Carleton Unite

A SOCIAL NETWORK FOR CARLETON

CARLETON UNIVERSITY
COMP 4905

Name: Sahaj Arora
Student Number: 100961220
Supervisor: Louis Nel
Date: December 15, 2017
ABSTRACT

This report is intended to walk you through the Carleton Unite hybrid mobile application. It explains how Carleton Unite app can serve as a highly resourceful tool for students to connect and get all the help they need in their journey through the university. It gives details about the concept of Hybrid app development. It describes why a hybrid app was a good choice for this project and how it can be useful for students to learn Hybrid app development. It describes the system specification, architecture, implementation, as well as instructions to successfully set up the environment required to run and/or modify the system. Furthermore, it also talks about the result and potential future development of the Carleton Unite system.
ACKNOWLEDGEMENTS

I would like to thank Professor Louis Nel, who agreed to supervise this project, recognized my skills, and provided me with his reviews on the final product.

I would also like to thank Carleton University for letting students pursue projects like these which not only help them to develop technical skills, but also help them to enhance their communication and writing skills. Projects like these undoubtedly give students an edge in the industry.

Last, but not the least, I would like to thank the creators of the website Udemy [1] and the teachers providing lessons on Udemy. Without those lessons, this project would not have been possible.
## TABLE OF CONTENTS

### CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>1</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>2</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>3</td>
</tr>
<tr>
<td>TABLE OF FIGURES</td>
<td>5</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>6</td>
</tr>
<tr>
<td>BACKGROUND</td>
<td>7</td>
</tr>
<tr>
<td>SPECIFICATION</td>
<td>8</td>
</tr>
<tr>
<td>ARCHITECTURE</td>
<td>9</td>
</tr>
<tr>
<td>NPM</td>
<td>9</td>
</tr>
<tr>
<td>FIREBASE PUSH NOTIFICATIONS</td>
<td>9</td>
</tr>
<tr>
<td>FRONT-END</td>
<td>10</td>
</tr>
<tr>
<td>IONIC 2</td>
<td>10</td>
</tr>
<tr>
<td>PARSE JAVASCRIPT API</td>
<td>11</td>
</tr>
<tr>
<td>BACKEND</td>
<td>11</td>
</tr>
<tr>
<td>NODE.js</td>
<td>11</td>
</tr>
<tr>
<td>EXPRESS.js</td>
<td>11</td>
</tr>
<tr>
<td>PARSE-SERVER</td>
<td>11</td>
</tr>
<tr>
<td>MONGO DB</td>
<td>12</td>
</tr>
<tr>
<td>IMPLEMENTATION</td>
<td>13</td>
</tr>
<tr>
<td>BACK-END</td>
<td>13</td>
</tr>
<tr>
<td>DATABASE STRUCTURE</td>
<td>13</td>
</tr>
<tr>
<td>NODE.JS SERVER</td>
<td>18</td>
</tr>
<tr>
<td>FRONT-END</td>
<td>20</td>
</tr>
<tr>
<td>PROJECT STRUCTURE</td>
<td>20</td>
</tr>
<tr>
<td>SOME IMPORTANT CODE SNIPPETS</td>
<td>25</td>
</tr>
<tr>
<td>SETUP AND LAUNCH INSTRUCTIONS</td>
<td>28</td>
</tr>
<tr>
<td>BACK-END</td>
<td>28</td>
</tr>
<tr>
<td>FRONT-END</td>
<td>29</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>RESULTS AND EVALUATION</td>
<td>31</td>
</tr>
<tr>
<td>FUTURE WORK</td>
<td>33</td>
</tr>
<tr>
<td>FEATURE ADDITIONS</td>
<td>33</td>
</tr>
<tr>
<td>CODE CLEANUP</td>
<td>34</td>
</tr>
<tr>
<td>TESTING AND RELEASING</td>
<td>34</td>
</tr>
<tr>
<td>CONCLUSION</td>
<td>35</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>36</td>
</tr>
</tbody>
</table>
# TABLE OF FIGURES

**FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User Chats - Mobile App</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>Messages - Mobile app</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>Node.js server file (server.js)</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>Ionic project directory structure</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>App root module: app.module.ts</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>My-Courses view: Mobile app</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>My Courses page : Mobile app</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>My Courses view: my-courses.html</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>My-Courses controller: my-courses.ts</td>
<td>24</td>
</tr>
<tr>
<td>10</td>
<td>Parse initialization (/src/app/app.component.ts)</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>Get courses followed by current user (/src/pages/my-courses/my-courses.ts)</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>Subscribing to Parse Live Query for incoming message (/src/pages/browse/browse.ts)</td>
<td>26</td>
</tr>
<tr>
<td>13</td>
<td>Subscribing to an event (/src/app/app.component.ts)</td>
<td>27</td>
</tr>
<tr>
<td>14</td>
<td>Publishing an event (/src/pages/browse/browse.ts)</td>
<td>28</td>
</tr>
</tbody>
</table>
INTRODUCTION

The goal of this project is to create an iOS and Android mobile application that will help students of Carleton University to stay connected with their fellow students and hence be able to ask or answer course specific questions on the go. This application not only helps its users to follow course threads (referred to as “Rooms” in the app) and ask or answer questions, but it also allows users to contact other users directly using the app’s very own live chat feature. It also lets users see what courses other users follow. Users of this app are notified of messages received via push-notifications, which lets users reply to their messages quickly, and hence enable students to get answers to their questions faster.

The intended audience of this application, as of now, is the students of Carleton University. This application can be further modified to be a generalized platform for all Universities across the world.

Because three months are not enough for a single person to develop such an app for both Android and iOS platforms natively, the cross-platform development technology called Hybrid App Development has been used to develop the Carleton Unite app for both Android and iOS. Hybrid App Development is a rapidly evolving technology that enables developers to have one code-base for multiple platforms.

This approach towards building a mobile platform to help students is not only an efficient solution in terms of time and productivity, but also a motivation to encourage students to learn Hybrid App Development during their journey through the university because along with sharpening one’s web-development skills, it also introduces one to the field of mobile app development, helping one briefly understand mobile app lifecycles and architectures, and eventually have a high edge in the industry right after graduation.
BACKGROUND

Being an Indian international student in a Canadian university, it was very difficult for me, in the beginning, to understand the way courses in a university worked. I was not completely familiar with things like locations of lecture rooms, assignment and exam schedules and structures, course content, study guidelines, grading systems, etc. It was in the first year of my studies at Carleton University when I realized that there should be an online social platform for students to ask questions and get answers quickly.

Students can surely send e-mails to other students using the Carleton’s online portal, yet it is very inefficient and slow to communicate via e-mails. Also, e-mails only feature direct messaging between two or more people. There exists no such platform which has a forum system to let students post questions or answers, follow any course forum they want, and get notified directly on their phones using push notifications. I felt a strong need to have such a system for students so that they can get all the help they need easily, quickly and on the go.

Hence, I came up with an idea to develop the Carleton Unite mobile application which would not only let students communicate with other students using live chat, but also let students stay up to date with course forums (referred to as “Rooms” in the app). Using the Hybrid App Development technology, I was able to develop this cross-platform mobile application for both Android and iOS in a period of three months. The only constraint to this approach is that hybrid apps do not perform as well as native apps yet. However, this technology is evolving rapidly with frameworks like Ionic, React, Xamarin, etc. While the performance is not yet as well as the native apps, it sure is a lot better than what it was a few years ago. If proper design guidelines are followed, then just like native apps, hybrid apps can be very fast and responsive.
SPECIFICATION

*Carleton Unite* needs to be a responsive mobile app that’s compatible with both iOS and Android devices. It should be a well-designed system that’s secure, fast, and perfected to provide the best user interface possible.

It should feature a reliable database, and well-optimized queries for fast CRUD operations. It should also:

- have a secure user-authentication system. Users should be able to register and log-in securely.
- allow users to set profile pictures.
- allow users to view profiles of other users of Carleton Unite app.
- allow users to browse course “rooms” (course forums). Every course is a forum of its own.
- allow users to add a Course Room to “My Courses”.
- allow users to post questions/concerns as “topics”.
- allow users to answer to “topics” in a course room.
- allow users to subscribe to the courses that they want to follow.
- allow users to view the courses that they are following under “My Courses”.
- allow users to chat in real time.
- allow users to receive push notifications when a message is received.
ARCHITECTURE

It was not a long time ago when people had to have a computer to do basic tasks such as sending e-mails, surfing the internet, managing internet banking, etc. Mobile applications have significantly evolved over the past decade. The uses of mobile applications today can only be limited by imagination. It is a growing sector both in terms of technology and portability. It is no wonder that mobile applications have greatly reduced the need of owning a laptop or a desktop.

Mobile application development officially started off as only being native. Native app development refers to the mobile applications that are developed in a platform-specific manner, using the Software Development Kit (SDK) provided by the platform vendors. Native applications use native components of the device. For example, native Android applications are developed in Android Studio using JAVA, XML and Gradle. Native iOS applications are developed in Xcode using Swift or Objective-C++.

The Carleton Unite system follows a client-server architecture. It has a backend (database and application-server), and a front-end.

NPM
NPM is a package manager for Javascript. This system uses npm to install the required packages for the Carleton Unite project. It reads the package.json file to determine the required plugins/dependencies, and then installs them using a simple command. This project uses npm both in the backend and frontend to install the required packages/modules.

FIREBASE PUSH NOTIFICATIONS
Push notifications are implemented in the system using Google FireBase. FireBase acts as a push-notification server and provides an API to handle push notifications in the front-end.
Although native apps are greatly optimized for their respective platform devices, yet it is no mystery that native app development can be a very slow and expensive process for an organization that needs to develop a mobile application for multiple platforms in a short period of time. To solve this issue, frameworks like Ionic, React, Xamarin, etc., were developed to enable developers to build cross-platform mobile apps called hybrid apps, using the already known major languages such as Javascript/Typescript, HTML5, Angular, XML, etc. Hybrid apps use the native framework’s WebView component to render the software as a web-app, instead of using the native components of the app.

Carleton Unite is a hybrid mobile application built using the Ionic 2 framework. Ionic 2 is an open-source framework that uses Apache Cordova, Typescript, HTML5, and Angular 4 to generate a mobile application that uses the platform’s native WebView component to render the app. It also provides support for a lot of native components of the device, for example, Camera, GPS, Accelerometer, etc. It has a large support community which enables developers to get help with their applications. Ionic currently supports three mobile platforms, namely, Android, iOS, and Windows. It also lets developers compile the project as a browser app but ionic does not provide official support for native components of browsers.

Carleton Unite has been developed for iOS and Android, and it uses the Carleton University’s logo as its logo [2]. With the help of Apache Cordova, Ionic 2 provides the capability of compiling the Carleton Unite project as an Android, as well as, an iOS app. The project follows design patterns such as Object-Oriented, Model-View-Controller (MVC), Observer, etc.
PARSE JAVASCRIPT API
Parse is a Backend as a Service (BaaS) providing company which was acquired by Facebook in 2013. It provides pre-built functions for various general operations for a backend, and lets developers focus more on the front-end and enhance the user experience [3]. Carleton Unite uses Parse Javascript API in the frontend to communicate with the backend and perform the required operations. Parse Javascript API provides all kinds of CRUD functions, as well as other functionalities such as User Authentication.

BACKEND

NODE.js
Node.js is an open source server framework that uses JavaScript. It uses asynchronous programming, thereby supporting simultaneous requests unlike frameworks like php [4]. Carleton Unite uses a Node.js server in the backend. This server is configured to connect to the MongoDB database, and initialize Express.js and Parse-Server.

EXPRESS.js
Express.js is a Node.js framework that acts as a middleware for this system. It helps manage everything required for the communication of the frontend with the backend, for example, routes, HTTP requests, parsing payloads, etc. It is used to avoid using the HTTP module of Node JS to configure everything from scratch.

PARSE-SERVER
As mentioned above, Parse is a BaaS solution that provides pre-built functions for various general operations for a backend, and lets developers focus more on the frontend and enhance the user experience. It is installed using npm, and configured for initialization in the server.js file, which is run using Node.js. Parse-Dashboard package is also configured and initialized in the main server.js file. Parse-Dashboard provides a user-friendly access to the MongoDB database, and other Parse services. Parse-Server also configures and initializes the push-notifications service. It provides functions for the front-end to perform CRUD operations to the database. It also provides functions for secure user authentication.
MONGO DB

This system uses MongoDB as the database. MongoDB is an object-oriented database. It is a reliable database that features dynamic schema modification, which means that one does not need to predefine a schema for the database. Applications can create or update the database tables dynamically. Although dynamic schema creation is a very strong feature that allows for faster development, yet it can be highly risky. For the sake of this project, MongoDB was a good choice because of the short period of time that was available to develop a multi-platform mobile application of this scale. It was also a good choice because in the beginning of the development, the system design idea was not concrete. I knew that I might need to redefine the database schema at any time during the development. The URL to the MongoDB server is specified in the server.js file under Parse configuration, to let Parse know where its database is sitting, and hence enable it to make a connection to the database and perform the necessary operations.
IMPLEMENTATION

This section describes the implementation of the entire system. It walks you through the project and database structure, as well as some key parts of the code.

BACK-END

DATABASE STRUCTURE

The Carleton Unite system uses an object-oriented database. The database can be visually understood by looking at the database tables (Classes) using the Parse-Dashboard. As it is an object-oriented database, we will refer to the tables as Classes, table rows as Objects, and table columns as Attributes.

Let us walk through some of the Classes in detail.

User

As the name describes, this class holds information of a User. Following is the description of its attributes:

- **objectId (String)**: It is a unique identifier for an object. It can be used for referencing data amongst Classes, as well as to fetch an object.

- **profilePic (String)**: It stores the Base64 encoded string of the picture that the user sets as its profile picture using the mobile app.

- **name (String)**: It stores the name of the user as a string.

- **username (String)**: It stores the user’s email address as a string. The mobile app asks the user for its e-mail address and stores it as the username to use it for user authentication. It also stores the same string in the email attribute of the User class.

- **email (String)**: It stores the user’s email address as a string.
• **major (String)**: It stores the user’s current University Major as a string.

• **password (String)**: It stores the user’s password which is encrypted by Parse.

• **topics (Relation <Topic>)**: It stores a list of references to every topic that the user has posted in any Course room (forum). This list is stored as a Relation that holds references to Topic objects.

• **courses (Relation <Course>)**: It stores a list of references to every course that the user has added to “My Courses” using the app. This list is stored as a Relation that holds references to Course objects.

• **createdAt (Date)**: It holds the date-time stamp for when the object was created.

• **updatedAt (Date)**: It holds the date-time stamp for when the object was last updated.

**Chat**

The *Chat* class is basically a message thread. It holds references to messages between users. Following image shows a list of various ongoing chats of a *User* in the mobile app:

*Figure 1 User Chats - Mobile App*
Following is the description of its attributes:

- **objectId (String)**: It is a unique identifier for an object. It can be used for referencing data amongst Classes, as well as to fetch an object.

- **messages (Relation <Message>)**: It stores a list of references to every message that belongs to this Chat. This list is stored as a Relation that holds references to Message objects.

- **lastMessage (Pointer <Message>)**: It stores a reference to the most recent message that belongs to this chat. This reference is needed to let the app show the most recent message in every Chat as it can be seen in Figure 2.

- **userCount (Number)**: It stores the number of users that belong to this Chat. At present, the mobile app is designed to support chat for only 2 users. However, in future, this attribute can help to extend the app’s chat functionality to allow for multiple users.

- **users (Relation <User>)**: It stores a list of references to every user that belongs to this Chat. This list is stored as a Relation that holds references to User objects.

- **badgeCount (Number)**: Badges can be used in the app to notify the user that something has changed. The badgeCount attribute is not currently being used in the app, but it can be used in the future to support badges.

- **createdAt (Date)**: It holds the date-time stamp for when the object was created.

- **updatedAt (Date)**: It holds the date-time stamp for when the object was last updated.
**Message**

As the name describes, a *Message* object holds information of a message when it is sent from one user to another. A *Message* cannot exist independently. It always has a parent *Chat*. The *Carleton Unite* mobile app lets the user see all his messages to a given user by clicking on the corresponding *Chat*. Following image shows a list of messages between two users.

![Figure 2 Messages - Mobile app](image)

Following is the description of its attributes:

- *objectId* (*String*): It is a unique identifier for an object. It can be used for referencing data amongst *Classes*, as well as to fetch an object.
- *from* (*Pointer* `<User> *User*>): It stores a reference to the *User* who created the message.
• to (Pointer <User>): It stores a reference to the User who is the recipient of the message.

• text (String): It stores the message content as a string.

• chat (Number): It stores a reference to the Chat that this message belongs to.

• users (Relation <User>): It stores a list of references to every user that belongs to this Chat. This list is stored as a Relation that holds references to User objects.

• seen (Boolean): This attribute is not yet being used in the system. However, it can be used in the future to mark messages as seen.

• createdAt (Date): It holds the date-time stamp for when the object was created.

• updatedAt (Date): It holds the date-time stamp for when the object was last updated.
NODE.JS SERVER

The Node.js server is a JavaScript file that initializes and configures Parse, Express, and HttpServer. Following are the contents of the Carleton Unite server file (/carletonUniteServer/server.js):

```
// initialize parse server settings
var app = new ParseServer({
  databaseURL: 'mongodb://localhost:27017/carletonunite',
  appId: 'carletonunite2250',
  'masterKey': 'masterKey',
  serverURL: 'http://localhost:8080/carletonuniteserver',
  liveQuery: {
    classes: ['Message']
  },
  'push': {
    'android': {
      'senderId': '7541608812785',
      'apiKey': 'Afiz5yu49U3kWiuNuCc7C889v8379vZ880_F8ytA'
    }
  }
});

// initialize parse dashboard settings
var parseDashboardSettings = {
  'app': [{
    'serverURL': 'http://localhost:8080/carletonuniteserver',
    'appId': 'carletonunite2250',
    'masterKey': 'masterKey',
    'appName': 'Carleton Unite'
  }],
  'user': [{
    'user': 'zahat',
    'pass': 'root',
    'masterKey': 'masterKey',
    'apps': [{
      'appId': 'carletonunite2250'
    }]
  }]
};

// initialize parse dashboard
var dashboard = new ParseDashboard(parseDashboardSettings, allowInsecureHTTP);

// initialize express
var app = express();
// make the parse server available at /carletonuniteServer
app.use('/carletonuniteServer', app, function(req, res, next){
  return next();
});
// make the parse dashboard available at /dashboard
app.use('/dashboard', dashboard);

// initialize http server
var HttpServer = require('http').createServer(app);
HttpServer.listen(8080); // listen to connections on port 8080

// initialize Parse Live Query Server
ParseServer.createLiveQueryServer(HttpServer);
```

Figure 3 Node.js server file (server.js)
As it can be observed, the server.js file initializes Parse (and all its services), HttpServer, and Express.

**Parse-Server Settings**

- *databaseUri*: The URI of the database that Parse should connect to.
- *appId*: An identifier for the server app.
- *masterKey*: Master key for authentication.
- *serverURL*: URL where the parse server will run.
- *liveQuery*: Parse LiveQuery is used in Carleton Unite to facilitate the live chat feature. LiveQuery server uses sockets to establish connection between a client and itself.
  - *classNames*: The classes in the database that LiveQuery will be used for. In Carleton Unite, it is only being used for Message class, in order to facilitate live chat.
- *push*: Settings for push notifications.

**Parse-Dashboard Settings**

- *apps*: The Parse app(s) that the dashboard should track. The Carleton Unite Parse Server values are specified here to let it be tracked by the Parse Dashboard.
- *users*: Users who have access to the Parse Dashboard.

**Express**

Express is configured to set endpoints for the parse-server and parse-dashboard.

**Http Server**

The Http Server is configured to host express at port 8020.

**Parse Live Query Server**

Parse Live Query Server is configured to use the Http Server for setting up Parse Live Query.
The front-end is an Ionic-2 project.

**PROJECT STRUCTURE**

The image below gives an overview of the project’s directory structure.

*Figure 4 Ionic project directory structure*

As it can be observed in Figure 4, the app directory is the root of the application. It consists of files which declare the root view of the app, initialize Parse, and initialize the global scss styling file of the app: app.scss. Figure 4 shows the contents of the app.component.ts file. This file is the root class of the application. This is where the
connection to the backend parse server is established by initializing Parse, and the root view of the application is set.

The image below shows the contents of the file app.module.ts. This file is an NgModule (a module imported from angular) which declares all the other modules to be used in the application. As it can be observed, every page and provider is declared in this module.

Figure 5 App root module: app.module.ts
**Component**

A *Component* is an angular *Directive* that defines the used change detection strategy.

A *TypeScript* class is set as a controller to an HTML view using *Component’s templateUrl* property. An *scss* file is set as a styling file for the HTML view using *Component’s selector* property.

**Page**

Every page in the *Ionic-2* project is a directory that consists of three files.

- *html file*: An HTML5 file where the view of a page is defined.
- *scss file*: A styling file for the view.
- *ts file*: A TypeScript file which acts as the controller for the view. All the business logic goes in this file.

For a deeper understanding, let us take an example of the *My Courses* page of the *Carleton Unite* mobile app. This page is where a user can view all the *Courses* that he is following (added to “My Courses”).

**My-Courses View: Mobile App**

*Figure 6* shows the *My Courses* page in the *Carleton Unite* mobile app.
My-Courses HTML5 (my-courses.html)

The image below shows the contents of the my-courses.html file. This is an HTML5 file for the My Courses view. It can be observed that angular bindings are also used in the HTML5 file. For example: *ngFor directive is used to iterate through the courses array that’s defined in this view’s controller (my-courses.ts). {{course.code}} is used as a binding for the code of the course that belongs to the current iteration of the *ngFor directive.

Figure 8 My Courses view: my-courses.html
**My-Courses controller (my-courses.ts)**

The *my-courses.ts* file is a *TypeScript* class that acts as a controller for the *My-Courses* view. The following image shows a part of the *my-courses.ts* file.

![Image of my-courses.ts file]

As it can be observed in the above image, the *MyCoursesPage* class is declared as an angular *Component*, and the styling and view files of the *My-Courses* page is attached to this *Component* using the properties *selector* and *templateUrl*. 
SOME IMPORTANT CODE SNIPPETS

PARSE API

Initialization

```typescript
Parse.initialize('carletonunite2296');
Parse.serverURL = 'http://192.168.0.14:8021/carletonuniteserver';
```

**Figure 10** Parse initialization (/src/app/app.component.ts)

The code snippet above shows how the ionic app initializes Parse with the Carleton Unite parse server’s appId, and serverURL. This allows the app to establish a connection with the parse server in the back-end, and thereby enable to app to use the Parse Javascript API to make CRUD operations to the MongoDB database.

Using Parse API to fetch data

```typescript
// Fetch courses that this user is following, using Parse.
getCourses(refresher) {
  var me = this;
  // Get currently logged in user
  var user = Parse.User.current();
  // Get "courses" relation from the User class
  var relation = user.relation("courses");

  // Fetch all courses from the obtained relation
  relation.query().find(
    success function (results) {
      // results contains the courses followed by the current user.
      var jsonArray = [];
      for (var i = 0; i < results.length; i++) {
        jsonArray.push(results[i].toJSON());
      }
      console.log(jsonArray);

      // Update the courses array of this class. This ultimately updates the view since
      // the view has a courses list that is bound to this array.
      me.courses = jsonArray;

      // If this function was called as result of the refresher component being pulled down,
      // dismiss the refresher to let the user know that the data has been fetched/refreshed.
      if (refresher != null) {
        refresher.complete();
      }
    },
    function (object, error) {
      console.log("Error getting my-courses: ")
      console.log(error)
      alert("Oops! Could not fetch courses. Please try again later.");
    }
  );
}
```

**Figure 11** Get courses followed by current user (/src/pages/my-courses/my-courses.ts)
The code snippet above shows how the Parse Javascript API can be used to query database and fetch the required data. In this case, Parse.User.current() returns the currently logged in User. Since user is a ParseObject, the courses relation can be obtained from it. The courses relation contains a reference to all the courses that this user follows. The query() function can be called on a Relation, and it can be chained with the find() function which is called on a ParseQuery.

The response from a Parse API call can either be a success response, or an error. To handle both cases, two callback functions are passed as parameters to the find() function. One callback handles the success response, and the other callback handles the error response. In this case, when a successful response is received, the controller converts the response-results to JSON, to allow for angular binding in the view. The courses array is updated, and hence the view is updated as well using angular binding.

PARSE LIVE QUERY

Subscribing to incoming message using Parse Live Query

```
// Initialize a subscription to listen to incoming messages using Parse Live Query
liveQueryMessages() {
  var me = this;

  me.current_user = Parse.User.current();
  this.query = new Parse.Query("Message");

  this.query.include("from");
  this.query.include("chat");
  this.query.equal(to, me.current_user);

  return this.query.find({
    success: function (results) {
      // Subscribe to the Message query where the receiver of the message is the current user
      me.subscription = me.query.subscribe("this-query");
      me.subscription.on('create', (message) => {
        // Publish toast event to display the received message as a toast
        me.events.publish('toast-event', {
          message: message.get("from").get("name") + ": " + message.get("text"),
          timer: 5000,
          position: 'top'
        });
      }, error: function (results, error) {
        console.log(error);
        alert(error)
      }
    }
  });
}
```

*Figure 12 Subscribing to Parse Live Query for incoming message* (/src/pages/browse/browse.ts)
The above code snippet demonstrates how the app subscribes to the query of fetching a message, the recipient of which is the currently logged in User, using ParseLiveQuery. A subscription is made to this query, and the predefined on('create', callback) function of the subscription is used to listen to the received result and perform the required task, which in this case is to display a toast message showing who the message is from, and the textual content of the message.

**EVENT SUBSCRIPTION**

**Subscribing to an event (/src/app/app.component.ts)**

```
events.subscribe('toast:event', (data) => {
  // Create Toast
  let toast = this.toastCtrl.create(
    { message: data.message, 
      duration: data.timer, 
      position: data.position
  });

  // This is fired when toast dismisses
  toast.onDidDismiss() => {
    console.log('Dismissed toast');
  });

  // Display Toast
  toast.present();
});
```

*Figure 13 Subscribing to an event (/src/app/app.component.ts)*

The above code snippet demonstrates the use of the Event-Subscription model that’s being used in the system, to subscribe to an event. In the displayed code snippet, the MyApp class subscribes to the toast:event. In other words, it registers itself as a listener to the toast:event. No matter when and where in the application the toast:event is published (triggered), the MyApp class will be notified of the event and it can therefore perform the desired task, which in this case is to create and display a toast message.
Publishing an event (/src/pages/browse/browse.ts)

```javascript
// Publish toast event to display the received message as a toast
events.publish("toast:toast", { message: message.get("from").get("name") + ": " + message.get("text"), timer: 5000, position: "top" });
```

Figure 14 Publishing an event (/src/pages/browse/browse.ts)

The above code snippet demonstrates the use of the *Event-Subscription* model that’s being used in the system, to publish (trigger) an event. In the displayed code snippet, the `toast:event` is published (triggered) when a `message` is received. When this event is published, all the classes that are subscribed to this event get notified, and can therefore perform the required task, which in this case is to create and display a toast message containing the sender’s name and the textual `message` content.

**SETUP AND LAUNCH INSTRUCTIONS**

Follow these instructions to setup the backend and front-end project, and successfully launch the *Carleton Unite* application on an Android or iOS device. Alternatively, the *Carleton Unite* mobile application can also be launched in a browser for testing. However, it should be noted that when the application runs in a browser, the *native* plugins such as *Camera, Push-Notifications*, etc. do not work.

**BACK-END**

Make sure you have access to the `/carletonUniteServer` directory. Then, follow the instructions below.

1. Install *MongoDB*.
2. Run `mongod.exe` executable file to launch the *MongoDB* database. By default, *MongoDB* is configured to listen to connections at port 27017.
3. Install *Node*.
4. Open terminal and `cd` into the directory `/carletonUniteServer`.
5. Type the command `npm install` and press Enter. This command asks NPM to install all the packages listed in the `package.json` file.

6. Open the file server file: `/carletonUniteServer/server.js`. Edit the file to configure the settings as needed. The most important part here is to make sure that the URL’s provided for MongoDB, Parse-Server and Parse-Dashboard are appropriate.

7. Open the terminal and make sure that the root directory is `/carletonUniteServer`. Type the command `node server.js` and press Enter. This command starts running the Node.js server.

8. Open a browser and navigate to `http://localhost:8020/dashboard` (your URL might be different - check `server.js`.) to access the Parse Dashboard. This is where you can view and modify the MongoDB database.

FRONT-END

Make sure you have access to the `/CarletonUnite` directory. Then, follow the instructions below.

1. Make sure you have Node installed.

2. Open terminal and `cd` into the `/CarletonUnite` directory. Type the command `npm install` and press Enter. This command will install all the required packages listed in the `package.json` file.

3. While being in the `/CarletonUnite` directory in the terminal, type the command `npm install -g ionic cordova`. This command will install ionic and cordova globally on your system.

4. Open the file `/CarletonUnite/src/app/app.component.ts` and make sure that the Parse Server URL is so as to match the URL where your Parse Server is running at.

5. Open terminal and `cd` into the directory `/CarletonUnite`. To run the app in
   - **Browser (Test environment)**: Type the command `ionic serve` and press Enter. A web page will open to display the app once the build finishes.
Typically, the app runs at the URL http://localhost:8100 but this might be different depending on the ports available on your system. Check the terminal to find out the URL at which the app server is running at.

- **Browser (Native):** Type the command `ionic cordova run browser` and press Enter. If you receive an error saying that the platform is not available, then add the platform using the command `ionic cordova platform add browser` and press Enter.

- **Android:** To build and run the Android app, your will need to have the Android SDK and JAVA JDK installed. Type the command `ionic cordova run android` and press Enter. If you receive an error saying that the platform is not available, then add the platform using the command `ionic cordova platform add android` and press Enter.

- **iOS:** To build and run the iOS app, your system will have to be a Mac OS, and have XCode installed. Type the command `ionic cordova run iOS` and press Enter. If you receive an error saying that the platform is not available, then add the platform using the command `ionic cordova platform add iOS` and press Enter.
RESULTS AND EVALUATION

The resulting system is a beautiful, well-optimized, and highly-functional mobile application for Android and iOS that can help students of Carleton University to stay connected with their courses and fellow classmates, and ease their journey through the university.

The system architecture used to accomplish this mobile application’s purpose has been a success, and therefore it has proved to be a good choice for this project. Given the available time for this project, the amount of work, and the available manpower, choosing this architecture was definitely a good decision.

The functionality of the mobile application was demonstrated to Professor Louis Nel, who appreciated it very well. All the application features that were initially proposed in the formal proposal were developed and built into the app successfully.

The majority of app’s features were manually tested and concluded to be successful. However, there was not enough time for carrying out rigorous tests or writing test-cases. Nevertheless, I am confident that the application is completely functional in a real-world scenario and there is no major bug that exists in the system, that could perhaps lead to a system failure.

Following manual tests were carried out:

- **User Registration**: User could successfully sign up with a valid email address. User could not sign up with an invalid e-mail address.
- **User Authentication**: User could not sign in with an invalid email/password combination.
- **View Courses**: User could successfully view all the courses in the Browse section of the app.
- **View Users**: User could successfully view other users in the system, in the Browse section of the app.
• Follow Courses: User could successfully add a course to his My-Courses list. User could not follow a course (add to “My Courses”) if it was already being followed by him.

• Post Topics: User could successfully post a topic in any Course Room.

• View User Profiles: User could successfully view profile of other users.

• Messaging: User could not create a new Chat with a user if a Chat between the two users already existed.

• In-App Message Notification: When in the app, user could successfully receive a notification as a Toast, for an incoming message.

• In-App Message Notification Data: When in the app, user could successfully see the sender’s name and the message content in the notification Toast for an incoming message.

• User Profile Picture: User could successfully set a profile picture by taking a picture using the device’s Camera, or by choosing a picture from the device’s photo gallery.

• Push Notifications: When the app was not in focus, the user could successfully receive a Push Notification for an incoming message.

The only thing that could not be accomplished in this project is to display badges when a message-notification was received. The database is ready to support badges, however the front-end does not have any implementation for it. I could not get this done due to time constraints. However, it is a major component for future work on this application.
FUTURE WORK

There is a lot that can be done to improvise on the Carleton Unite system. Some of the work that can be done is as follows:

FEATURE ADDITIONS

Some of the features that can be added to the mobile application are:

- **Badges**: The mobile application can be updated to support badges for notifications. The mobile application's Chats tab can display a badge count every time the logged in User receives a new message.
- **Group Chat**: The mobile application can be updated to support group chat. A group chat will allow multiple users to chat in real-time using the Carleton Unite app.
- **Message Seen**: The chat feature of the mobile app can be updated to let users know when their message has been seen/read by the recipient of the message.
- **File Sharing**: The chat feature of the mobile app can be updated to support sharing of files between users. An example use case of this feature would be to enable students to share lecture notes, past exams, etc.
- **View User Activities**: The mobile application can be updated to allow users to see other users' activities in their Profiles. Activities could include user tasks like following a course, posting a topic in a Course Room, posting an answer for a topic, updating profile picture, etc.
- **Like Posts**: The mobile application can be updated to allow Users to like posts by other users.
- **Add Friends**: The mobile application can be updated to allow Users to add other Users as friends.
- **Notifications for Course Room activities**: The mobile application can be updated to send notifications to users when an activity takes place in a Course Room that
they are following. This can allow users to always stay up to date with their courses.

- Generalized University Social Network: This can be a major update to the entire system as it can change the Carleton Unite system from just supporting Carleton students to supporting all universities across the globe. It can be converted into a standardized social network for universities.

- Performance Upgrade: The performance of the Carleton Unite mobile application can be improved for iOS by installing the WKWebView plugin, and for Android by installing the Crosswalk plugin.

**CODE CLEANUP**

- Create helper classes in the front-end project to provide common functions and/or constants which can be used anywhere in the application.

- Use Parse Cloud functions to define CRUD functions as API endpoints in the backend, rather than performing them from the front-end. The front-end can be updated to call the Cloud functions for desired CRUD operations. The benefit of doing this would be that the heavy-weight processing would be performed on the server instead of the mobile client. Mobile devices have limited hardware, whereas the backend server can have a strong hardware and hence improve the performance of the mobile application overall.

- Create custom providers in the Ionic project for exclusive services. For example, a provider could be created for handling all Parse Cloud function calls. This would better organize the code, remove duplication and coupling, and make it easier to read, understand and modify.

**TESTING AND RELEASING**

Unit tests can be written for better testing and improving reliability and robustness of the system. Once sufficient tests pass, the app can be published to the Apple App Store and Google Play Store.
CONCLUSION

The resulting mobile application aims to bring the students of Carleton University closer together, and help them ease their journey through the university. The highly user-friendly interface and feature-packed system of the Carleton Unite mobile app will surely provide a great experience to its users.

This project has helped me learn a wonderful stack of technologies that are widely used in the industry today. The end-product is not only an utterly useful social networking tool for students, but it is also a resource for students to learn and recognize the benefits of Hybrid App Development. This type of development brings together the two major components of front-end today, namely websites and mobile applications. It bridges the gap between web technologies and native mobile technologies. It is a wonderful framework that allows inexpensive development of highly useful, efficient, and beautiful mobile applications in short time frames.

I highly recommend educational institutions to include such frameworks in their curriculum to not only teach web technologies, but also introduce students to the world of mobile application development at the same time. I would like to conclude this report by mentioning that the intention of this project was successfully met, and the resulting mobile application was successfully demonstrated to this project’s supervisor Professor Louis Nel. I hope this report helped its readers get a good sense of the current Carleton Unite system and its future potential.
REFERENCES

1. https://www.udemy.com


4. https://www.w3schools.com/nodejs/nodejs_intro.asp