COMPREHENSIVE FOOD ORDERING SYSTEM IN AUI

Mouad Lasri

Supervised By

Dr. Nasser Assem
Student Statement:

I, Mouad Lasri, do affirm that I have applied ethics to the design process and in the selection of the final proposed design. And that, I have held the safety of the public to be paramount and have addressed this in the presented design wherever may be applicable.

Signed May 5th, 2020

Mouad Lasri

Approved by the Supervisor(s)

Nasser Assem

Dr. Nasser Assem
ACKNOWLEDGEMENTS

I would like to thank Dr. Nasser Assem for agreeing to supervise my project. With his guidance, I am applying the skills that I have learned during my journey at AUI, and most importantly, broaden my knowledge in software development and design.

Thanks to Dr. Nasser Assem’s continuous advice and precious help, I am able to fulfill the requirements of my capstone project which represent an important steppingstone into my journey of becoming a computer scientist.

Finally, exceptional recognition goes to my family and friends for their generous emotional support. They showed a considerable support, understanding, and patience throughout my experience with this capstone project.
Table of Contents

ACKNOWLEDGEMENTS .................................................................................................................3
ABSTRACT .........................................................................................................................................7
1. INTRODUCTION .............................................................................................................................8
2. STEEPLE Analysis ............................................................................................................................9
   2.1 Social .........................................................................................................................................9
   2.2 Technologically ..........................................................................................................................9
   2.3 Economical ..............................................................................................................................10
   2.4 Environmental ..........................................................................................................................10
   2.5 Political ......................................................................................................................................10
   2.6 Legal ..........................................................................................................................................11
   2.7 Ethical ........................................................................................................................................11
3. Feasibility: ........................................................................................................................................11
   3.1 Financial Feasibility: .................................................................................................................11
   3.2 Technical Feasibility: .................................................................................................................11
   3.3 Resource Feasibility: .................................................................................................................12
   3.4 Risks: ..........................................................................................................................................12
4. Requirements Specifications: ........................................................................................................12
   4.1 Functional Requirements: ........................................................................................................12
      4.1.1 User Requirements (Students)............................................................................................12
      4.1.2 User Requirements (Restaurant)........................................................................................13
      4.1.3 Non-Functional Requirements .........................................................................................14
      4.1.4 Mobile Application’s Use Case Diagram ..........................................................................15
      4.1.4 Web Application’s Use Case Diagram ..............................................................................16
      4.1.5 Sequence Diagram ...........................................................................................................16
5. System Architecture ......................................................................................................................18
6. Methodology: ...................................................................................................................................22
7. Design: ............................................................................................................................................24
   7.1 Database Entity Relationship Diagram .....................................................................................24
Table of Figures:

Figure 1: Mobile App Use Case Diagram .................................................................15
Figure 2: Web Application Use Case Diagram ......................................................16
Figure 3: POST Request Sequence Diagram .......................................................17
Figure 4: GET Request Sequence Diagram .........................................................17
Figure 5: RESTful Architecture .............................................................................18
Figure 6: REST API Methods .................................................................................19
Figure 7: HTTP Request Anatomy .........................................................................19
Figure 8: Overall System Architecture ..................................................................20
Figure 9: Entity Framework Workflow ..................................................................20
Figure 10: Entity Framework DB Context ..............................................................22
Figure 11: Incremental Model ................................................................................23
Figure 12: Entity Relationship Diagram ...............................................................24
Figure 13: Class Diagram .......................................................................................25
ABSTRACT

Restaurants at AUI suffer from long queues of students waiting in line to order their food with an outdated system in place. This project is an attempt to reduce the congestion of students in on-campus restaurants, and their waiting time. The system offers a data analytics platform that will help both restaurants and students. Restaurants will be able to track their orders and have access to insightful information about the state of their activities. And students will be able to check when is the best time to order their food and can do so directly from their mobile phones. The project presented in a comprehensive system that is made of a web application for restaurants, and a mobile application for students using the latest technologies as well as following the most recent standards and protocols. Both applications will be interacting with each other to offer a smooth user experience.

*Keyword: mobile application, analytics, track, orders, interaction, user experience, web application, congestion, system.*
1. INTRODUCTION

This capstone project is a comprehensive data analytics system that aims to reduce the congestion of students in campus restaurants and reduce waiting time by checking in real time the best time to order food on their phones.

Also, the system optimizes the restaurants’ inventory, time management and order-handling by providing insightful and detailed analytics through a web interface dashboard that displays the food item ordered, time of order, time of order completion, and overall performance of the menu items.

The motivation behind this project was to get rid of the inefficiency of using old order-recording equipment that slowed down the waiting queue and mismanagement of food inventory, resulting in shortage of highly demanded menu items due to poor forecasting. Since the system is transaction-heavy, I opted for Microsoft Azure SQL database and use the power of cloud computing to process the heavy load of requests coming in.

As for visualization tools I used, I explored the different open-source JavaScript libraries that are easily compatible and portable on React. For the website, I opted for the Chart.js library, which has over 48,000 stars on GitHub and one of the most used JavaScript libraries for visualization. As for the mobile application, I opted for the “react-native-chart-kit” library which offer a complete panoply of useful charts that fit my needs.

As a result, the restaurants have access to high quality and insightful visualization which enable them to improve their services by assessing the students’ needs and activities. In addition to that, the system was developed with a long-term mindset and maintainability in mind. As such, I built an API that will serve as an intermediary layer between the system’s applications and the database with the objective of keeping a consistent access to it, and to avoid changing the internal code of each application (web, mobile, etc.) in case of a change in the database.

Moreover, the API will allow interaction with the database for any student who wishes to develop his own application or service and needs the users’ and restaurants’ data.
2. STEEPLE Analysis

2.1 Social

This solution mainly targets the social aspect of the issue faced. This application will save a lot of time to students, and reduce the congestion times in the restaurants. By a simple click on their mobile phones, students will be able to make their orders instantly from their favourite restaurants and get notified whenever their order is ready, thus reducing any in-place waiting time. Moreover, they can keep track of their orders and make any modifications they want to the order without any waiting time or delay. They can also keep track of the total amount they spent in each restaurant. From the restaurants point of view, they will have access to a data analytics dashboard to keep track of the current, completed and in-coming orders. The dashboard will provide them with useful analytics that they use to improve their services and make targeted decisions like knowing what the busiest times are, popular menu items etc…

2.2 Technologically

This project uses state of the art technology to provide the best functionalities and security level. The system is composed of five main components. The first one is the mobile application that the students will use to make their orders. The mobile application is made with React Native, a framework developed by Facebook that allows to develop mobile applications using React. The framework is the fastest in the market and is known for its ease-of-use and fast development. The second part of the system consists of the web application that restaurants in AUI will use. It is made by React. The biggest advantage of React is the fast that it is a SPA (Single Page Application) framework, which me there is only one HTML face being sent over the server and rendered. Therefore, time spent communication with the server is reduce significantly which result in a fast and responsive website. The back-end (also called “server-side”), will be made using Nodejs, the hottest JavaScript server-side framework at the moment. This way, both the front-end and the back-end will use the same language saving a considerable amount of development time. The fourth part is the API that will act as an intermediary between the mobile and web applications, and the database. The API will follow the RESTFUL architecture as it is the most used protocol nowadays and has the advantage of completely separating the user
interface (front-end) from the server (back-end). The API will be with ASP.Net Core API framework which proves resilient and secure.

The project is composed of four main parts. The first one being the backend (also called “server-side”) of the application. It refers to the actual operations done in the server side which are all the features that the application presents. We decided to follow the REST software architecture. Besides the fact that it is the most used protocol nowadays, RESTFUL APIs’ advantage lays in the fact that it totally separates the user interface from the server’s. Using an API in such a system allows for long-term maintainability and third-party usage of the data generated by the system. The API will communicate with the client-side using HTTP requests such as GET, POST, PUT, DELETE…

To test the API, the most popular API tester tool is Postman. Postman allows testing API on a local server in a fast, easy and straight-forward manner. All HTTP requests can be made through Postman with a simple click and through an intuitive interface.

The fifth component is the database. Since the system is transaction heavy, I will be using an SQL server. To take advantage of cloud computing capabilities which is what is used nowadays, I will be using Azure SQL Database to be more specific which is a very powerful and secure database service offered by Microsoft.

2.3 Economical

On the long term, the system will save both money and time for students. The application is free to use, and the costs of maintenance and hosting are not big. The students will not be affected at all economically, and the restaurants will incur very small expenses due to the cloud infrastructure needed to run the applications.

2.4 Environmental

The system does not have any repercussions and effects on the environment. Physical hardware is not needed as everyone is performed on the cloud and online. However, as the project goes, further research will be done in this regard.

2.5 Political

The system does not have any political purpose and therefore it has no impact whatsoever on the political ground. Also, the system will never be used for political reasons in the future.
2.6 Legal
Since the system is for AUI students and AUI restaurants, it will above by AUI rules and will follow a ToC (terms & conditions) made by AUI to ensure and regulate the use of the system by both parties. Failure to comply with Al Akhawayn’s regulation can result in fines, or legal actions; hence, it is primordial to notify the university for any important design change. Additionally, we will be exclusively using open source frameworks, libraries, tools and packages,

2.7 Ethical
From an ethical perspective, no system is free of malicious intent. Therefore, it is our responsibility to make the system as secure as possible to avoid any data and personal information leak. It is my duty as a student to collaborate closely with the restaurants and AUI administration to ensure a bug-free and backdoor-free software using the most secure tools, frameworks and software especially since the system will hold sensitive and personal information about the students and the restaurants.

3. Feasibility:

3.1 Financial Feasibility:
Because this project is an online application software, there will be hosting costs associated to the Azure SQL Database server and the hosting the of the web application as well as the mobile application and the API. However, since everyone will be hosted on the cloud, the financial requirements will not be substantial as the cost will be proportional to the use of the different servers.

3.2 Technical Feasibility:
We decide to choose the hottest and latest technologies because they provide the best user experience, architecture, learning curve and results for our system. To that end, a very brief and quick description of the the technologies that will be used:
- HTML: Shall be used to generate to make the layout of the web application.
- CSS: Shall be used to style the web application and its different components.
- JavaScript: Shall be used to make API requests for the database and as base languages for the front-end as well as the back-end frameworks.

- ASP.Net Core: Shall be used to build the API application; i.e: develop the necessary APIs routes, endpoints and logic to communicate with the database as well as returning the right data for the right user.

- React-Native: Shall be used to develop the front-end for both Android and IOS applications.

- Azure SQL Server: Shall be used as our cloud database.

- Git: Shall be used for version control.

- Postman: Shall be used to test the API routes.

### 3.3 Resource Feasibility:

The system does not require a lot of processing or powerful hardware to build. Especially since I will be using cloud resources offered by Microsoft for the database requests which will reduce the computation and processing load on my laptop.

### 3.4 Risks:

The first and most important risk that we face developing this software is the limited time that we dispose of in order to learn new technologies, and maintain an acceptable academic performance in other classes. Another risk that we might face as software developers, is the fragility of the system. We plan on solving this issue by apply cutting edge security frameworks and follow the security norms and standards

### 4. Requirements Specifications:

#### 4.1 Functional Requirements:

##### 4.1.1 User Requirements (Students)

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>User Requirement</th>
<th>System Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The user shall be able to create an account</td>
<td>The user shall be able to create his own account by entering his personal information and a secure password</td>
</tr>
</tbody>
</table>
The user shall be able to log in to his account by entering his appropriate credentials.

The user shall be given the choice to order from the restaurant of his choices in the main menu of the application.

The user shall be able to choose a restaurant and menu items to make his order.

The user shall be able to add as many items as he wants by clicking on an item that will automatically be added to his cart.

The user shall be able to delete any item from his cart directly if he changes his mind on any of the items added.

The user shall be able to write a short feedback report detailing his comments and concerns on anything related to the restaurant services on the mobile application.

The user shall be able to give a unique rating to all the restaurants present in the application.

The user shall have access to his profile page where he can check the history of this orders.

The user shall be able to cancel the current order before the time limit.

The user shall be able to access a dashboard to check his own activity on the application.

The user shall be able to choose discount options to spend his points on in a simple and effective way.

### 4.1.2 User Requirements (Restaurant)
<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>User Requirement</th>
<th>System Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The restaurant shall be able to check their analytics</td>
<td>The restaurant shall be able to access detailed charts and information about their orders, users and activities</td>
</tr>
<tr>
<td>2</td>
<td>The restaurant shall be able to check their item menus</td>
<td>The restaurant shall be able to access a page that shows all the menu items they are offering for the students</td>
</tr>
<tr>
<td>3</td>
<td>The restaurant shall be able to change a menu item status</td>
<td>The restaurant shall be able to change a menu item status into unavailable if its stock has ran out in the inventory</td>
</tr>
<tr>
<td>4</td>
<td>The restaurant shall be able to see the orders of the students</td>
<td>The restaurant shall be able to have access to all of the orders made by students and change their status accordingly and filter them out by user id</td>
</tr>
<tr>
<td>5</td>
<td>The restaurant shall be able to check the feedback of the students</td>
<td>The restaurant shall be able to access all the feedbacks written by their students and filter them out by user id.</td>
</tr>
</tbody>
</table>

### 4.1.3 Non-Functional Requirements

The non-functional requirements of the web application and mobile application are the same as both software uses the same API application to make their API calls.

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>User Requirement</th>
<th>System Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The system shall be easy to navigate and easy to use</td>
<td>The system shall have an intuitive design to make it easier for the users to navigate through</td>
</tr>
<tr>
<td>2</td>
<td>The system shall not take a lot of memory space.</td>
<td>The system shall not exceed 5 megabytes of storage space required.</td>
</tr>
<tr>
<td>3</td>
<td>The system shall have a fast run-time</td>
<td>The system shall open in less than 3 seconds.</td>
</tr>
</tbody>
</table>
4.1.4 Mobile Application’s Use Case Diagram

The use case diagram represents the action an actor (or actors) take vis-à-vis the system they are using and interacting with (UML 2 Use Case Diagrams: An Agile Introduction, 2020).

![Mobile App Use Case Diagram](image)

*Figure 1: Mobile App Use Case Diagram*
4.1.4 Web Application’s Use Case Diagram

![Use Case Diagram](image)

*Figure 2: Web Application Use Case Diagram*

4.1.5 Sequence Diagram

Since both the web application and mobile application communicate solely with the API through HTTP requests, and then the API communicates with the database, the sequence diagrams for both applications is the same when it gets to the HTTP requests I used (mainly GET and POST requests):

**POST Request Sequence Diagram:**
GET Request Sequence Diagram:

Figure 3: POST Request Sequence Diagram

Figure 4: GET Request Sequence Diagram
5. System Architecture

The system’s overall architecture is different from a typical software engineering project in which we often have the back-end code baked with the front-end code, following an object-oriented approach. In the case of FoodEx, both the web application and mobile application that the restaurant and the students will respectively used are only a front-end that communicate with an API by sending HTTP requests, and getting back the data and the HTTP request status. To that end, I opted for the REST (Representational State Transfer) architecture. In a REST architecture, “the server does not store any state about the client session on the server side” (REST, n.d.). The advantage of a REST architecture is the fact that every HTTP request happens and works independently from each other, and it does not even store authentication information by using cookies and sessions. Instead, all the required data and request parameters are sent to the API in every single request.

The following figure shows a brief summary of the RESTful architecture in which the client (front-end) communicates with the API, then forwards its communication with the database, and then returns the data and status to the API in a backward fashion, to finally send it back to the front-end.

![RESTful Architecture](image)

Figure 5: RESTful Architecture

As of the HTTP methods used in above-mentioned architecture, there are four main methods that are used:
As for the HTTP request anatomy, it is made of 8 distinct elements:

**HTTP URL Anatomy**

```
1 2 3 4 5 6 7 8
https://www.example.com:3000/path/resource?id=123#section-id
```

**Key**

1. Scheme - defines how the resource will be obtained.
2. Subdomain - www is most common but not required.
3. Domain - unique value within its top-level domain.
4. Top-level Domain - hundreds of options now exist.
5. Port - if omitted HTTP will connect on port 80, HTTPS on 443.
6. Path - specify and perhaps find requested resource.
7. Query String - data passed to server-side software, if present.
8. Fragment Identifier - a specific place within an HTML document.
This is an overall design of the whole system and how each component interacts with another:

![Overall System Architecture](image)

**Figure 8: Overall System Architecture**

In order for the API to communicate with the database, and send back and forth data with compatible data types in C# format when it comes to API-Database communication and JSON format when it comes to API-Client communication, an Object-Relational Mapper need to be used. An ORM is a tool or software that is used to convert the different data types into compatible ones, thus allowing an easier coding flow and development. Therefore, it is possible to access the database directly from inside the API codebase, which is C# using ASP.Net Core in this case.

Entity Framework is a C# package that will act as my ORM for this project. It is easy to use, offers all the necessary functionalities that I need to connect with my database as well as returning the data back to the client-side in JSON format (Getting Started - EF Core, 2020).

![Entity Framework Workflow](image)

**Figure 9: Entity Framework Workflow**
As the above image shows, Entity Framework offers an ORM interface between the application (client-side), and the ADO.Net Provider which is a data provider that is used to connected to an SQL database and retrieve data using SQL queries. It is part of the .NET framework.

Moreover, the Entity Data Model (EDM) describes the object-relational mapping between the entities in the database, and their C# classes counterparts. There are three ways to do such mapping:

- The Database First approach consists of using an existing database (SQL for example) and Entity Framework will generate the corresponding Data Model in .edmx format. The .edmx file is simply an XML file that describe the mapping details and layout between the storage model and the conceptual model. The generated data model can be interacted with using Entity Framework Designer. This is the least used method.

- The Model First approach consists of using an existing Data Model (.edmx), and generate a database with its corresponding tables with it.

- The Code First approach is the most widely used one. The first method is the Code First which consists of building plain C# classes which will represent our data model, then use Entity Framework to automatically create a new database where the tables are mapped with the classes created in the data model part.
The last method, and is the one that I used, consists of using an existing database (SQL database in my case), and I generated the mapped data model (C# classes) that I use to interact with my database using C# code in ASP.NET Core.

In Entity Framework, each C# class represents an entity, and the database is represented with a Database Context. Therefore, the Database Context is used for querying, and the entities’ states get changed, as shown in the following image:

![Figure 10: Entity Framework DB Context](image)

6. Methodology:

“Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle. Incremental development is done in steps from analysis design, implementation, testing/verification, maintenance” (Incremental Model in SDLC: Use, Advantage & Disadvantage, Guru99).

Given the technologies I used and the approach I took, the incremental software development model makes the most sense. Since React uses Components (explained in details in the implementation and technology enablers sections), it is easier to design, develop and test independent features which make an iteration, before moving into the next feature.
Using the incremental software development model made the development of the system faster, and easy to make modifications on it.
7. Design:

7.1 Database Entity Relationship Diagram

A lot of time was dedicated to the database design because it constitutes the central piece of the whole system. A lot of data is being exchanged and therefore a good database system is required. Moreover, the database design has gone through many changes and iterations. The final database design iteration is the following:

![Entity Relationship Diagram](image)

Figure 12: Entity Relationship Diagram

7.2 Class Diagram

The following class diagram shows the different classes as well as the attributes and methods that constitute them. It combines the classes of the web and mobile applications since they are related and use commonly most of the ASP.Net Core API endpoints:
8. Implementation Details:

8.1 Application Programming Interface (Model and Controllers):

The most crucial and central piece of the system if the API. It is the layer between the database and the web & mobile applications. Therefore, all HTTP requests will go through the API in order to communicate and get data from the database. Thus, the architecture of the API needs to be on point. So far, the API was done using ASP.Net Core, an MVC framework made by Microsoft that offers a good development environment with the integration of Git and Visual Studio. ASP.Net Core also has a plethora of packages ready to use that make a lot of the coding easier and increase the abstractions of database communication. I am using Entity Framework as my ORM (Object-Relational Mapping) to map my C# models with my database tables which will allow me to make modifications to my database directly through the API using C# code and models. All the tables present in my database are modelled as C# class files with the same data types as the columns in Azure SQL Database.
Then, as mentioned earlier, ASP.NET Core is an MVC framework, which means the project is divided into Models and Controllers that communicate with each other, using C# code.

The following screenshot shows a snapshot of the DbContext, which is the instance of the SQL Database that allows us to interact with it and retrieve/update/delete data.

An example of the AppUser class that is mapped to the app_user table. We can see that all the SQL columns, primary keys and foreign keys are properly mapped using Entity Framework:
The following screenshots show the details of the configuration file which contains the different services and libraries attached to it, as well as a connection string to our database.

As for the API endpoints, here is an example that shows a GET HTTP request by a restaurant that shows their orders details and their respective order status (in-progress, completed, delivered):
A POST HTTP request that shows how an order is confirmed and added by a user in a restaurant:

```csharp
// POST: api/restaurants/order
// POST confirm the order of the user and add it to the database
[HttpPost("order")]
public IActionResult ConfirmOrder([FromBody] ConfirmOrder order)
{
    // To return a status code of 400, all three DB Inserts need to be successful
    // Thus the try/catch statement after each change save
    try
    {
        db = this.db;
    }
    catch
    {
        return BadRequest();
    }

    // First, create a new order with the user id in the MakeOrder table
    var newOrder = new MakeOrder
    {
        UserId = int.Parse(order.UserId)
    };
    db.Add(newOrder);
    try
    {
        db.SaveChanges();
    }
    catch
    {
        return BadRequest();
    }

    // Second, populate the others tables: app_order (1)
    var orderId = newOrder.OrderId;
    var appOrder = new AppOrder
    {
        OrderId = orderId,
        RestaurantId = int.Parse(order.RestaurantId),
        OrderStatus = 1
    };
    db.Add(appOrder);
    try
    {
        db.SaveChanges();
    }
    catch
    {
        return BadRequest();
    }

    // Third, populate the others tables: order_details (1) with individual items
    foreach (CartItem item in order.ItemsList)
    {
        // Create a new instance of MakeOrder object
        var orderDetails = new OrderDetails
        {
            OrderId = orderId,
            ItemId = item.ItemId,
            Quantity = item.Quantity
        };
        db.Add(orderDetails);
        try
        {
            db.SaveChanges();
        }
        catch
        {
            return BadRequest();
        }
    }

    return Ok(orderId);
}
```
First we connect to the SQL database through the Database Context. Then, in order to add a user order into the database, there are many tables that need to be updated (order_details, make_order, app_order...) and so we need to insert into those tables in a sequential way and catch any errors along the way. Then, we return an Ok() status which is a HTTP status of 200, meaning that the operation was successful.

8.2 Mobile Application (Students):
The first thing a student needs to do is log in with his account to access the application:

The code to handle the log in behaviour which is a POST request sent to the API:

```javascript
submitHandler = (e) => {
    e.preventDefault();
    console.log("CURRENT STATE s = ", this.state);

    // var responseStatus = await axios.post('https://localhost:44212/api/Users/', this.state);
    axios.post('https://87286160.ngrok.io/api/Users/', this.state).then(response => { // console.log('STATUS : ', response.status);
    console.log('User Found! Id : ', response.data.userId);
    // this.props.navigation.navigate('Home', {[connectedUserId]: response.userId});
    this.setState({
        authFlag: true,
        userId: response.data.userId
    });

    this.storeUserId(response.data.userId);
    // this.props.navigation.navigate('Home', {[connectedUserId]: response.data.userId});
    })
    .catch(error => {
    console.log('Error : ', error.response);
    this.setState({ errorMessage: 'Wrong Credentials! Please try again.' });
    });
```
Then, the authentication flag is updated to signal the user has been authenticated successfully and we also return the user id of the user.

Next, the user is prompted with the restaurant choices:

![Restaurant Choices](image)

When he clicks on a restaurant, he is then taken to the restaurant main page in which all the restaurant items are loaded. First, when the React component is mounted, we send a HTTP GET request to the API and setting the restaurant id as parameter in the request URL. Then, when the response is received, we extract the data from the HTTP response, loop over it, and construct a JavaScript object variable to make it easier to map through it and render it as JSX.
```javascript
async componentDidMount() {
  // loading fonts
  await Font.loadAsync({
    'poppins-regular': require('./assets/fonts/Poppins-Regular.ttf'),
    'poppins-bold': require('./assets/fonts/Poppins-ExtraBold.ttf'),
    'roboto-regular': require('./assets/fonts/Roboto-Regular.ttf'),
    'roboto-bold': require('./assets/fonts/Roboto-Black.ttf'),
    'pacifico': require('./assets/fonts/Pacifico-Regular.ttf')
  });

  axios.get('https://872b86160.ngrok.io/api/Restaurants/\{this.props.route.params.restaurantId\}/ItemCategories').then(response => {
    var results = [];

    // this loop constructs a dictionary based on the item category (ie: 'Paninis': [items])
    // the API does not return nested objects therefore the dictionary is built in client-side
    for (var i = 0; i < response.data.length; i++) {
      var category = response.data[i].categoryName;
      if (results[category] === null) {
        results[category] = [];
        results[category].push(response.data[i]);
      } else if (results[category] !== null) {
        results[category].push(response.data[i]);
      }
    }

    // Update the state with the appropriate data
    this.setState({
      restaurantId: this.props.route.params.restaurantId,
      restaurantImage: this.props.route.params.restaurantImage,
      restaurantCategories: results,
      totalItems: response.data.length,
      loading: false
    });
  }).catch(error => {console.log(error)});
}
```
The user can add the food items and the quantity he likes, and they will appear in his shopping cart:

![Image of food items in shopping cart]

The user can choose to delete an order, or to confirm and place it. To place the order, we send a HTTP POST request this time, and we give it as body parameter the order object that represents the shopping cart itself. Then, the API will handle the necessary insertions into the database.

```javascript
placeOrder = () => {
  // construct object to send as POST request to the API
  // console.log('Connected User ID => ', this.state.connectedUserId);
  var orderDetails = {
    ItemList: this.state.cart,
    UserID: this.state.connectedUserId,
    RestaurantId: this.state.restaurantId
  }

  axios.post('https://0720b100.ngrok.io/api/Restaurants/order', orderDetails).then(response => {
    console.log('STATUS => ', response.status);
    console.log('Data => ', response.data);

    // response.data contains the orderId (returned by the API)
    this.setState({
      orderId: response.data
    });

    this.toggleModalConfirmation();
  }).catch(error => {
    console.log('Error: ', error.response);
  });
}
```
The user can also check his profile by dragging the screen from the left:
The detailed analytics page is the user’s dashboard when he can check his activity:

Multiple charts can be viewed in this page:
8.3 Web Application (Restaurants)

The web application is the dashboard for the restaurants on AUI campus. They log in with their own credentials, and the first page they see is their dashboard:
Each chart is a component by itself independent from the others. An example of the code for some charts (recent orders):
The pie chat that shows the most popular categories is also an HTTP GET request to the API:
The dashboard provides insightful analytics for the restaurants on campus. It will allow them to take better business-driven decisions and fulfil the students’ needs.

The order page shows all the orders of a restaurant and their different status:

Expanding one category of orders will give detailed information about each. For example, expanding the Delivered Orders would display the following orders:
The Users page display the list of all the users that have ordered from the restaurant with their rating, and total ordered items:

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>Student ID</th>
<th>Phone Number</th>
<th>Date Joined</th>
<th>Total Orders</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael</td>
<td>Green</td>
<td>67890</td>
<td>555-0000</td>
<td>Wed Apr 22, 2023</td>
<td>23</td>
<td>★★★★★</td>
</tr>
</tbody>
</table>

The Items page display the list of all the items that the Restaurant offers, and can change their status if they run out of it in their inventory:

<table>
<thead>
<tr>
<th>Item ID</th>
<th>Name</th>
<th>Category</th>
<th>Image</th>
<th>Price</th>
<th>Waiting Time</th>
<th>Total Ordered</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Panini Roux</td>
<td>Pastries</td>
<td><img src="image1.jpg" alt="Image" /></td>
<td>25.65</td>
<td>18</td>
<td>14</td>
<td>Inactive</td>
</tr>
<tr>
<td>7</td>
<td>Panini Velveine</td>
<td>Pastries</td>
<td><img src="image2.jpg" alt="Image" /></td>
<td>22.5</td>
<td>18</td>
<td>16</td>
<td>Active</td>
</tr>
<tr>
<td>8</td>
<td>Panini Charculea</td>
<td>Pastries</td>
<td><img src="image3.jpg" alt="Image" /></td>
<td>21.2</td>
<td>7</td>
<td>9</td>
<td>Active</td>
</tr>
<tr>
<td>9</td>
<td>Cheese Burger</td>
<td>Burgers</td>
<td><img src="image4.jpg" alt="Image" /></td>
<td>25.2</td>
<td>9</td>
<td>9</td>
<td>Inactive</td>
</tr>
<tr>
<td>10</td>
<td>Fillez Bolognese</td>
<td>Fillies</td>
<td><img src="image5.jpg" alt="Image" /></td>
<td>25.0</td>
<td>7</td>
<td>1</td>
<td>Active</td>
</tr>
</tbody>
</table>
9. Technology Enablers:

I use a wide range of technology in order to fulfil the needs of my system and to solve the problems that students suffer from on campus when it comes to the congestion in the different restaurants

9.1 React

React is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies. React can be used as a base in the development of single-page or mobile applications.

The unique characteristic of React is the ability to deconstruct the web application into small components. This allows for the development of each component independently as well as test it. Also, React allows for very quick development and fast running times, and changes into the page without reloading it.

To illustrate how React works, refer to the image below:

React creates a virtual DOM (Document Object Model) that mimics and watches the original browser DOM. Whenever there is a changer in the browser DOM, React makes the necessary changes in its own virtual DOM (which is the one that the client sees) without refreshing the
page. It works very well because each DOM element is, as mentioned previously, a component by itself with its own independent state.

9.2 React-Native
React Native lets you build mobile apps using only JavaScript. It uses the same design as React, letting you compose a rich mobile UI from declarative components. With React Native, you don't build a "mobile web app", an "HTML5 app", or a "hybrid app". It allows building a real mobile app that's indistinguishable from an app built using Objective-C or Java. React Native uses the same fundamental UI building blocks as regular iOS and Android apps.

9.3 ASP.Net Core
ASP.Net Core API is a framework developed by Microsoft that runs on top of the .NET framework. It allows building web application following the Model-View-Controller model and also build Application Programming Interfaces using C# language. The framework is very well documented and maintenance, and has a large number of packages that can be instantly added into the project if needed to provide a higher level of abstraction in order to save development time and cost.

9.4 Azure Data Studio
Azure Data Studio is a cross-platform tool that allows us to connect to the different Microsoft services and data platforms, including the SQL database. Azure Data Studio can be used as a very light-weight DBMS that has fast loading times and database querying. It offers a modern editing experience, embedded code snippets and an incredible ease of use.
9.5 GitHub

To easily keep track of code iterations and to better manage my workflow, I used GitHub to incrementally push our code to. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

9.6 Postman

Postman is the leading API client on the market that makes it easy to test API and to make HTTP Requests to any API endpoint in an instant. It also allows to write, save and share API requests as well as reading the responses and saving them. I used Postman extensively when I was testing my ASP.NET Core Web API.

9.7 Android Studio

Android Studio is an IDE developed by Google to develop mobile applications on Android operating system. Android Studio also offers mobile emulators, which is what I used it for, to emulate any device I wish to emulate in order to test the code I wrote with React-Native and test the workflow of the mobile application.

9.8 Visual Studio

Visual Code is a text editor developed by Microsoft. I used it to write the code for both the web application and the mobile application using JavaScript and React framework.

As for the Web API, I used Visual Studio Code which is the IDE version of the software in which I developed the API with C#.
10. Conclusion

This project gave me the opportunity to apply what I learned, both academically and personally, to solve a real problem that I, and students around me, suffer from on a daily basis on campus. I followed the software development process from software specification, to design, implement and finally testing the different parts of my application. I also had the chance to use cutting-edge technologies that are widely used around the world. I achieved the requirements that I initially set, as well as the user and system requirements in a timely manner.

The problem I faced in the development of this project was related to the scope and scale of the project. Indeed, the problem is made of three distinct applications (Web, Mobile and API). While the mobile application focused on the food ordering side more than the analytics part, the web application was all about analytics, and how restaurants can draw meaningful conclusions and take better decisions to better serve AUI students.

With a long-term and open-source mindset, the codebase for the three applications are available on my GitHub. Students who wish to use the API endpoints will be welcome to do so if they want to build their own applications and solve different problems based on the data provided by the API.

11. Future Work

The Web Application is where I want to base and focus my future work on. I would like to upgrade the dashboard using real-time analytics, either by using Apache Kafka, with Python, or using Web Sockets with Socket.io, a JavaScript library. Furthermore, I want to add more interactivity to the charts, and add more of them based on the feedback I get from the restaurants. Ultimately, the objective of this system is to help students and reduce the congestion in the campus restaurants. The key component of this solutions lies in the hands of the restaurants themselves. Therefore, the better the dashboard is, the better information the restaurants have access to, and the better the services they can provide to my fellow AUI students.
12. References:


REST. (n.d.). Retrieved from https://restfulapi.net/statelessness/