**Project Deliverable B Briefing: Client Needs, Problem Statement, and Benchmarking**

GNG 1103 – Engineering Design

Team A14

|  |  |  |
| --- | --- | --- |
| Student Number |  | Name |
| 300112652 |  | Josie (Jingshu) Fu |
| 300112337 |  | Nicolas Strebel |
| 300072471 |  | Disala De Silva |
| 300149657300199652 |  | Brunel SontiaSacha Saleh |
|  |  |  |
|  |  |  |

Thursday, October 1 2020

Faculty of Engineering - University of Ottawa

**Introduction**
“Jamz delivery service”(JDS) is a food delivery platform targeted to consumers just outside of the Ottawa area. By using drones JDS eliminates the spread of coronavirus by having 100% zero contact delivery. JDS has taken care of the entire hardware aspect of the project (Drones, navigation system etc). But their team now requires a front line easy to access user interface that customers can use in order to be provided their services. After we went through our initial client meeting with JDS our group was able to outline 8 different client needs which we plan to implement into our Android based user interface.

**Client Statements Obtained from the Client Interview**Many observations were made during the meeting, therefore a table of their needs was produced. Each item was closely analyzed and ranked based on importance. The highest priority was given to the essentials and foundations of the program which provide the backbone for the rest of the user interface to be built on. These needs can be seen below in Table 1.

# **Table 1**: List Client Needs and Translated Client Needs

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Client Statement** | **Interpreted Need** | **Imp** |
| 1 | The app must be on IOS, Android or a Website Platform | The application will run on Android devices | 5 |
| 2 | The app must be able to add items from available restaurants to a “cart" and make an order using those cart items  | The app will have a page which keeps track of what is being ordered and previous orders | 5 |
| 3 | The app must be able to display the live location of the drone using the Google Maps API | The app connects to the GPS of the drone to give real time information on the order’s location | 5 |
| 4 | The app must have a clean, simple and easy-to-navigate user interface platform | UI has a clean and simple design with easy-to-navigate user experience | 4 |
| 5 | The app must be able to display available restaurants and show menu items from each restaurant | The delivery application will have a selection of restaurants and a menu item selection for dishes (with dietary restrictions) | 4 |
| 6 | The app must be able to display the progress of the order and indicate when the food has arrived at the customer’s location | The delivery application will have an order confirmation page/screen and delivery notifications | 3 |
| 7 | The app must be able to make user accounts that stores the user’s name and address | The app will have a simple login/sign-up page for users for a personalized experience and privacy with their personal details | 2 |
| 8 | The app must be able to sort restaurants based on category | A simple page where different types of foods can be selected to narrow down restaurant choices | 1 |

#

# **Problem Statement**

JAMZ needs a modern, simple and easy-to-navigate drone food-delivery service application targeted toward rural areas outside of Ottawa who have limited access to other food-delivery services.

# **Benchmarking**

Three other large companies have been selected for their popularity and use in North America. *Uber Eats, Doordash* and *Skip the Dishes* all offer food-delivery services and have customer-friendly UI. They each have a simple colour palate, consisting of 2-3 colours, with 1-2 of them being black/white/grey while the other is a bright, vibrant colour. In terms of UI, all apps immediately request an address when opening the app, in order to show you surrounding restaurants and have quick-access buttons to the restaurants, search, orders and accounts sections. All offers filter in order to search for the right restaurant that include components such as cuisine, delivery time, delivery price, price range, rating, diet and price range. Certain features that stood out from *Uber Eats* was its ability to have favorites and an order history to easily order past items. From *Doordash*, they show special offers and deals for the restaurants and allow group orders. From *Skip the Dishes*, you can add price specifications to the searches such as “$10 lunch”. Other interesting features were a hygiene rating like in *Deliveroo* and the option to see which restaurant has the fastest delivery on *Postmates.* All of the shared features will hopefully be implemented in our own design if time permits. **Conclusion**

In conclusion, the first client meeting was extremely insightful and useful for further project planning, research, and development. By identifying the problem statement, client needs and their importance, and conducting research through technical and user benchmarking, the team now has a better understanding of what the main objectives of the project are and what the client and users of the application needs. With this, the team can make better judgement of where the focus should be for further research and product concept development.

\*\*\*Redid, posted here just in case\*\*\*

/\*GNG1103 Arduino Laboratory\*/

int photo = 0; // Initialize the variable that will store the value of the photoresistor

int LED1 = 6; // Variable that will store the digital pin 6 for the first LED

int LED2 = 7; // Variable that will store the digital pin 7 for the second LED

void setup(){

pinMode(A2, INPUT); // The photoresistor is connected to analog pin A2 which behave as an INPUT

pinMode(LED1, OUTPUT); // LED1 is connected to digital pin 6 which behave as an OUTPUT

pinMode(LED2, OUTPUT); // LED2 is connected to digital pin 7 which behave as an OUTPUT

Serial.begin(9600); // Start the serial monitor at a baud value of 9600

}

void loop(){

 photo = analogRead(A2); // The variable photo stores the value of the photoresistor

 if(photo < 50){ // The value of the photoresistor is less than 50

 digitalWrite(LED1, HIGH); // LED1 is ON

 digitalWrite(LED2, HIGH); // LED2 is ON

 delay(photo); // Wait for the same amount of time as the value of photo

 digitalWrite(LED1, LOW); // LED1 is OFF

 digitalWrite(LED2, LOW); // LED2 is OFF

 delay(photo); // Wait for the same amount of time as the value of photo

 }else if(photo > 50 && photo < 200){ // The value of the photoresistor is between 50 and 200

 digitalWrite(LED1, HIGH); // LED1 is ON

 digitalWrite(LED2, LOW); // LED2 is OFF

 delay(100); // Wait for 100 milliseconds

 }else if(photo > 200 && photo < 400){ // The value of the photoresistor is between 200 and 400

 digitalWrite(LED1, LOW); //LED1 is OFF

 digitalWrite(LED2, HIGH); // LED2 is ON

 delay(100); //wait for 100 milliseconds

 }else if(photo > 400){ // The value of the photoresistor is greater than 400

 digitalWrite(LED1, LOW); //LED1 is OFF

 digitalWrite(LED2, LOW); //LED2 is OFF

 delay(100); //wait for 100 milliseconds

 }

 Serial.println(photo); // Print the value of the photoresistor on the serial monitor

}

int photo = 0; // Initialize the variable that will store the value of the photoresistor

int LED1 = 6; // Variable that will store the digital pin 6 for the first LED

int LED2 = 7; // Variable that will store the digital pin 7 for the second LED

void setup()

{

pinMode(photo, INPUT); // The photoresistor is connected to analog pin A2 which behave as an INPUT

pinMode(LED1, OUTPUT); // LED1 is connected to digital pin 6 which behave as an OUTPUT

pinMode(LED2, OUTPUT); // LED2 is connected to digital pin 7 which behave as an OUTPUT

Serial.begin(9600); // Start the serial monitor at a baud value of 9600

}

void loop()

{

photo = analogRead(photo); // The variable photo stores the value of the photoresistor

if (photo < 50) // The value of the photoresistor is less than 50

{

digitalWrite(LED1, HIGH); // LED1 is ON

digitalWrite(LED2, HIGH); // LED2 is ON

delay(100); // Wait for the same amount of time as the value of photo

digitalWrite(LED1, LOW); // LED1 is OFF

digitalWrite(LED2, LOW); // LED2 is OFF

delay(100); // Wait for the same amount of time as the value of photo

}

else if (photo > 50 && photo < 200) // The value of the photoresistor is between 50 and 200

{

digitalWrite(LED1, HIGH); // LED1 is ON

digitalWrite(LED2, LOW); // LED2 is OFF

delay(100); // Wait for 100 milliseconds

}

else if (photo > 200 && 400 < photo) // The value of the photoresistor is between 200 and 400

{

digitalWrite(LED1, LOW); //LED1 is OFF

digitalWrite(LED1, HIGH); // LED2 is ON

delay(100); //wait for 100 milliseconds

}

else if (photo > 400) // The value of the photoresistor is greater than 400

{

digitalWrite(LED1, LOW); //LED1 is OFF

digitalWrite(LED2, LOW); //LED2 is OFF

delay(100); //wait for 100 milliseconds

}

Serial.println(photo); // Print the value of the photoresistor on the serial monitor

}