BusiTrack

Project Team: Wahab Ehsan, Avery Anderson, Jaron Dunham, Jared Chadwell, Brandon Chandler

Table of Contents

1. Project Definition

It will be a web application that helps business owners manage their businesses, the process can also be managed through text messaging. Many individuals prefer texting as the main type of communication, so having an application that can also be managed through text will greatly increase its usability. The owner is able to sign up and add their business through the web app, then a message through to the provided phone number will respond to verify. The owner can either use the web app or if in a hurry can text commands. The web app will be able to take the business's daily expenses and gains, and show the monthly or yearly profit/loss.

Architecture

Following the MVC architecture, there are two views, a text message view, and a desktop view. The text message view will be the phone's text message interface. The desktop view will contain the web app in a browser. Each view will interact with our AWS web server via controllers and the web server interacts with the database via another controller.

This project is expected to use the following:
HTML
Python
Javascript
CSS
AWS Lambda
AWS EC2
Twilio
MongoDB

2. Project Requirements:

The requirements for this project include:

- Front-end for web with React js
- Back-end for web with Node and Express js
- Back-end for Phone with Python
- Server for web and phone to communicate using AWS
- Database to store user and business data with MongoDB

No need for additional Hardware except for a Computer and Phone.
Security for users will be a login for the web part but for the phone part users are responsible for the security of their own messages.

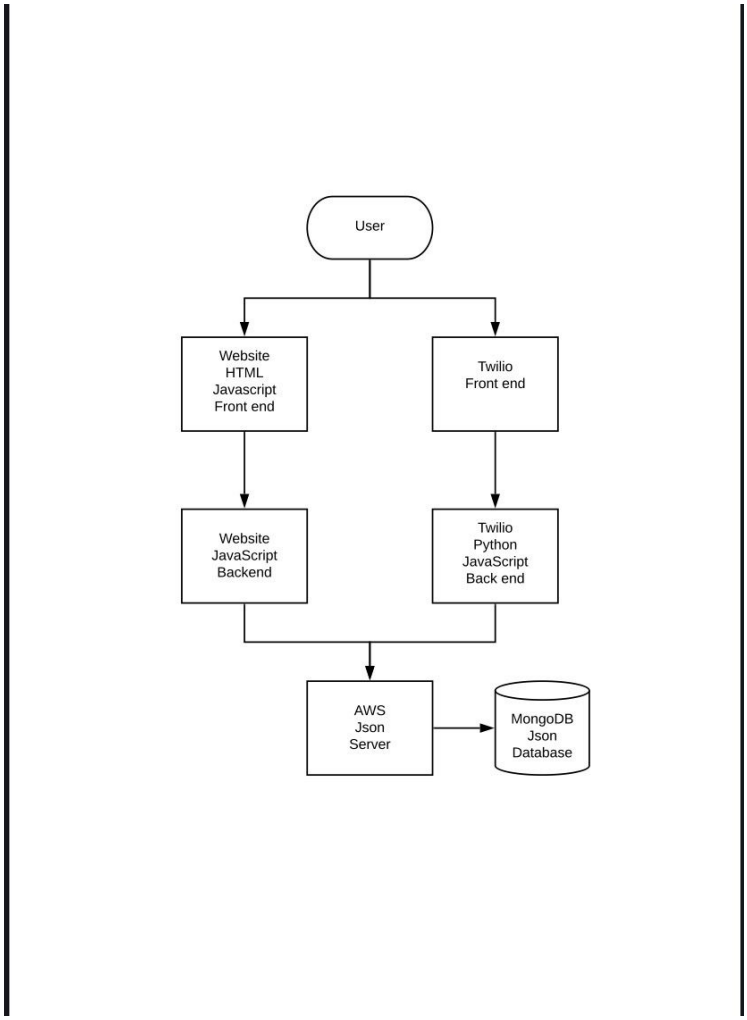
3. Task Required:

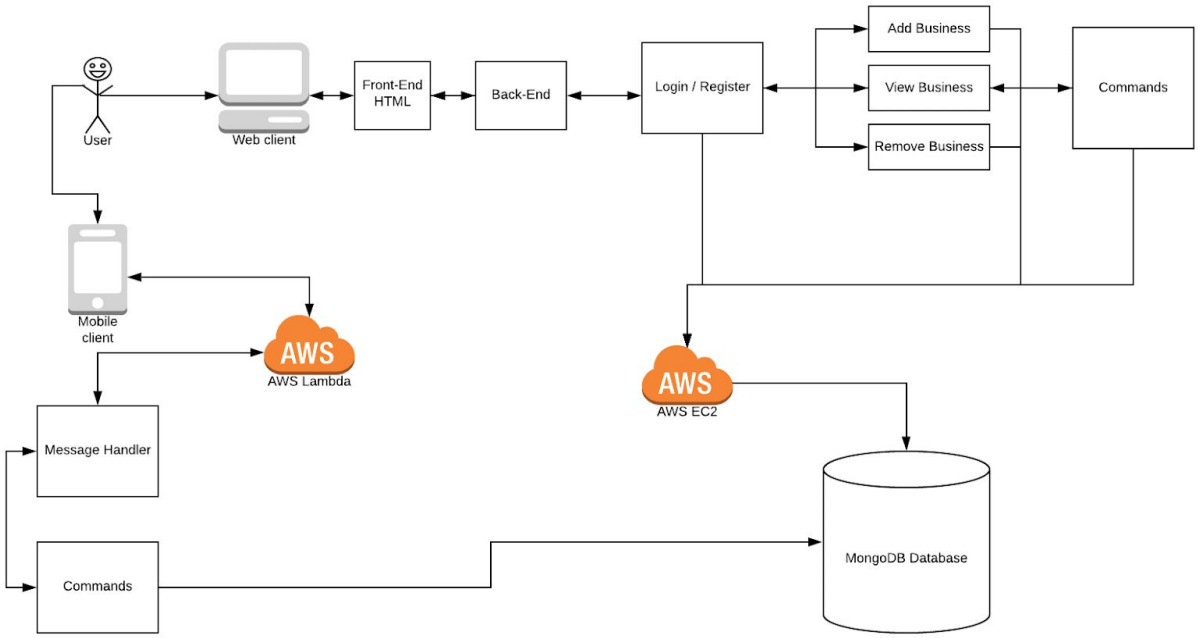
- Wahab: Connect AWS Lambda to Twilio, so the back-end of the phone can be connected to the servers. Make commands that the user can use to update the database through phone. Make sure the commands made with Python Object Oriented Programming and should be dynamic enough so if there needs to change, it would be easier to implement.
- Avery: Database management and connection to Server using MongoDB
- Jared: AWS and Node server management.
- Jaron: Front-end development of web application using HTML and Javascript
- Brandon: Back-end of web development using Javascript.

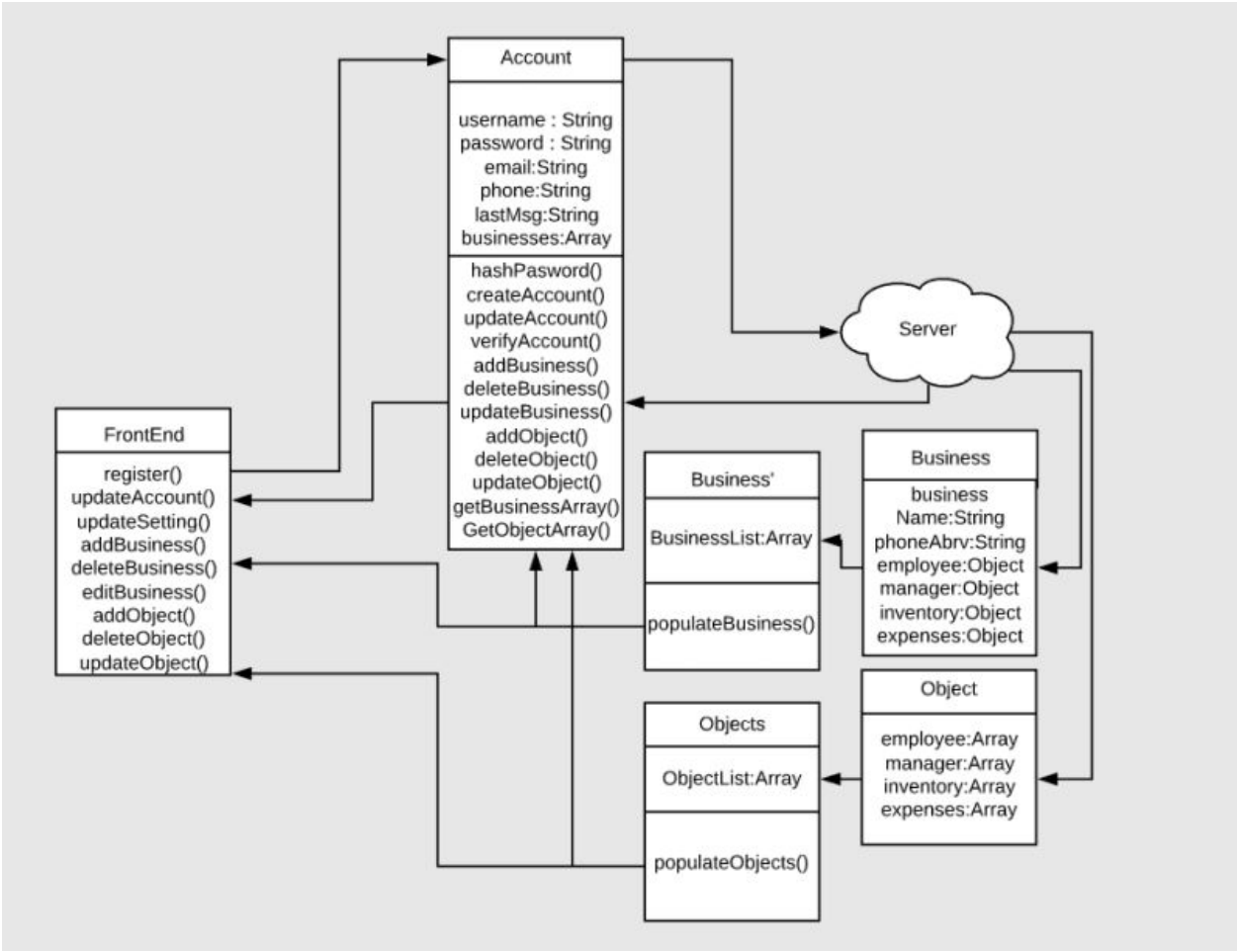
Approach:

Working on Agile approach with the team working on different parts at the same time and connecting at the end to have a working application.

System profile:




User
Web client
Front-End HTML
Back-End
Login / Register
Add Business
View Business
Remove Business
Commands
Mobile client
AWS
AWS Lambda
AWS
AWS EC2
Message Handler
Commands
MongoDB Database


Account
username : String
password : String
email:String
phone:String
lastMsg:String
businesses:Array
hashPasword()
createAccount()
updateAccount()
verifyAccount()
addBusiness()
deleteBusiness()
updateBusiness()
addObject()
deleteObject()
updateObject()
getBusinessArray()
GetObjectArray()
Server
FrontEnd
register()
updateAccount()
updateSetting()
addBusiness()
deleteBusiness()
editBusiness()
addObject()
deleteObject()
updateObject()
Business'
BusinessList:Array
populateBusiness()
Business
business
Name:String
phoneAbrv:String
employee:Object
manager:Object
inventory:Object
expenses:Object
Objects
ObjectList:Array
populateObjects()
Object
employee:Array
manager:Array
inventory:Array
expenses:Array

Feasibility:

Project requires a database that can be edited dynamically without predefined relations. MongoDB is a NOSQL DBMS and allows for dynamic modification without predefined relations using JSON.

The web app is designed for business owners (primarily those managing multiple businesses) to keep things in order and update tables in the DB for a specific business in a web browser or via keyword and new entry pairs sent via SMS from their registered mobile device.

JS can be used to communicate with the frontend JS in the browser.

Twilio will facilitate the SMS exchange between users and the AWS server (Python).

Supporting functions will authenticate a user's SMS submission (Python).

Once authenticated the DB will be updated.

To communicate with Twilio a translator/controller from python to JS and vice versa will be required.

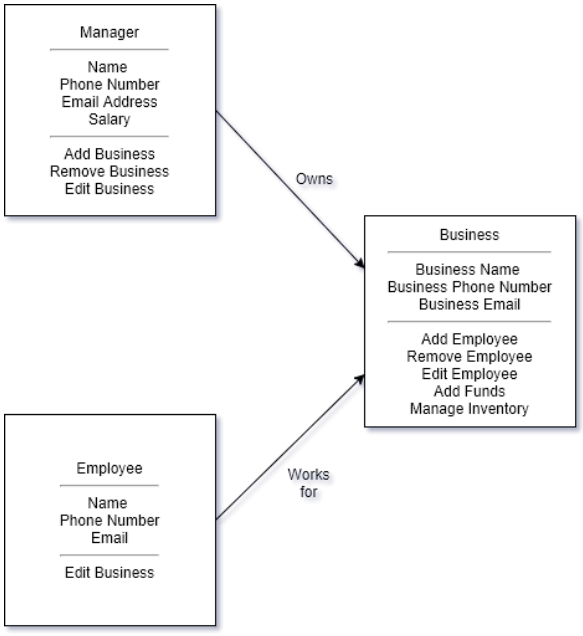
The front end for the browser will consist of HTML/JS and will easily communicate with AWS and the DB as they both use JS.

Draft models:

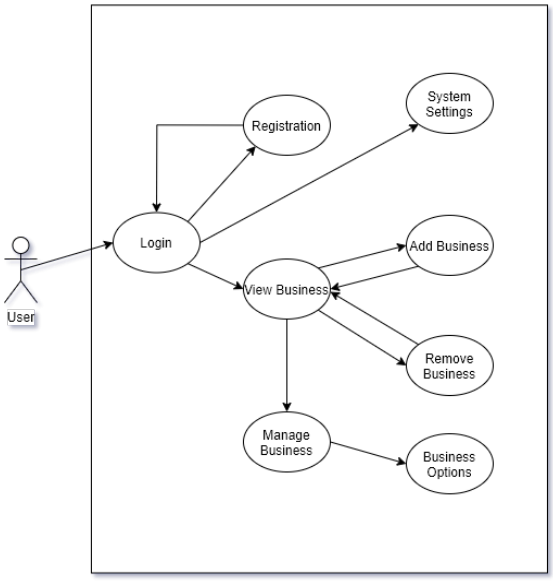
Required features consist of the Web App and feasible features consist of text to DB communication. Ideal feature can be Machine Learning, which allows users to have future prediction of data based on previous data provided.

4. System - Design Perspective

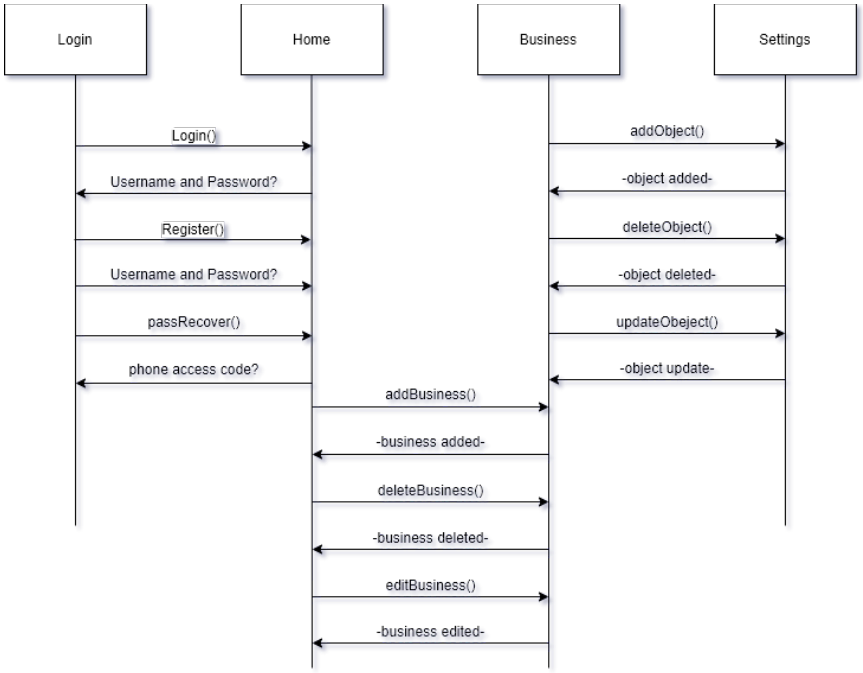
- Design point-of-view
    - Front End Website (Jaron's Task)
        - Class Diagram



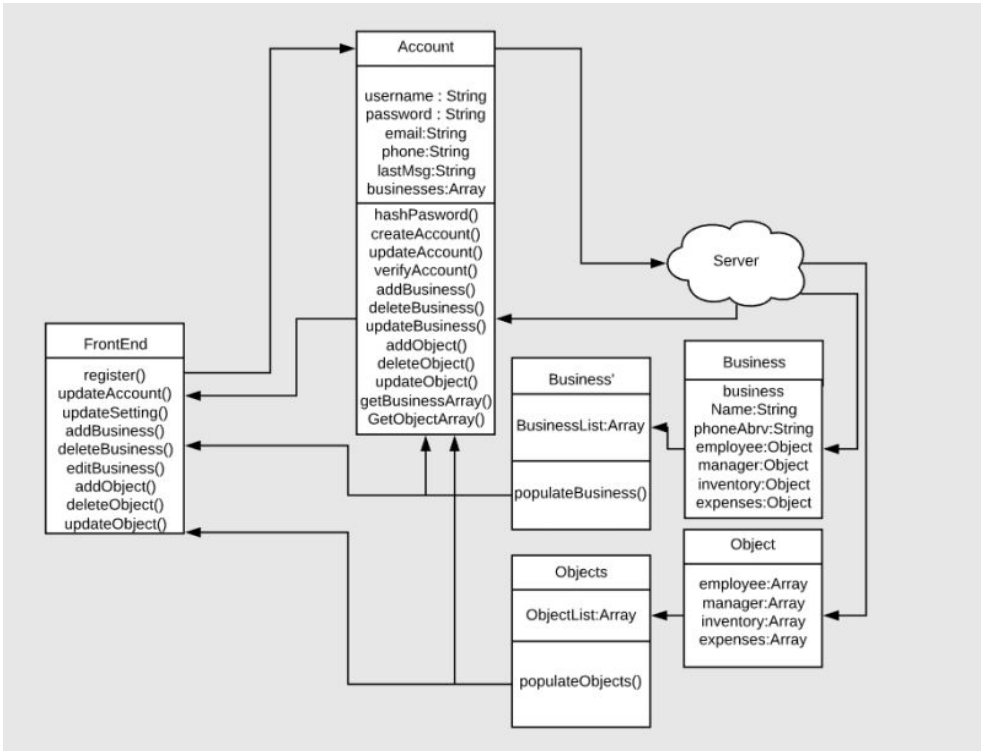
-
  - Use-Case Diagram



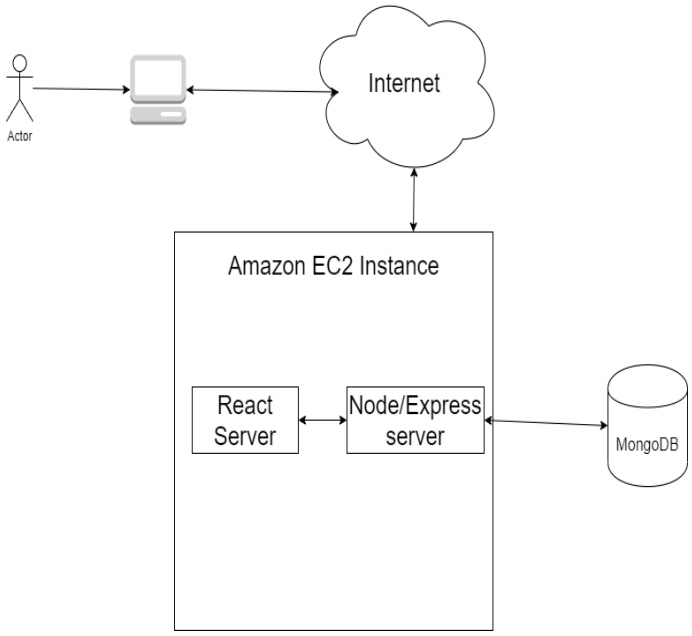
-
  - Sequence Diagram



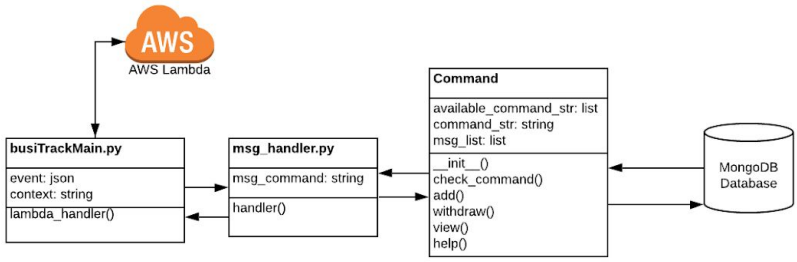
- Backend website (brandon task)
  - UML Diagram



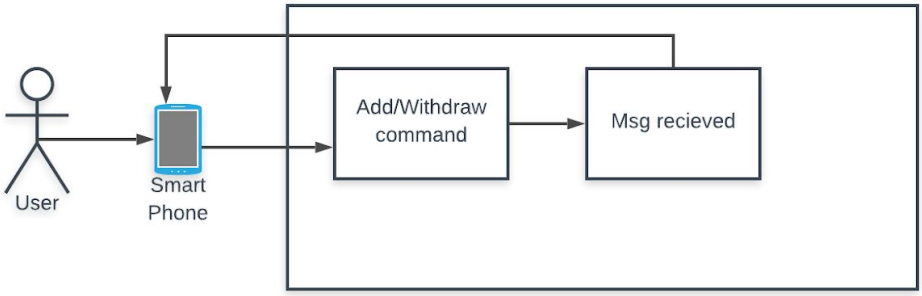
- Housing subsystem



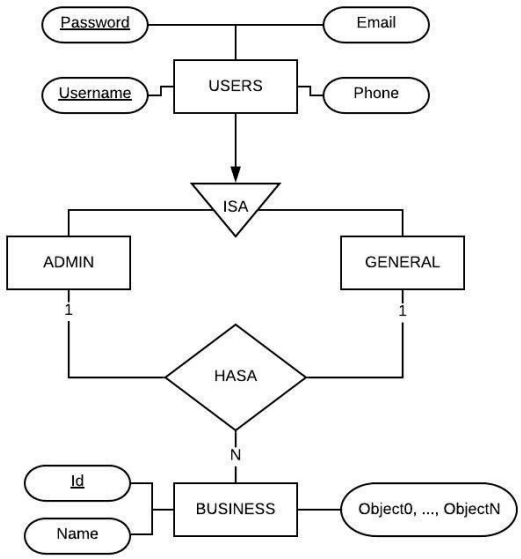
- Back-End Phone (Wahab)
  - UML Diagram



  - Use Case Diagram



- Entity Relationship Model
  - EER Diagram



5. System - Analysis Perspective

- Analysis point-of-view
  - Subsystem Front End Website (Jaron)
    - The front end of the website is obviously the part that the user (in this case, the employer or their employee) will be interacting with. Starting off, a login system would be needed in order to separate and secure the information for an employer's business'. Next a page would be needed to access every business owned by the employer, which in each business page would be various settings that could be used, such as editing information about employees. And finally, some general website settings would be needed
  - Subsystem Backend Account (brandon)
    - The User account Objects will retrieve requests from the front end and mediate with the data through the server ensuring security and expediency
  - Subsystem Backend Businesses-Business(brandon)
    - The account once confirmed will populate the array with businesses with each unique business to allow for access to the front end
  - Subsystem Backend Objects-Object (brandon)
    - Each business contains an array of objects that are each unique for the user so it will populate each business with its corresponding objects array to be sent to the front end
  - Subsystem Server (Jared)
    - The stack we will be using for this project is called a MERN stack (Mongo, Express, React, Node). Our AWS EC2 instance will hold 2

servers, one for the React frontend and one for the Node and Express backend. The react frontend will make calls to the node and express frontend who will in turn make calls back to react and also the mongo database for updates and information requests.

- Subsystem Frontend to Backend (Brandon)
  - The frontend will consist of a react server to create UI components and visualization and make calls to the node and express server for routing and database updates.
- Subsystem Backend to Database (Avery)
  - The node server will be equipped with the mongoDB npm package and the body parser package. Body parser will get the users info from the forums and store it in variables then calling on the mongoDB schemas in javascript to store the information.

- Subsystem Phone Backend (Wahab)
  - The phone backend is where the user/manager will text commands to update their business's info and expenses. The idea is to create simple commands that are typable and fast to come up with so the user doesn't spend time coming up with commands and more so just typing it. General commands have been set up so for including, add, withdraw and view. The message handler cleans the message before it's been sent into actual execution of the command. From there the response is created for appropriate command.

- System
  - Data Analysis
    - The data will include:
      - USER entity Info
        - Username
          - Datatype: String
          - Description: A USER entity must create a unique Username.
        - Password
          - Datatype: String
          - Description: A USER entity must create a password to be hashed and stored in the database.
        - Email
          - Datatype: String
          - Description: A USER entity can provide an email address as a means of contact.

- lastMsg
  - Datatype: String
  - Description: stores the last message a user sent from their mobile device for reference
- Phone
  - Datatype: String
  - Description: A USER entity can provide a phone number as a means of contact and/or to edit BUSINESS entities through phone feature.
- Businesses
  - DataType: Array
  - Description: Indices contain unique business objects

  Business Object data
  - businessName
    - Datatype: String
    - Description: A USER entity must provide a unique name for each of their corresponding BUSINESS entities.
  - phoneAbrv
    - Datatype: String
    - Description: A USER entity can provide an abbreviated name for a business to use as a primary key for that business when making a request via mobile device
  - Employee
    - Datatype: Array
    - Description: Indices contain unique employee objects. Employee objects are composed of the following fields:
      - (ssn, fName, lName, contact, address, pay, group)
        - Datatype: All are Strings
        - Description: Each field contains info associated with a particular employee
  - Manager
    - Datatype: Array

- Description: Indices contain unique manager objects. Manager objects are composed of the following fields:
    - (ssn, fName, lName, group)
        - Datatype: All are Strings
        - Description: Each field contains info associated with a particular manager

- Inventory
    - Datatype: Array
    - Description: Indices contain unique item objects. Item objects are composed of the following fields:
        - (itemNumber, itemName, itemPrice, Stock)
            - Datatype: All are Strings
            - Description: Each field contains info associated with a particular item

- Expenses
    - Datatype: Array
    - Description: Indices contain unique expense objects. One object represents a single day's worth of expense logs. Expense objects are composed of the following fields:
        - (date, amount)
            - Datatype: date is dateTime and amount is stored as a double
            - Description: Each field contains unique expense data for a given day.

6. Project Scrum Report

- Product Backlog (Table / Diagram)

Tasks	Priority
Get the database running	**1**

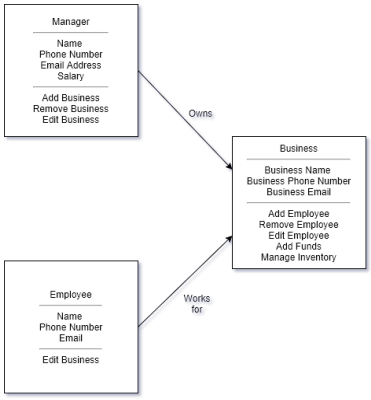
Get the web app running	**2**
Connect database with web app	**3**
Connect phone backend with database	**4**
Get all systems linked together	**5**
Allow for information to viewed on the website and phone front ends	**6**

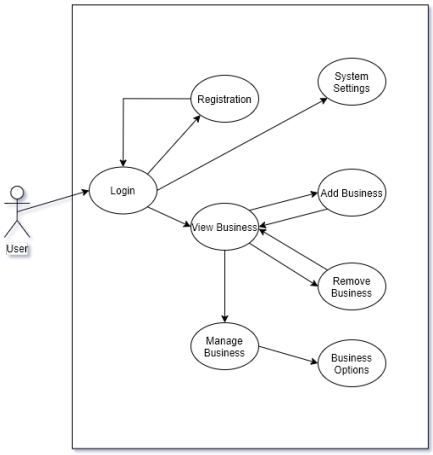
- Sprint Backlog (Table / Diagram)

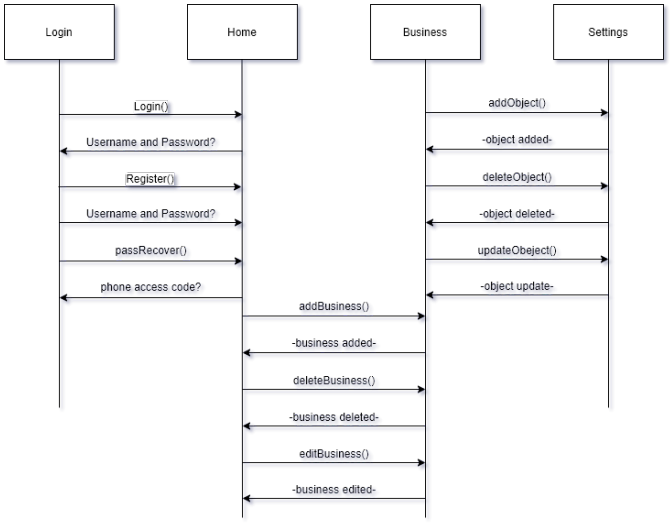
To do:	Doing:	Done:
Better verification for phone number	Add,Update,Delete for front end	Phone backend with database connection
		Implement add and withdraw command
		Implement login system
		Server connection established
		Database running
		Connect all systems
		Data view in front end
		Implement View command on phone

7. Subsystems

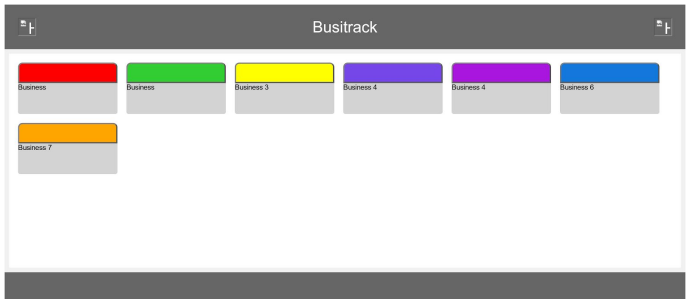
1. **Webpage (Front-End) - Jaron Dunham**
    - Initial design and model
        - Illustrate with class, use-case, UML, sequence…. Diagram

- Class Diagram
  - 

- Use-Case Diagram
  - 

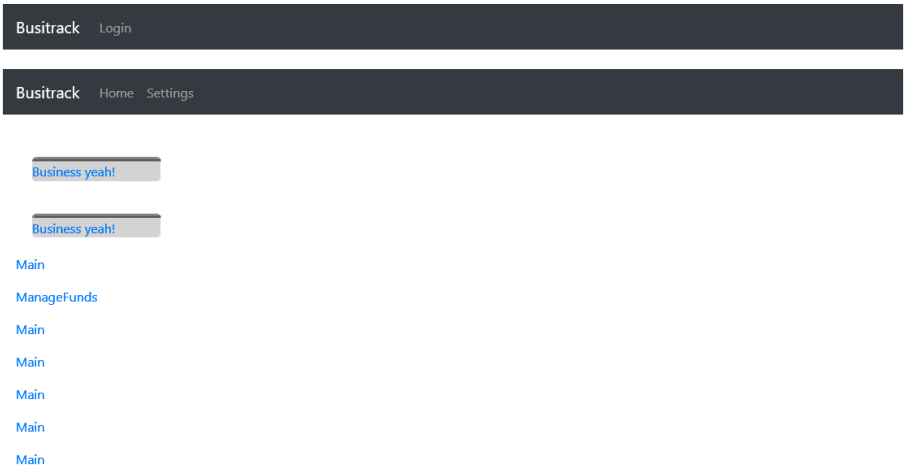
- Sequence Diagram
  - 

■ Design choices (original design)

- 

- Data Dictionary
  - Routes: How application endpoints respond to user requests
    - Login post: authenticates password which then will retrieve the User profile based on username
    - Signup post:Verify's unique username which then will send user data to be saved
    - Fetch Business get: Sends the business data attached to the connected user
    - User business get: fills business through loop
    - Add business post: adds business to user data
    - Remove business post: removes business from user data
    - Add Employee: adds employee to user data for specific business
    - Remove Employee: removes employee from user data for specific business
    - Add manager: adds manager to user data for specific business
    - Remove manager: removes manager from user data for specific business
    - Add inventory:adds inventory to user data for specific business
    - Remove inventory: removes inventory from user data for specific business
    - Passport Login: verify username and hash password upon success creates the User object to be used throughout

- Passport Signup: Verify unique username and upon success hashes the password to be sent and saved in database
- NPM: Node Package Manager
- Node: JavaScript runtime environment

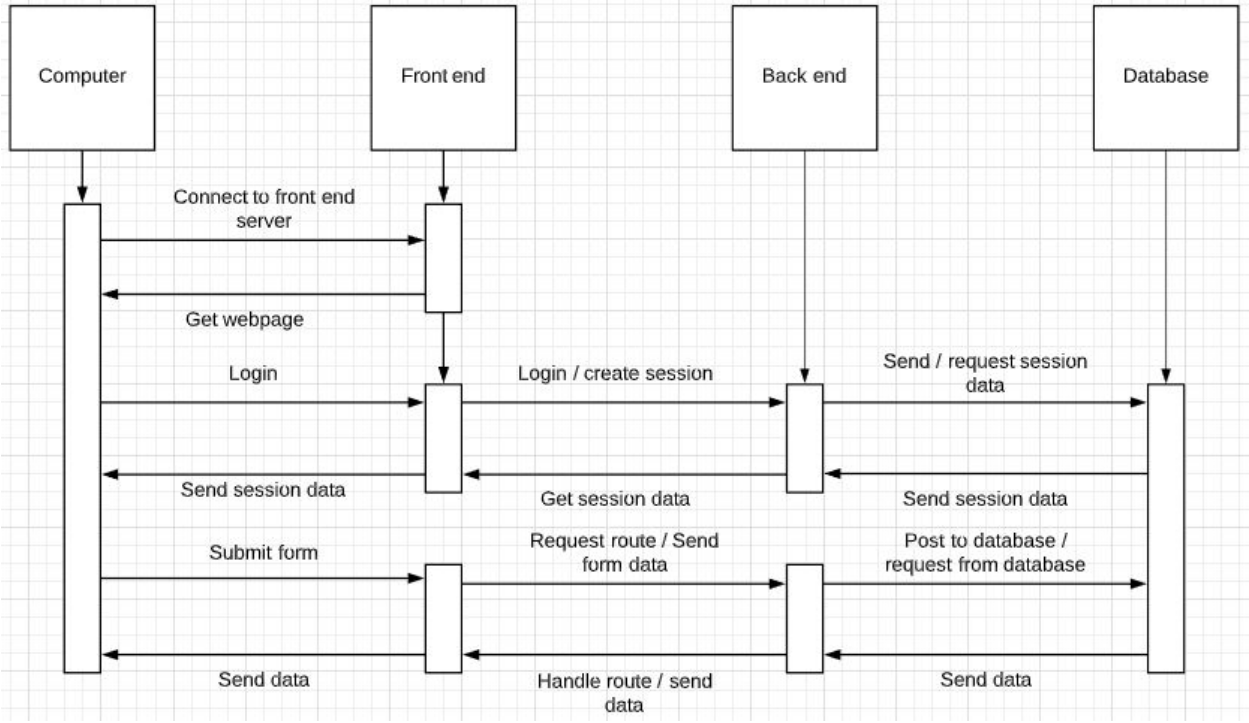
- Refinements:
  - Website Layout (using Node/React)
    - What was used before was a temporary layout where everything could be accessed through simple html links
    - Now using react, we can load up pages so they appear on screen when clicked
    - 

- Scrum Backlog (Product and Sprint - Link to Section 6)
- Coding
  - Approach (Functional, OOP)
    - React js / Node js
      - This is being used for creating the beside, this allows us to store parts of the website away for use later on and in multiple different places if need be
      - It also allows for things to be preloaded so that when they are access, you don't have to do to a completely new page

- Language
  - Html
    - CSS
  - Javascript
    - React.js
    - Node.js
- User training
  - Learning to navigate through the website
  - Learning the command structure needed for mobile device

2. Server (back-end) - Jared Chadwell

- Initial design and model
  - Illustrate with class, use-case, UML, sequence…. Diagram
    - Sequence diagram

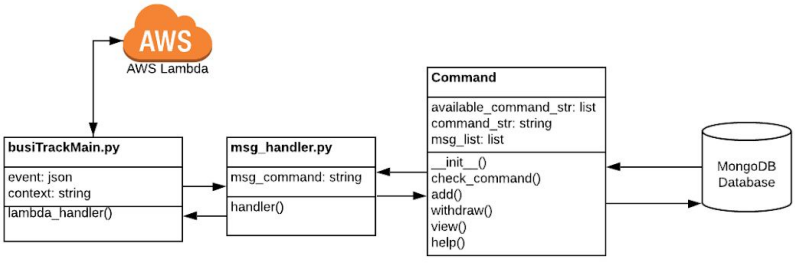


  - Design choices

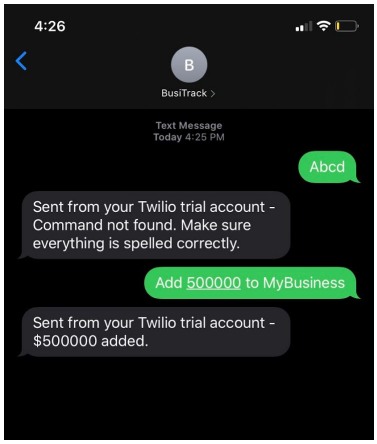
- Data Dictionary
  - Routes: How application endpoints respond to user requests.
  - Middleware: Provides services to software applications
  - NPM: Node Package Manager
  - Dependencies: npm package middleware that the application depends on to run
  - Node: JavaScript runtime environment
  - Express: npm package that provides robust routing
  - Cors: Cross Origin Resource Sharing
  - Axios: Promise based HTTP client for the browser and node.js
- If refined (changed over the course of the project)
  - Reason for refinement (Pro versus Con)
  - Changes from initial model
  - Refined model analysis
- Scrum Backlog (Product and Sprint - Link to Section 6)
- Coding
  - Language
    - JavaScript
      - Node js
      - Express js

3. Phone System (back-end and front-end) - Wahab Ehsan

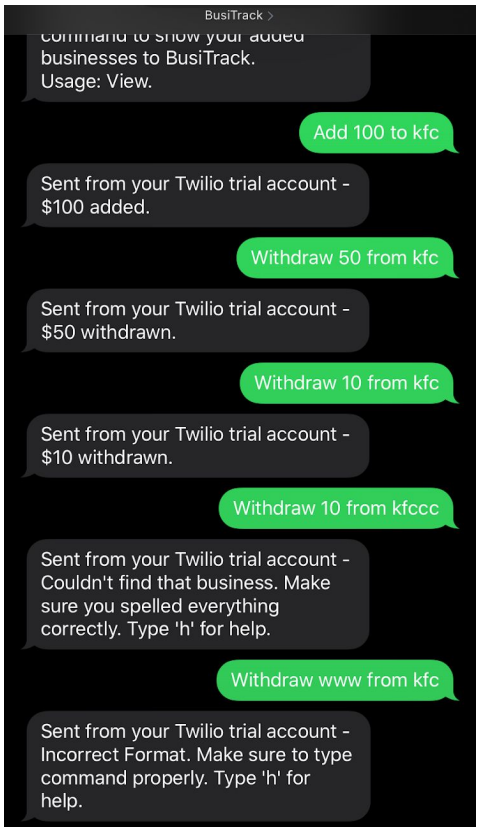
- Initial design and model
  - Illustrate with class, use-case, UML, sequence…. Diagram



- Kept the same design but added a User class which also has access to Mongodb and has a translator in between files with contact with db. Discussed below.
  - Design choices



-
- User gets a confirmation text saying if the amount has been added.



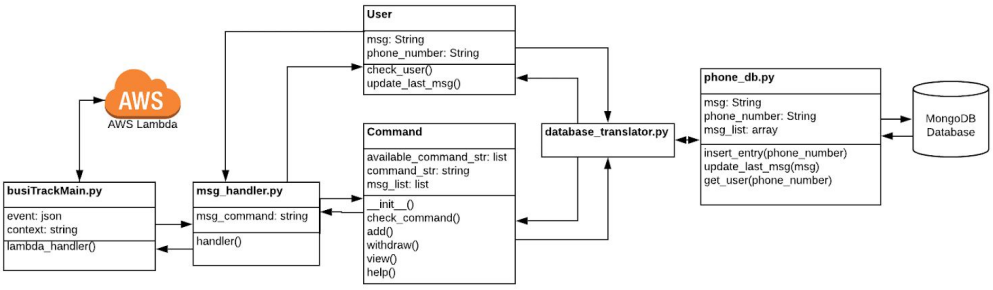
-
- If the user has business ‘kfc’, it can verify the business name and respond accordingly if spelt wrong.
- Also makes sure the command is in correct format.

- Data Dictionary
  - Phone_number: Users phone number
  - Msg_list: Array of parsed message content from user.

- Twilio: Online service, which allows programmable texting
- PhoneAbrv: This is the business name without spaces and is used for phones to compare and select businesses for users. EX) if the business name is “Bob’s Auto”, then phoneAbrv would probably be ‘bobs’ or ‘bobsauto’. The user gets to name in the web application.
- Message handler: Handles message and parses into list and checks if valid command.
- Commands: Commands are messages that will be sent to the phone to update the database.

○ If refined (changed over the course of the project)

- Was having to not check the phone number of the user at all. Implemented this so it is more secure and sends a link to sign up to register.
- Phone database file separate from web application database file but same database, for different queries.
- Redinged design (Diagram and Description)



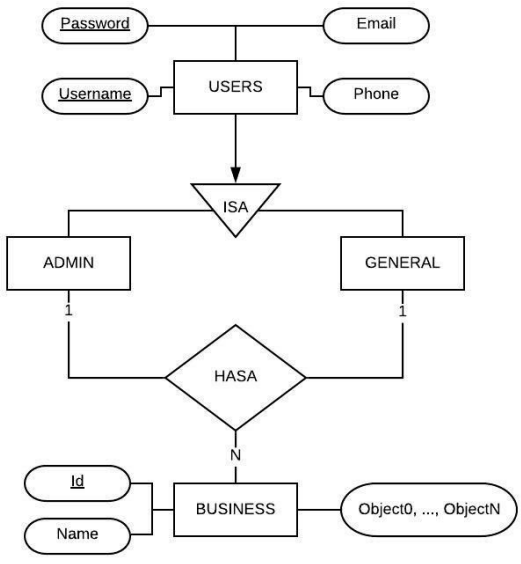
- The diagram above is the updated version of the previous diagram of UML with added User model and controllers.
- Changed some wording on the phone text replies to sound friendly and informative.

○ Coding

- Python was used for the back-end with Object oriented programming for Commands and Users.

        - AWS Lambda was used for the messages to be connected to the database with Twilio as the phone number service.
    - User training
        - To use the command features, make an account through the web application and then follow these instructions:
        - Add Command: Use this command to keep track of profits or earnings. Usage: ‘Add (amount in digits) to (Your Business)’.
        - Withdraw Command: Use this command to keep track of losses or expenses. Usage: ‘Withdraw (amount in digits) from (Your Business)’.
        - View Command: Use this command to show your added businesses to BusiTrack. Usage: ‘View’.
        - Help Command: Use this command to get a list of commands that work with the system. Usage: ‘H’.

4. **MongoDB (DBMS) - Avery Anderson**
    - Initial design and model
        - Illustrate with class, use-case, UML, sequence…. Diagram
            - EER Diagram



        - Design choices
            - Initially planned for a traditional relational database model as typically seen in SQL oriented database

management systems that would allow for dynamic creation of business entities

- Data Dictionary
  - USER entity Info
    - Username
      - Datatype: String
      - Description: A USER entity must create a unique Username.
    - Password
      - Datatype: String
      - Description: A USER entity must create a password to be hashed and stored in the database.
    - Email
      - Datatype: String
      - Description: A USER entity can provide an email address as a means of contact.
    - lastMsg
      - Datatype: String
      - Description: stores the last message a user sent from their mobile device for reference
    - Phone
      - Datatype: String
      - Description: A USER entity can provide a phone number as a means of contact and/or to edit BUSINESS entities through phone feature.
    - Businesses
      - DataType: Array
      - Description: Indices contain unique business objects Business Object data
      - businessName
        - Datatype: String
        - Description: A USER entity must provide a unique name for each of their corresponding BUSINESS entities.
      - phoneAbrv
        - Datatype: String
        - Description: A USER entity can provide an abbreviated name for a business to use as a primary

key for that business when making a request via mobile device

- Employee
  - Datatype: Array
  - Description: Indices contain unique employee objects. Employee objects are composed of the following fields:
    - (ssn, fName, lName, contact, address, pay, group)
      - Datatype: All are Strings
      - Description: Each field contains info associated with a particular employee
- Manager
  - Datatype: Array
  - Description: Indices contain unique manager objects. Manager objects are composed of the following fields:
    - (ssn, fName, lName, group)
      - Datatype: All are Strings
      - Description: Each field contains info associated with a particular manager
- Inventory
  - Datatype: Array
  - Description: Indices contain unique item objects. Item objects are composed of the following fields:
    - (itemNumber, itemName, itemPrice, Stock)
      - Datatype: All are Strings
      - Description: Each field contains info associated with a particular item
- Expenses
  - Datatype: Array
  - Description: Indices contain unique expense objects. One object represents a single day's worth of expense logs. Expense objects are composed of the following fields:
    - (date, amount)
      - Datatype: date is dateTime and amount is stored as a double

- Description: Each field contains unique expense data for a given day.

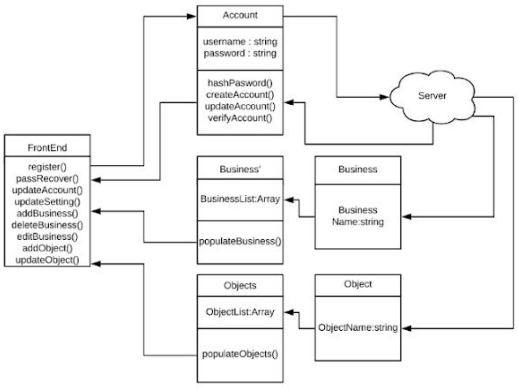
- If refined (changed over the course of the project)
  - Reason for refinement (Pro versus Con)
    - Opted to swap out the traditional relational model used with SQL databases for the nested structure provided in MongoDB. Finding a user in one collection or document and pairing them with their user created business entities takes time and would become cumbersome with a large user base. Nesting a user's businesses within the businesses field for that particular user only requires for that user's document to be searched rather than multiple documents or collections.
    - Opted to limit the fields for a business object to six predefined fields (businessName, phoneAbrv, employee, manager, inventory, expenses). Doing this will give less freedom to users but will remove some of the planning they would need to do to structure their businesses
  - Changes from initial model
    - The keys defined in the initial model remain keys for the new model and are used to identify unique objects within the nested object arrays.
    - The logic used to identify user's and their associated data are the same as before. The location of the data is the only thing that has changed
  - Refined model analysis
    - The new model now only has to search a unique user's document in the user collection to have access to all data related to that user and their businesses. This prevents MongoDB from wasting time searching multiple documents and/or collections for keys to determine relationships between users and their businesses.
  - Refined design (Diagram and Description)

- (Need to learn how to build a diagram for a NoSQL DBMS)

- Scrum Backlog (Product and Sprint - Link to Section 6)
- Coding
  - Approach (Functional, OOP)
    - Used OOP approach. MongoDB stores data as objects, so the code must make use of OOP.
  - Language
    - Javascript
- User training
  - Training / User manual (needed for final report)
    - Both the web and phone front ends should restrict how the user interacts with the database.
- Training
  - No users should have direct access to the database. Training should be limited to navigating the website and using a mobile device to interact with a user's data.

5. Web-Backend (Brandon Chandler)

- Initial design and model
  - Illustrate with class, use-case, UML, sequence…. Diagram



  - Design choices
    - The design has not changed much using a router for the front end movement and calls, the Main Object, and Business are moved to DBMS and the Business' and Objects are currently under development, the account

system is up and running with some minor changes needed at this point.

- Data Dictionary
  - Express main
    - Server file-initializes the express/node server which is what completes requests and communicates with the MongoDB server.
    - Package-is the reference pull that brings in the node-modules
    - Config
      - Database-call to the database so that there is no need to change it multiple times can be changed once here
      - Passport-the main login/register function that uses the bcrypt system inside User to get the hashed string for password while also keeping the unhashed string from passing over the network.
    - App
      - Routes-this is the calls which keep the connection private will read the data that called the commands and then within and currently sets up the Passport calls. Nothing moves across the network from Passport to Routes and Routes has view rights to anything that calls it.
- If refined (changed over the course of the project)
  - Reason for refinement (Pro versus Con)
    - We are using react which at the time of creating the system I missed, so we have had to introduce the Axios system for communication between currently doing the final modification for this, it is all going smoothly
  - Changes from initial model-moved the Initial builder for Business and Object to DBMS and can pull simply into an array from there.

        - Refined model analysis-the model holds and the connections are the same just who's working on it is different.
        - Redesigned interactions (Diagram and Description)
        - Scrum Backlog (Product and Sprint - Link to Section 6)
    - Coding
        - Approach (Functional, OOP) - OOP
        - Language
            - Javascript-express
    - User training
        - Training / User manual (needed for final report)
            - Everything on the backend is hands off, it has no connection with the user, the front end training takes care of anything that the user requires.
            - For developers in the config file database will need to be updated to whatever MongoDB you are using.
            - For developers in the config file houses the universal keywords used throughout the system so if you need to change a keyword changing it here is the only necessary step.

8. Complete System

- **GitHub Link to Project:**
    - https://github.com/WahabEhsan/BusiTrack
- **Application Implementation:**
    - http://ec2-52-15-53-59.us-east-2.compute.amazonaws.com:3000/
- **User Manual as separate file.**
- **Team Member Descriptions:**
    - **General:**
        - Due to Covid-19 outbreak, Our team faced several challenges including communication. The team was able to use online resources like, Discord and Zoom to continue communication. Times for meetings were set several times a week and each

member attended the meetings with the goal to complete the project.
        - The project came out to be what was envisioned with few differences because of the circumstances. There were few other features that could've been added if we had more time. Overall, it turned out to be a successful and functioning project.
    - **Wahab Ehsan:**
        - Responsible for developing the phone back-end for the project. Worked with commands and the responses the user would get after using the phone side. Made sure the commands were worded friendly and informative.
    - **Avery Anderson:**
        - Responsible for the development of the MongoDB DBMS architecture and CRUD operations. Manages the DBMS repository.
    - **Jared Chadwell:**
        - Responsible for back-end to front-end communication, debugging, and EC2 instance maintenance
    - **Jaron Dunham:**
        - Responsible for the development of the Front-End GUI for the project, dealt with creating the pages and components needed for each respective page
    - **Brandon Chandler:**
        - Responsible for interconnectivity of backend routing and user authentication, ensuring smooth and secure connection with server.