrate any and all possible resources with minimal refactorization is a key challenge facing this project.

We also need to consider the stability of our users connections to our service. Students will not always be able to stay connected to the internet while their documents download or upload. Our solution needs to be able to handle any sudden interruptions in the connection and have the ability to resume its work with minimal overlap.

**2.4.2 Requirements**

We plan to develop a service that will allow us to pull content from multiple resources requested by our client and store it in a uniform format so that it can be distributed to our users and display in a consistent format across multiple devices. The key features that we have defined are as such:

* **Uniform Data Storage**
  + Data pulled from resources will be stored on the server in a uniform format.
  + Format allows for quick parsing of data.
  + Consistent structure as most data will be passed to multiple users on many different devices/operating systems.
* **Multi-source compatibility**
  + As developers we need a way to pull data from any source and create a file containing all of the relevant information from that source.
* **Scalability**
  + We will need to be able to make new sources available to our users as the reach of the project grows without having to refactor the entire supporting structure.
* **Stability**
  + We need to be able to establish a stable connection with a user's device once an internet connection is available.
  + In the event of a disconnect in the middle of operation, we should be able to reestablish the connection and pick up where they left off.

# 2.5 Responsible Data Management

## **2.5.1 Purpose**

It is imperative that the Digital Backpack handles sensitive user-data End-users must be confident that any information they transfer using the Digital Backpack is safe. Furthermore, the Backpack must be compliant with the Family Educational Rights and Privacy Act (FERPA) which makes fastidious security exceedingly important. Given the nature of our proposed network architecture, there are four main points of intrusion to the system: the user’s device, the proxy server, the database, and the network connections between these devices. Each of these points must be maximally secured against intrusion to ensure the inviolability of user data.

## **2.5.2 Requirements**

Security will be a consideration for every aspect of our system. For the purposes of our working prototype, the system must have standard security against intrusion. The following items must be implemented for our product to be appropriately secure:

* **Encryption**
  + Any and all sensitive data stored at any location must be encrypted. Additionally, this data must be encrypted while it is being transported between devices.
* **Authentication**
  + Communications between devices must be authenticated to confirm that any information served is served only to authorized devices.
* **Device Security**
  + The end devices, the user’s device and the proxy server in this case, must be secure against virtual and physical intrusion.

Team digiLearn  
3.0 Technology Analysishorizontal linehorizontal line

short line

This section goes into depth on the specific technologies we will use to address the challenges discussed in section 2.0. Each subsection will list the potential alternatives, our chosen approach, and the feasibility of the technology.

# 3.1 User Interface

## **3.1.1 Alternatives**

In deciding what tool to use for designing the UI, we have reviewed the challenges we might encounter; we have found different solutions that we will analyze based on a set of criteria we think is best for the project. We will rate each tool between 1-3, with 3 being the best fit with the mind's requirements.

* **Sketch** 
  + It's a digital design toolkit which is based on vector graphics. It's a great tool to use when designing the interface, but it is only based on iOS, but our team plans to do Android and iOS software implementation. There is also an account membership fee needed.
* **InVision Studio** 
  + It's a free vector-based digital product design platform that helps develop screen design processes. The main idea is to make it easier for UI designers to create prototypes of the designs they envisioned. The downside of this is that it doesn't offer the designing aspect. It would be a great tool to use when we're creating prototypes for the client.
* **Axure** 
  + It's similar to InVision, as it's also a digital product design platform. However, it is not as effective as InVision because it tends to be more glitchy when trying to design multiple screens, and it can get laggy when switching from different screen designs. There is an account membership fee required to be able to use the tool.
* **Adobe Experience Design** 
  + It is a relatively new tool that allows the combination of Sketch and InVision Studio because it provides the designing process and prototyping mobile applications. It encourages team collaborations within multiple projects.

## **3.1.2 Our Approach**

The following table is the summary of the result of each tool based on the criteria we have chosen:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tools** | **Easy to Use** | **Collaboration Friendly** | **Compatibility** | **Total** |
| Sketch | 1 | 1 | 0 | 2 |
| InVision Studio | 1 | 1 | 1 | 3 |
| Axure | 0.5 | 0.5 | 1 | 2 |
| Adobe Experience Design | 1 | 1 | 1 | 3 |

In conclusion, our team decided to use Adobe Experience Design as the tool to use when tackling the application interface challenges as it satisfies the criteria we had in mind. It will help us make multiple designs and collaborate on the same file synchronously, and from there, we can quickly create wireframe prototypes of the design we created. Since it is fairly new, we would have InVision Studio as our backup.

**3.1.3 Feasibility**

To prove that Adobe Experience Design will be able to help us in the design process of the project, we plan to design a simple UI that will demonstrate the following process:

* User login layout
* Homepage layout
* User profile layout

# 3.2 Cross Platform Development

The cross platform development tool is important to our mobile application. We need the tool to be able to create a simple and easy-to-use application that is able to work with IOS, Android, and Web Applications. This tool also must have an architecture that is visibly pleasing and runs quickly and smoothly to make it easier and more enjoyable to the customers we are targeting. The framework we choose to code our application with must also be easy for us to fix bugs quickly. The following section includes the tools that were looked at and researched, as well as the tool that our team decided to use and how we will be using it for cross-platform development for our mobile application.

**3.2.1 Alternatives**

* **Flutter**

Flutter is a service recommended to our team by our client Dr.Vigil-Hayes. Flutter is a cross platform tool made by Google. Flutter works with Android, IOS, and web devs. Flutter enjoys priding itself on working very quickly allowing creators to build UI and fix bugs within seconds. Using Google’s code language, Dart, Flutter creates beautiful and simple looking applications with a large selection of widgets.

* **React Native**

While searching forums of cross platform development compared to Flutter, I found React Native. React Native is a cross platform tool that can be used with both Android and IOS. React Native was released in 2015 by Facebook and is used by that platform. It is also used with many other apps such as Instagram, Discord, and Skype. Using JavaScript, React Native allows its users to create a simple UI that allows you to integrate quickly.

* **Ionic**

While searching for other tools similar, but providing differences, to Flutter and React Native, I came across Ionic. Ionic is a UI Framework that is flexible across many platforms. The technology was created in 2012 and is used by companies such as AAA and NASA. Ionic uses JavaScript to easily create fast, simple, interactive applications. The cross platform Ionic creates allows the architecture the user has created to adapt to different platforms. Other than JavaScript, Ionic grants its users the ability to use other frameworks such as Angular, React, and Vue.

## **3.2.2 Our Approach**

As a team we carefully went over each tool and what services they would provide for us. We ended up choosing Flutter, the cross platform development tool created by Google. There are various reasons why we choose to go this route. The design that Flutter provides looks very nice and neat and the application runs very smoothly, which is an experience we would like to give the users of the Digital Backpack. Flutter has a wide variety of widgets to use for the application as well. Although React Native has all of the requirements our team is looking for, forums comparing it to Flutter shows the cons the tool has. While both have great simple UI, Flutter adapts better to different platforms, while React Native looks the same. F

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tools | Simple Architecture | Reaches Across Multiple Platforms | Runs Quickly and Smoothly | Easy to Fix Bugs Quickly | Total |
| Flutter | 2 | 3 | 3 | 3 | 11 |
| React Native | 2 | 2 | 2 | 3 | 9 |
| Ionic | 3 | 3 | 2 | 2 | 10 |

**3.2.3 Feasibility**

To prove the feasibility for Flutter, we will have to do some testing with the tool first. After downloading the tool, we will need to research the Dart language and how to implement it. Flutter also includes multiple widgets that will help us design our application. While playing around with the widgets will help us figure out what features we would like the application to include. One of the widgets that Flutter includes is a platform view widget. Once this widget is implemented through both the Android and IOS sides, they can be put together to create a platform view for both.

# 3.3 Opportunistic Content Delivery Networks

Opportunistic Content Delivery Networks (oCDNs) facilitate asynchronous communication in situations where a consistent connection is not available. Delay Tolerant Networks (DTN) is a term sometimes used interchangeably with oCDNs. This document will refer to the architecture as oCDNs. The following sections discuss the oCDN architecture, software possibilities, cloud services, our approach, and the feasibility of this technology.

## **3.3.1 oCDN Architecture**

The traditional internet protocol suite is primarily based on TCP and IP. TCP/IP has certain assumptions that do not allow for prolonged delays in communication. The oCDN architecture uses a method of store-and-forward message routing to account for these variable delays. The Digital Backpack relies on queues to keep track of requested user data. We will develop the necessary nodes to maintain these queues within this system.

To implement an oCDN architecture, there are some key characteristics that are necessary. Desired characteristics for our chosen applications are as follows:

* Integrates necessary services such as Google Classroom\*
* Provides quick download and upload speeds
* Scalability
* Fair Pricing

\**Integration with external sites is further discussed in section 3.4*

## **3.3.2 Proxy Servers**

Proxy servers are a necessary part of the Digital Backpack ecosystem. A proxy server acts as a middleman between the user and the main server. Proxy servers often store commonly requested data, such as the front page of a website or a website’s logo. The benefit of this is faster connection speeds, as not all the data needs to be requested from the main server. Ensuring speedy connections is essential for the Digital Backpack, which relies on gathering data opportunistically. There are many potential services to use for the proxy server.

* **Squid**

Squid is a web proxy cache server application that provides proxy and cache services for HTTP, FTP, and other such protocols. Squid promises reduced bandwidth and improved response times by caching and reusing frequented webpages and is scalable from business to enterprise levels. Squid offers many features including access control, authorization, content distribution, and traffic management.

* **Kong**

Kong is an open source API gateway. An API gateway is a management tool that acts as a middleman between client and backend services. It works as a reverse proxy that accepts API calls, aggregates various services, and returns the appropriate result. These services tend to handle common tasks such as user authentication, rate limiting, and statistics. Kong is considered a transparent proxy, which can allow for content filtering and transparent caching.

* **Apache Traffic Server**

Apache traffic server is a scalable HTTP caching proxy server. ATS offers high-performance for both forward and reverse proxying of HTTP/HTTPS traffic. A reverse proxy intercepts requests and serves either a cached copy of requested data, or submits a new request to the originating server. A forward proxy acts as the origin server to the user.

The best proxy service that suits the needs of this project is Squid. The explanation for this is further discussed in section 3.3.4.

## **3.3.3 Cloud Services**

To host our server we will be utilizing cloud services. Cloud services can be used alternatively to physical data centers, which can be much more costly. These services offer computing power, storage, and databases. Some use cases for cloud computing include data backup and recovery, email, virtual desktops, software testing, and web applications. The benefit of cloud computing is it’s flexible pricing and scalability.

* **DigitalOcean**

DigitalOcean is a cloud hosting service built by developers with a simple interface to provide for easy configuration. Because of its simplicity and ease of use, it is targeted towards smaller scale projects that don’t require an all encompassing cloud computing ecosystem.

* **Google Cloud**

Google Cloud is Google’s hosting service. As a major corporation Google promises security and privacy. There is a free trial tier for their cloud services, but pricing varies depending on what specific services are needed.

* **Azure**

Azure is a cloud hosting service that offers broad catering to a wide variety of needs. Webapps can be built using the framework of the developer’s desired language of choice. Azure has stability but at higher operating costs.

The best cloud service that suits the needs of this project is DigitalOcean. The explanation for this is further discussed in section 3.3.4

## **3.3.4 Our Approach**

Based on recommendations from our client and an analysis of the available services, we have chosen Squid to serve as the proxy server, and DigitalOcean to serve as the cloud computing service for our project.

|  |  |  |  |
| --- | --- | --- | --- |
|  | API Integration | Connection Speed | Scalability |
| Squid |  |  |  |
| Kong |  |  |  |
| Apache Traffic Server |  |  |  |

*Figure: Proxy Server Comparison Chart*

Squid works best for what we need for the proxy server. Generally, proxies serve the same purpose of caching data to reduce bandwidth. Choosing a service ultimately came down to the overall performance in regards to speeding up connections through effective caching. Another factor was the pricing of the service as there is no revenue associated with the Digital Backpack.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Configurability | Scalability | Pricing |
| DigitalOcean | Lower configurability, but higher ease of use |  | Starter - 0$/mo  Basic - 5$/mo  Professional - 12$/mo |
| Google Cloud | Decent configurability |  | Highly Variable |
| Azure | Higher configurability but lower ease of use |  |  |

*Figure: Cloud Service Comparison Chart*

DigitalOcean suits our needs as a simple cloud service to host our proxy server. Our goal is to make a small scale prototype for testing to work for around 25-50 students. DigitalOcean has the benefit of being an easily configurable service while still having the potential to scale, making it perfect for our envisioned project. Additionally DigitalOcean offers rates on the cheaper side when compared to cloud services intended for extensive enterprise level systems.

**3.3.5 Feasibility**

To prove the feasibility of our chosen approach, we will implement a simple proxy server hosted on the cloud. We will test its capabilities using sample data from a database.

# 

# 3.4 Multi-resource integration and storage

The range in maturity, education, and age of our users presents an interesting technical challenge. We need to be able to pull resources requested by the users, push resources provided by teachers, have the option for educators to restrict the content available to their students, all while still having the ability to access just about anything educational through our service. End user storage capabilities is also a limiting factor in this scenario, many mobile devices only have approximately 16GB of on-board storage, much of which is taken up by the operating system and applications. Therefore we also need to find a solution that is compressible and can possibly be stripped down to the raw content from the connected services.

Internet connection stability is not a given with this project. Our service also need to be able to handle sudden disconnections, packet loss, and other forms of connection disruptions. If, for example, a user has to go home immediately after school and doesn't have time to finish downloading or uploading their files, a note should be made of where the transfers were stopped and when a connection is reestablished, resume the process from there.

The main factors of the above scenario that this section will address are the ability for our service to push and pull resources from the services that have been outlined for us by our client. These services include:

* Google Classroom
* Google Docs
* Khan Academy
* Youtube

as well as providing an interface that allows for other services to be "connected" to ours as needed. Because the content from each of these services is so diverse we need to be able to organize it in such a way that allows for it to be filtered and then pushed to the students devices without disrupting the homogeneity of our app.

At the core of these requirements we have [number] characteristics that we would like to address:

1. Support for the requested services.
2. Scalability for future support of other services.
3. Modularity in the event that one of the implemented services changes their content format.
4. The ability to implement our own structure and/or format to the data received from each service.
5. Simplicity in the data from each service so that it can be scaled up or down depending on the needs of the end user.
6. The ability to handle unstable connections and resume operation after potentially long periods of lost connectivity.
7. inexpensive/free due to the non-profit nature of this project.

## **3.4.1 Alternatives**

Parsing HTML files/web crawlers

OpenSearchServer - <https://www.opensearchserver.com/documentation/api_v1/web_crawler.md>

The initial idea our team came up with for collecting web content for our users was to implement a web scraper/crawler that would fetch HTML files and their embedded objects as they were needed and store the raw data locally. Web crawlers and scrapers have been around for decades and are a tried and true method for pulling web content for local storage. The above link is a web crawler with a relatively simple and easy to use interface.

Network Sockets

Python Socket - <https://docs.python.org/3/library/socket.html>

To establish a connection with our users a simple network socket protocol could be used in tandem with a web crawler to pass them the data pulled from websites. Network sockets have been in use for just as long if not longer than web crawlers and were originally defined by Joel M. Winett in 1971 for RFC 147 ( <https://tools.ietf.org/html/rfc147> ). Like web crawlers, sockets are a simplistic solution that can be relatively easy to implement and allow for multiple connections to the same data at the same time.

RESTful Services

REST architecture - <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>

Our client, Dr. Vigil-Hayes, recommended we look into REST APIs and associated RESTful services. REST services as a concept were defined by Roy fielding in 2000 alongside the HTTP 1.1 protocol. REST services provide a structure to requesting web content and local standardization of content for ease of use and modularity. While being a relatively new technology, many large organizations and companies have begun to adopt the concept in an attempt to standardize data that is used across multiple interfaces, operating systems, and applications.

## **3.4.2 Our Approach**

In combination with the requirements specified by our client and the characteristics that we defined in section 3.4, we analyzed each of the options that were introduced in the previous section to try and find the best solution for our project. We first defined a few of the requirements that we felt were more important as the others, namely:

* The ability to interface with the services requested by our client.
* Modularity to be able to add interfaces as needed in the future.
* inexpensive/free to use due to the non-profit nature of the project and our limited budget.

These three requirements shaped our initial scope of research into possible alternative solutions. The alternatives listed in section 3.4.1 all fit this description and so we felt as though these were a good place to start testing.

While we did not do any testing specific to this project of any of these solutions, we pulled from our team's experience with each of these solutions to come to a decision on which solution to use in our initial implementation. A few members of our team are currently taking CS 460 - Computer Networks with our client at NAU and have experience implementing network sockets, others have professional experience using and implementing RESTful services. Most of us grew up with pre-google internet and have personal experience with web crawler based search engines. All of this experience has allowed us to compile a list of pros and cons to each alternative.

Web Crawlers

Pros - Allow us to pull content from any online resource at any point, relatively simple implementation, many different iterations to choose from in the event we find a shortcoming or feature that our chosen solution doesn’t implement.

Cons - Content is slow to produce because each page has to be parsed individually, content is not in a standard format after parsing, large objects or stream data such as video or audio cannot be easily accessed or stored, potential legal issues acquiring and storing audio or video content. This only solves half of the problem, getting the web content. It does not allow us to pass the content to the users on its own.

Network Sockets

Pros - Fast connections, secure connections can be implemented, real-time data transfer is possible, very little overlap in data during connection instability due to sequencing (TCP)

Cons - connection instability could lead to corrupted files on either side of the connection, the required computational overhead for a continuous connection can be inefficient for small scale hardware implementations or large scale software implementations. This as well is also only half a solution, it only allows us to establish a connection with the users, doesn’t inherently give us the ability to pull web content.

RESTful Services

Pros - Interfaces with all of the requested resources and many more, building “interpreters” for each of the requested services allows us to build a local standard for data pulled from each source and therefore create a separation in development between the mobile app and the server, the inherent “statelessness” of REST services allows us to dynamically request and provide data both on the server-resource connection and the client-server connection with little to no need for a persistent connection.

Cons - REST services don’t inherently have any sort of authentication built into them to ensure that the user is who they say they are, however because of the fluid nature of RESTful services we can implement a solution to that ourselves and integrate that into our service.

The following table is a visual representation of the pros and cons listed above in relation to our desired characteristics:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | support | scalability | modularity | Localized structure | Data simplicity | Connection | cost |
| Web Crawlers | all | Difficult but possible | none | case-by-case | Extremely complex | n/a | free |
| Sockets | n/a | n/a | n/a | n/a | Relatively simple | Dependent on protocol | free |
| REST services | all | Inherent by design | Inherent by design | Inherent by design | Relatively simple | Dependent on protocol | free |

As we can see REST services meet almost all of our requirements with flying colors (no pun intended) and should allow us to build a service that fulfils our needs and allows us to make modifications to each part of the system independent of the requirements of other functionalities.

**3.4.3 Feasibility**

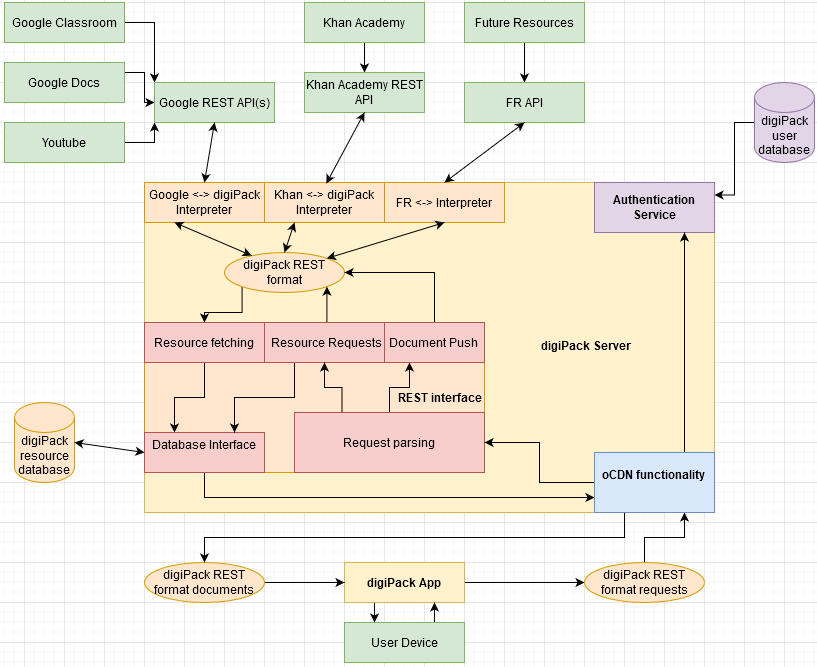
Because RESTful services are more of a concept than a fully fleshed out API or library we will need to do quite a bit of testing and development to create a service that fits our needs. Our initial development and testing plans will be to settle on a format (HTML, XML, JSON) to use as our internal standard and what those files will include. From there we will develop a storage solution and interface. The interface will have three core sections that will have to be developed and tested:

1. A way to request and receive files from the interpreters
2. An interface with the local resource database
3. An interface to send and receive data with the user(s)

These three sections will be further broken up to improve functionality, modularity, and minimize coupling. The interpreters will have to be built for each resource and so we will start with one resource interface as a proof of concept and expand as needed. These interpreters will take the data from each resource’s existing REST API and create a file based on our previously mentioned local format. We will also need an interface with the servers database to store, organize, and search through the RESTful data. We will also need to implement a way for users to push data (documents, emails, etc.) to their respective resources through our app. Finally we need a way to ensure that the users are who they say they are and a way to tell that through the data sent between their device and our server. An interface will be created to parse their requests and cross reference their login data with our local authentication specified in section 3.5.

All of these pieces will come together to create the backbone of the digiLearn server and functionality. Each will have to be tested for proper input and output (file type, syntax, etc.), as well as their ability to handle multiple requests at the same time. By breaking the “REST Interface” down into subsections we can build and test these pieces independently so that once each is complete we are able to show and prove that each functions properly.

Below is a flow chart generalizing the use cases and interfaces that we currently foresee our REST service needing to be fully functional. The green represents services or interfaces that already exist, yellow for the digiLearn serve as a whole, orange for the REST service, red for the internal functionality of the REST service, purple for the authentication covered in section 3.5 and blue for the oCDN connectivity functionality of the server.



# 3.5 Security

The security of our system is integral to our product being usable. There are four main components of the system which must be secure: the proxy server, the user’s device, the database, and the network connections between these devices. The following sections will explore the possible approaches for each of these components, and this section of the document will conclude with a summary of our approach to system security and the feasibility of keeping user data secure in this context. To outline the specifications for the security of the DigiPack system, consider the following desired characteristics:

* **Proxy Server**
  + The proxy server must only accept requests from the user’s device and from authorized external services.
  + The proxy server must be protected against intrusion from unauthorized sources.
  + In the case of intrusion, user data stored locally must be further secured with strong encryption.
* **User Device**
  + The application on the user device must only communicate with the proxy server. It must be impossible for the application to communicate with any other device.
  + Other applications on the user device must be prevented from subverting this connection to communicate with the proxy server.
  + In the case of intrusion, user data stored locally must be further secured with strong encryption
* **Network**
  + Connections to every device in our system must only be accepted from authorized sources. These connections must be protected from interception.
  + Sensitive user data transported between devices over the internet must similarly be safe from tampering or theft.

## **3.5.1 Proxy Server Security**

The proxy server is acting as an intermediary between the user and a variety of services on the internet. As such, it is unavoidable that the proxy server will have to store sensitive information while the user’s device is offline. As a matter of best practice, it is essential that user data is secure for the entirety of its life on the proxy server. The following technologies will be considered for this purpose:

* **Server Firewall**

THe proxy server could be equipped with a firewall, such as the popular Uncomplicated Firewall (UFW) to manage traffic that comes into the server. This would be an effective way, in conjunction with OAuth2, to prevent the server from being compromised.

* **Encryption**

As an extra layer of protection, any user data that is stored on the server should be encrypted so that it is still secure in the case that the server is compromised. Standard RSA / AES encryption would suffice for this purpose, but this raises the issue of keeping essential keys safe. One option is to store the keys in memory once the server is started. This would require manual intervention, however, and may result in encrypted data being inaccessible in the case that the server must be restarted.

## **3.5.2 User Device Security**

The Digital Backpack must function offline. Thus, the digital backpack must store potentially sensitive data locally. This makes the user device a potential place where user data may be compromised. As a result, it is essential that all data on the user’s device is stored securely. The following technologies will be considered for this purpose:

* **Secure Storage**

Mobile devices offer a secure storage environment (Keychain for iOS and KeyStore for Android) explicitly for the storage of particularly sensitive data. This is a small amount of storage, however, and so would only be valuable for encryption keys and passwords. In addition, a browser application would not be able to rely on this service.

* **Encryption**

User data can be encrypted when stored locally. The encryption keys could then be stored in the secure storage box described above. The main drawback is that encrypting and decrypting data can be a time-consuming process.

* **User Sessions**

Appropriately managing user sessions will increase the security of the user’s device and thus the system as a whole. User session should be terminated periodically, a login should be required. Biometric authentication will not be permitted as this technology may exclude many devices and limit the scope of our potential user base.

* **External Authentication**

If necessary, user sessions could be authenticated by matching the inputted password with a cryptographically hashed password stored in a separate database. This solution requires a connection to the internet and thus is not ideal for most applications in the context of the digital backpack.

**3.5.3 Network connections**

Any communication occurring between devices must be secured to prevent requests from being tampered with and to keep user data secure during transit. In the following list, multiple technological alternatives are detailed:

* **VPN Service**

The use of a proprietary VPN service would ensure that any user data sent over public networks, as is often done in the context of the Digital Backpack, is protected until said data reaches the VPN’s endpoint. However, user data would then be vulnerable in transit from the VPN’s endpoint to the end device. In addition, the use of this outside service would require a monthly fee which may be prohibitive to the deployment of our product.

* **RSA / AES Encryption**

A combination of AES and RSA encryption would allow for all user data and requests to be encrypted for the entirety of their transit from one endpoint to another. The public-key system in RSA will enable the automation of trading keys and establishing a secure connection of encrypted data. This solution raises the issue of securely storing these encryption keys. Additionally, encryption unavoidably has an impact on performance, but the bulk of this performance impact can be handled offline, so it won’t interfere with the limited, opportunistic connections to the internet.

* **OAuth2**

OAuth2 is an industry-standard authentication framework which would resolve the issue of making sure all end devices in our system only communicate with authorized devices. OAuth2, however, does nothing to protect data while it is being transferred.

* **Transport Layer Security**

Transport Layer Security is a standard security measure and part of the https protocol. SSL implements a secure, encrypted, and bound connection between the client and the host. This protocol is supported by Flutter libraries, which would make implementation relatively simple. On the other hand, the fact that this protocol is standardized may be a vulnerability.

**3.5.4 Our Approach**

For the three major facets of security in our system, we have decided on the following strategies to ensure the security of sensitive user data:

* **Proxy Server**

The Proxy server will be reinforced with a firewall using UFW. All user data that is stored on the proxy server will be encrypted with a local AES encryption key that will be stored in the server’s RAM to prevent unauthorized access

* **User Device**

Access on the user device will be managed with user sessions that are authenticated with a username and password. In the case of the mobile application, the password will be compared to a hashed password stored locally in the device’s secure storage. In the case of the web application, the password will be compared to a hashed password stored remotely in a database.

Sensitive information stored on the user device will be encrypted with an AES key. This key will be stored in the device’s secure storage

* **Network**

All network traffic in the DIgital Backpack system will be implemented in accordance with the https protocol utilizing TLS for transmission security. Connections will be authenticated using OAuth2 technology, as it is the industry standard.

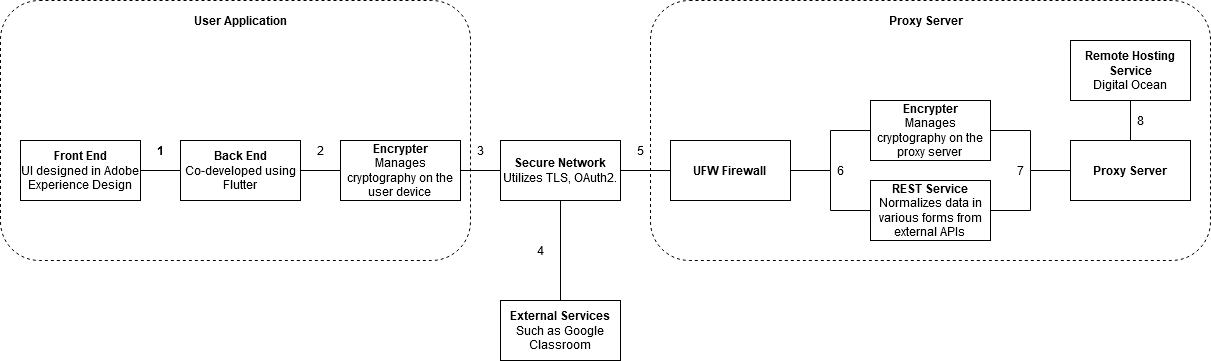
**3.5.5 Feasibility**

To prove that handling sensitive user data in the context of the Digital Backpack is feasible, we will be performing testing and experimentation with the technologies outlined in section 3.5.4. Once we have a functional prototype, we will routinely test the security of our device by making a best effort to compromise our own security.

Team digiLearn  
4.0 Technology Integrationhorizontal linehorizontal line

short line

This section discusses the final plan for our system. Each aforementioned technology comes together to create the Digital Backpack. To begin, please consider the following preliminary system diagram which summarizes our proposed architecture at this point in development.



The following subsection will further explain the above diagram. The subsection number corresponds to a number on the diagram. For example, section 4.1.0 gives more detail to area 1 on the diagram.

**4.1.0 Front / Back End Interaction**

As is standard, the Back-end of the user application will be the model from which the front-end user interface works off of. More unique to the Digital Backpack is the need for multiple, unique interfaces to adequately serve students in a wide age range. The Back end, then, will also provide a mechanism to switch between these interfaces at the user’s preference.

**4.2.0 User Application Encrypter**

The encryptor is a component of the back-end logic in the user application that handles encrypted data coming in from the network and prepares data from the user application to be sent over the network.

**4.3.0 Networking the User Application**

The user application will be compliant with all standards for secure connection as laid out in section 3.5.4. In summary, the application will utilize OAuth2, and TLS, and will encrypt all sensitive data sent over the network.

**4.4.0 Networking External Services**

External services will communicate with the proxy server over a network connection. Depending on what technology is supported by these services, these services may or may not comply with the standards for network connection as we laid out in section 3.5.4.

**4.5.0 Networking the Proxy Server**

The proxy server will be compliant with all standards for secure connection (as described in section 3.5.4) when communicating with the user device. When communicating with external APIs, such as Google Classroom, the proxy server will comply with the standards that these APIs require.

**4.6.0 Data Handling on the Proxy Server**

Information that leaves the proxy server either needs to be encrypted or pushed through the RESTful service to remain compliant with the format needed by external APIs or the user application.. Similarly, information reaching the proxy server either needs to be decrypted or pushed through the RESTful service to be processed into a consistent and useful format for use in the larger Digital Backpack system. In some cases, both of the above actions may be necessary.

**4.8.0 Server Hosting**

The proxy server will be hosted on a digital droplet provided by the Digital Ocean service.

Team digiLearn  
5.0 Conclusionhorizontal linehorizontal line

short line

With many students across the country lacking internet access at home and potentially struggling to find reliable internet access in general, and with the rising prevalence of remote learning, the digital gap and its impact on student outcomes is more prominent than ever. The Digital Backpack offers a solution by implementing an opportunistic Content Delivery Network or oCDN to enable our user application, the digiPack, seamless transition between online and offline learning and thereby lessen the effects of this digital gap.

In this document, we enumerated the technological challenges faced by this goal and explored a myriad of potential solutions. For the purpose of making this application available to as many students as possible, we have settled on Flutter as a platform to simultaneously develop our application for iOS, Android, and as a web application. The user interface for the mobile application will be developed using Adobe Experience Design. For the scope of this project, we have decided to host the Linux proxy server using the Digital Ocean cloud service to host said server. We have also concluded that developing a RESTful service to facilitate communication between our system and various external APIs is definitely possible within the scope of this project. For security, a suite of standard measures will ensure that user data is kept completely safe and secure.

In conclusion, our research into the technological feasibility of the Digital Backpack has led us to establish a strong foundation for the overall architecture of the system, and have concluded that there are no technological challenges that will prevent the creation of the Digital Backpack system.