Software Platform to improve safety at Gwinnett Technical College – Students use mobile device to request a Chaperone to walk with from one building to another building on campus.

Advanced Systems Class, Fall Semester, 2015

PROJECT CHARTER

Mission Statement
We seek to provide security to the students and staff of Gwinnett Technical College through an interactive software platform that connects all users as a pedestrian community.

Project Title: Chappy
Project Start Date: 9/10/15 **Projected Finish Date:** 12/10/15
Budget Information: $87,200 (est.)

Project Manager: Ryan Brown, ###-###-####, rbrown5262@student.gwinnetttech.edu
Project Objectives
The Chappy application will accomplish the following objectives:

- Clients
  - Provide a SaaS solution for affordable, secure pedestrian travel to and from locations around the Gwinnett Technical College campus
  - Provide a social community platform for Gwinnett Technical College staff and students to meet in person
- Company
  - Generate revenue based on a SaaS subscription model for staff and students as well as in-application advertisement

Main Project Success Criteria
Upon project completion, success or failure will be determined by the following criteria within the following timeframes:

- Production Deployment
  - 100% deployment of all essential product features
  - 90% or more of all non-essential product features completed on deployment
  - Application deployment with 100% bug-free code
- Year 1
  - 60% Gwinnett Technical College population app usage
  - Fully realized IRR
  - Offers received from other campus' for product expansion

Approach

Application
In order to meet the above criteria and objectives, the following systems and technologies will be used:

- Product Software/Hardware
  - Database
    - WinHost Cloud Hosting Service

        - Microsoft MSSQL 2008 R2
        - Microsoft Windows Server 2012
    - Application Server
        - WinHost Cloud Hosting Service
        - Microsoft Windows Server 2012
        - Microsoft Windows Communications Framework
        - IIS 7.0 with FTP enabled
    - Client (UI) Targets
        - Microsoft Windows Desktop
        - Apple iOS
        - Google Android OS
        - Web
- Development Software
    - Versioning: Git with GitHub (15)
    - Server: Microsoft Server 2012 (3)
    - IDE: Visual Studio 2014 (15)

Labor Allocation
The following team members will be allocated to the following portion of the project for its full duration:

- Leadership - Ryan, Adam
- Database - Cole, David, Corey, Binita
- Application - Blake, Jason, Tim, Jeremy
- Client (UI) - Stephanie, Ricky, Syam, Corey

Project Phase Timeline
The following is an estimated timeline of the project's phases and deliverables:

- Phase 0
    - Pre project planning
- Phase 1
    - Database tables created
    - Application server created
    - Database/Application server integration verified
    - Version 1 of client wireframes created
    - Feasibility testing
    - Verification of Proof of Concept
- Phase 2
    - Client/Application server integration verified
    - Database stored procedures completed
    - Application server basic functionality completed
- Phase 3
    - Client look and feel polishing
    - Application bug fixing
    - Database query tuning

- Phase 4
    - Beta and UA testing
    - Bug fixing
- Phase 5
    - Product launch

SCOPE STATEMENT

The following provides an overview of the Chappy Application Project scope.

Living Document Notice

It is acknowledged by all parties that this is a living document and is likely to contain revisions. Upon approved document revision, all parties are to be notified in a timely and agreed-upon manner with regards to changes made. It is the responsibility of each individual to review the document and note any changes made that affect them.

The authorized owner of this document responsible for approving changes and revisions is:

Ryan Brown – Project Manager

Business Opportunity

College campuses with many buildings expose security risks for students and staff travelling on foot between them. Instead of hiring extra security to walk students to and from each building, it would be more cost effective to utilize the student population. By creating a crowdsourced platform for this service, costs are significantly reduced to the institution while providing increased security to its students and staff.

Project Goals

The goal of the Chappy Application Project is to develop a SaaS platform to provide students with the opportunity to chaperone other students to and from college campus buildings.

Project Completion Deliverables

The following items will be delivered upon project completion:

- Chappy Application Beta
    - Database Server
    - Application Server
        - Web client
    - Client Software
        - Windows Desktop
        - Android
        - iOS
- End user documentation
- Maintenance and administration documentation

Non-Goals

The following items will be considered out of scope for the purposes of this project. In the event that any of the below items should be added to the project, a Change Request form should be completed and documented.

- …[1]

Project Milestones

The following outlines project milestones to be delivered at their set dates:

- **Phase 0 – 2 weeks from project start**
  - Tasks
    - Pre project planning
  - Deliverables
    - Project charter
    - Project scope document initial draft
    - Project WBS draft
- **Phase 1 – 2 weeks**
  - Tasks
    - Create database tables
    - Create application server
    - Integrate and verify application and database server connectivity
    - Create client wireframes
  - Deliverables
    - Development-ready database server
    - Development-ready application server
    - Client wireframes
- **Phase 2 – 3 weeks**
  - Tasks
    - Integrate clients with application server
    - Finalize database tables and stored procedures
    - Complete application base functionality
  - Deliverables
    - Test-ready database server
    - Test-ready application server
    - Test-ready clients
- **Phase 3 – 2 weeks**
  - Tasks
    - Improve overall performance
    - Fix bugs
    - Improve client UI
  - Deliverables
    - Application beta

[1] 9/24/15 – Specific non-goals have yet to be discussed with the project team

- **Phase 4 – 2 weeks**
  - Tasks
    - Beta and UA testing
    - Bug fixing
    - Create product documentation
  - Deliverables
    - Version 1.0 of application
    - Product documentation
- **Phase 5 – 1 week**
  - Tasks
    - Train sales and support teams
  - Deliverables
    - Support team
    - All project documentation
    - Business documentation

Objections or Issues

This section is reserved for objections or issues to the project as they arise, as well as documentation on issue resolution.

FUNCTIONAL SPECIFICATIONS

Core Functionality:

Customer Request Chaperone- The creation of transaction begins here, Having a Customer id and two locations, starting and end, this method will instantiate a new a Transaction and place a status of looking for chaperone.

Chaperone Responds- This method will place a chaperone on a transaction. The method is called from the Chaperone selecting a Customer to pick up. Transaction is updated with appropriate Status.

Chaperone Arrives- Timestamp on Transaction for Customer being pick up. Transaction is updated with appropriate Status.

Chaperone Escorts Customer- Time stamp for beginning of journey. Transaction is updated with appropriate Status.

Chaperone Confirm Destination Arrival- Timestamp on Transaction that both agree they have made it to the Customers destination.

Asynchronous – Program the system to allow for asynchronous operations, allowing for multiple process to occur without being hung up with procedural programming. Provides a better user experience allowing the user to have a more fluid experience while background processes continue to work.

Customer Rates Supplier- Method that allows for a larger text block from customer and a rating (Ex. 4 stars) to be added to the Transaction.

Supplier Rates Customer- Chaperone has a large text block and a rating system (Ex. 4 stars) to rate the customer added on the Transaction.

Customer Functionality:

Panic Button- A method that would be used to alert security that a problem has occurred, whether that is police or campus security is to be determined.

Rate Chaperone- Large text block and rating passed, added to transaction.

Cancel Transaction- Cancels the created transaction, changes status of transaction removing it from available open transaction query.

Request Chaperone- Creates a new transaction, User ID, starting and stopping destination are passed in this method. Awaiting Chaperone Status is set.

Chaperone Functionality:

Panic Button- A method that would be used to alert security that a problem has occurred, whether that is police or campus security is to be determined.

Cancel Transaction- Chaperone cancels their arrival, sets the transaction back into looking for Chaperone state. Removes Chaperone from the Transaction.

Respond to Supplier- Places Chaperone on transaction and changes Transaction status removing it from the open transaction query.

Start Travel- Time stamp on Transaction for beginning of transportation.

Rate Customer- Large text block and rating passed, Ex. 4 stars, added to the transaction.

Shared Functionality Chaperone/Customer:

View ratings- Brings back reviews for person you are looking at, Chaperone or Customer. Pulls list of all reviews and averages ratings from all previous transaction the user is associated with.

Registration- Creates a new user in database, passing in Name and Password. Checks against database to see if Username already exist and prompting user about success or failure.

View History- Returns list of Transactions for a given user.

Add Image- Adds image to profile account, places image in database.

View Image- returns stored image in database.

System Functionality:

Create Account- Takes a Username, password, and which user type they are creating i.e. Chaperone or Customer. Adds new row to database.

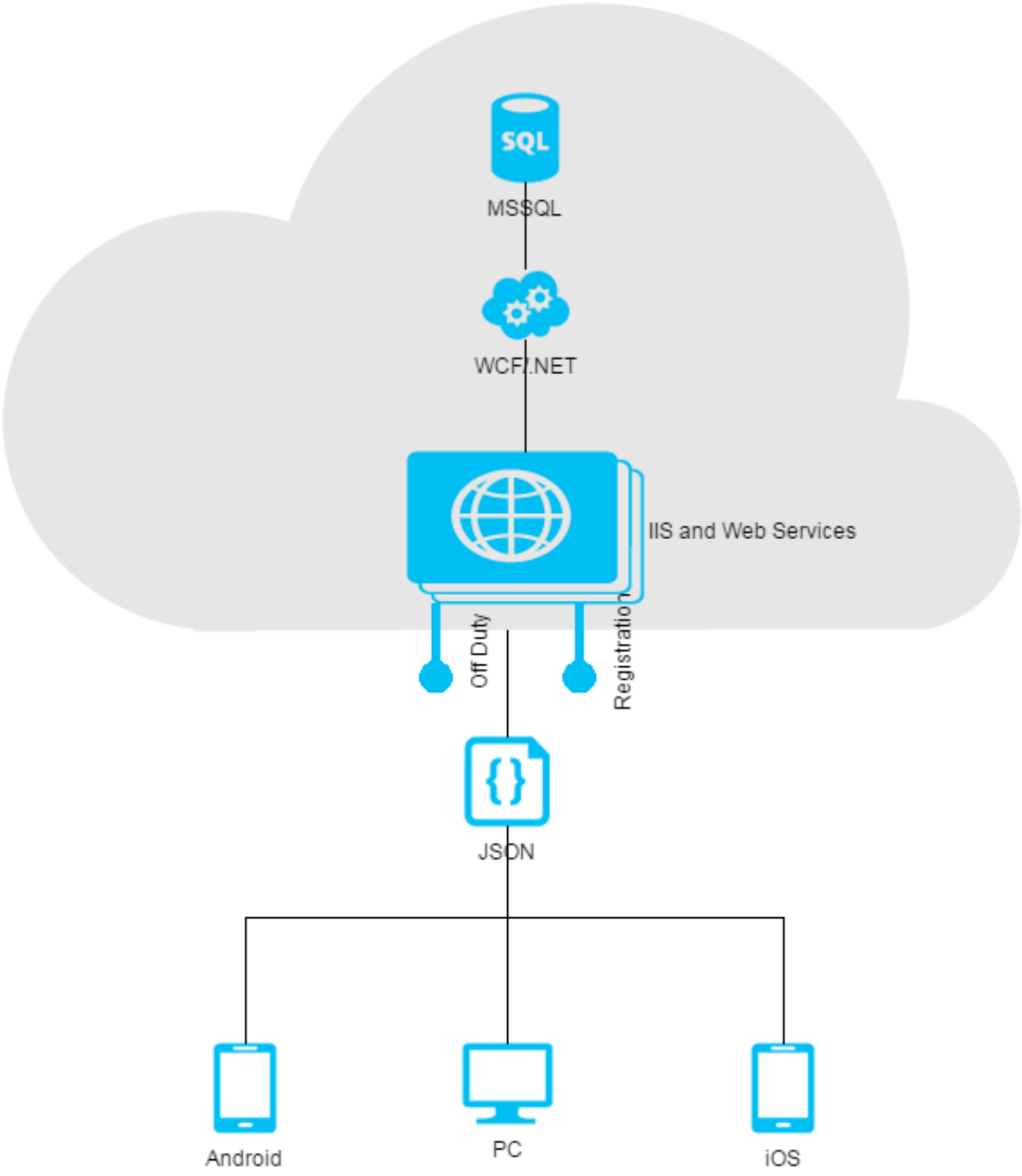
Assign Chaperone- Assigns a Chaperone to a Transaction, changes status of Transaction.

View Transaction- Returns information about given Transaction.

Modify Transaction- Updates fields on a Transaction, would need to be overloaded for all variations.

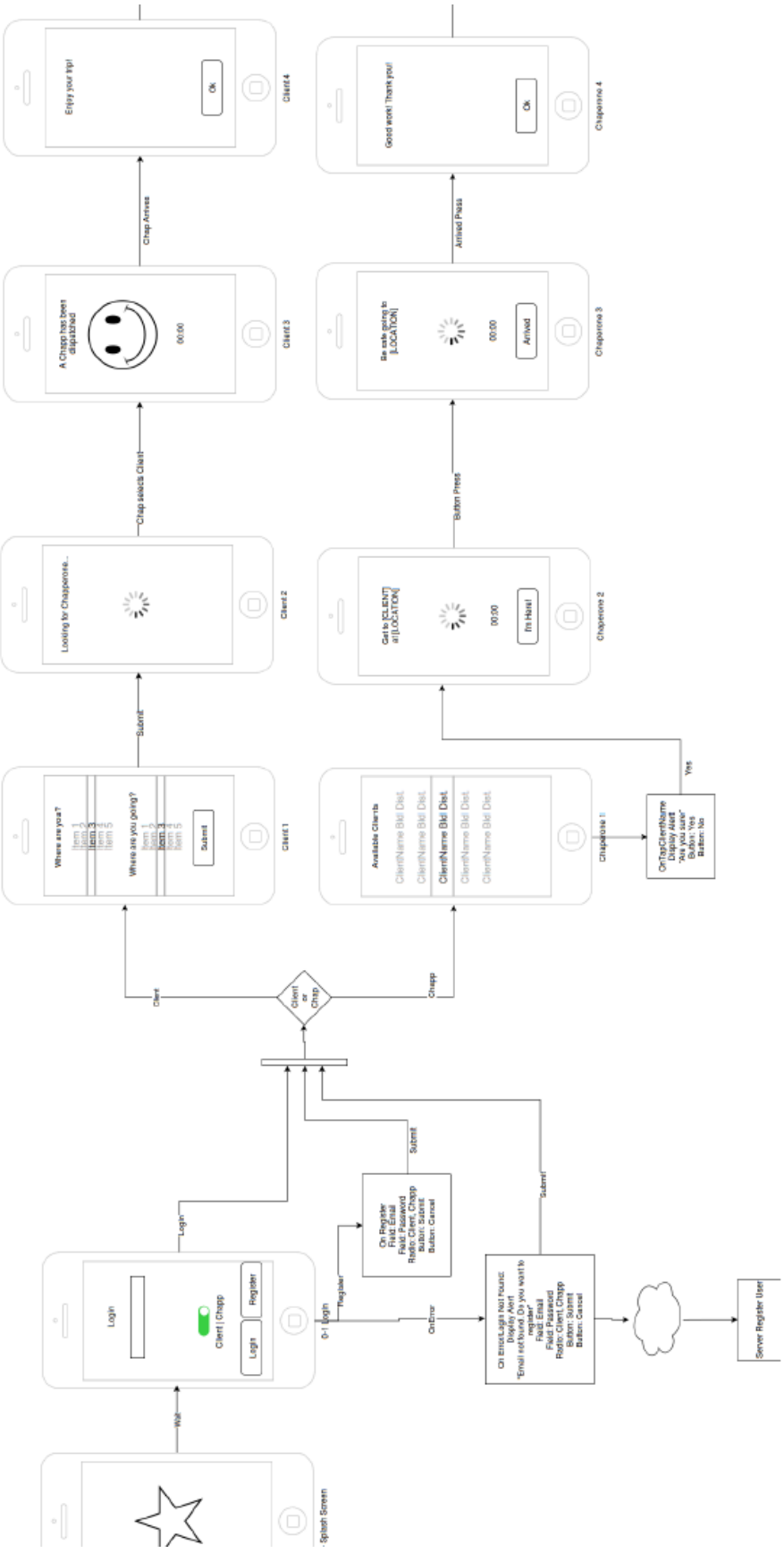
BI Tools- Tools for taking care of Chappy program.

Asnyc Messaging- Messaging to allow for a fluid presentation to the user of the Chappy App while background process continue to run.



BASIC ARCHITECTURE

WIREFRAMES


Splash Screen
Wait
Login
Register
Client | Chapp
Login
0-1
Register
On Register
Field: Email
Field: Password
Radio: Client, Chapp
Button: Submit
Button: Cancel
Submit
OnError
On Error:Login Not Found
Display Alert
"Email not found. Do you want to register"
Field: Email
Field: Password
Radio: Client, Chapp
Button: Submit
Button: Cancel
Submit
Server Register User
Client or Chapp
Client
Chapp
Client 1
Where are you?
Item 1
Item 2
Item 3
Item 4
Item 5
Where are you going?
Item 1
Item 2
Item 3
Item 4
Item 5
Submit
Submit
Client 2
Looking for Chaperone...
Chap selects Client
Client 3
A Chapp has been dispatched
00:00
Chap Arrives
Client 4
Enjoy your trip!
Ok
Chaperone 1
Available Clients
ClientName Bid Dist.
ClientName Bid Dist.
ClientName Bid Dist.
ClientName Bid Dist.
ClientName Bid Dist.
On ChapClientName
Select Alert
"Are you sure"
Button: Yes
Button: No
Yes
Button Press
Chaperone 2
Go to [CLIENT] at [LOCATION]
00:00
I'm Here!
Chaperone 3
Be safe going to [LOCATION]
00:00
Arrived
Arrived Press
Chaperone 4
Good work! Thank you!
Ok

CHAPPY USE CASES

User Registration:

1. User loads app
2. App checks cache for existing account, finds none, requests user to login.
3. User enters e-mail address, toggles radio button indicating chaperone or client, hits 'Register button.
4. App sends object to the web service, which checks whether the e-mail address is already registered in the server.
5. If not registered, inserts user information into the database and returns an object back to the app containing required data.
6. If registered, send error back to app indicating that the email address is already registered.

User Login:

1. User loads app.
2. User enter email address, toggles whether they are a client or chaperone, and presses 'Login' button.
3. App sends message to web service, which checks to see if the email address is in the chosen (client or chaperone) database list.
4. If the email address in found in the database, proceed to next use case based on whether use is a client or chaperone.
5. If the email address is not found in the database, alert the user that the email address was not found.

Client requests chaperone:

1. Client presses 'Request Chaperone' button.
2. Another view comes into focus, client selects what building they are in from a drop-down list and what building they want to go to from a drop-down list, and presses 'Submit' button.
3. App sends a new transaction request to web service which creates a new transaction in the database.
4. App polls the web service every x seconds to check if a chaperone has accepted the transaction.
5. When a chaperone is found, change view to display the distance in which the chaperone is away from the client.

Chaperone checks for open clients/transactions:

1. Chaperone opens client, logs in as a chaperone.
2. App sends message to the web service to check for any open transactions on the server and send back a list. Sends an empty list if there are no open transactions.

Chaperone accepts a client:

1. Chaperone selects an open client from the list of open client transactions.

2. App sends a message to the web service with the selected transaction and the web service updates the transaction on the server to indicate a chaperone has selected the client.

Chaperone arrives at client pickup location:

1. Chaperone and client meet up.
2. Chaperone presses 'Arrived at Client' Button
3. Client App sends message to web service telling it to up update current transaction arrival date in the server.

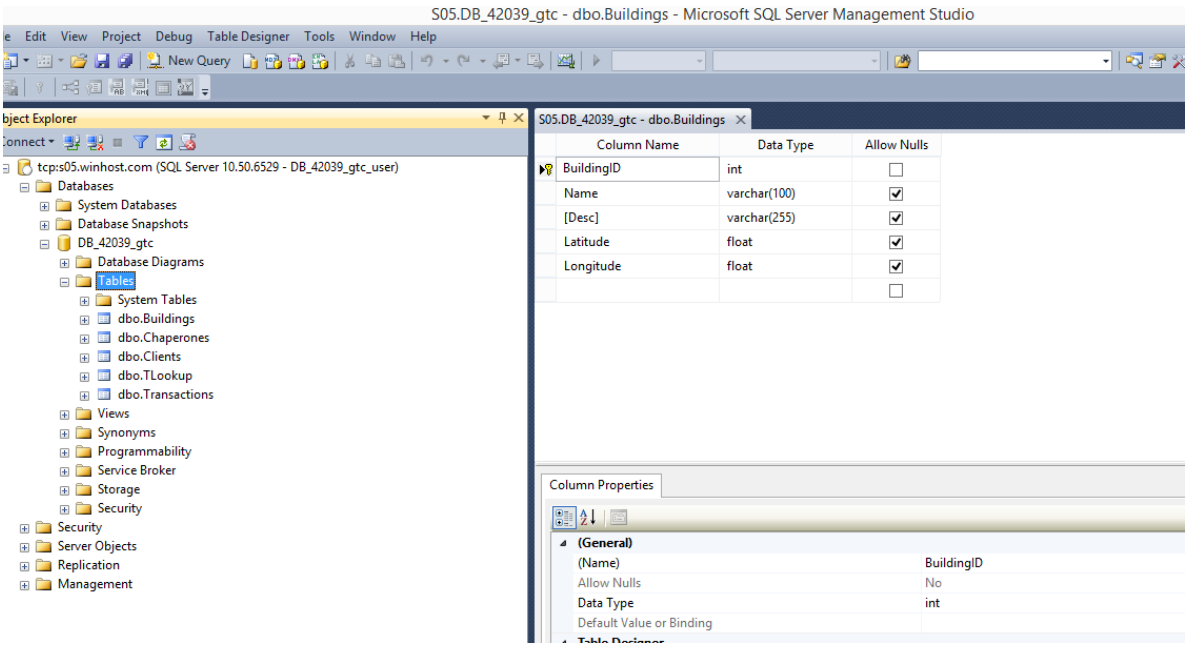
Chaperone/Client arrive at destination:

1. Chaperone/client arrive at destination.
2. Chaperone presses 'Arrived at Destination' button, sends a message to the web service.
3. The web service updates the transaction in the database.

BACK END (DATABASE)

SQL Server Management Studio

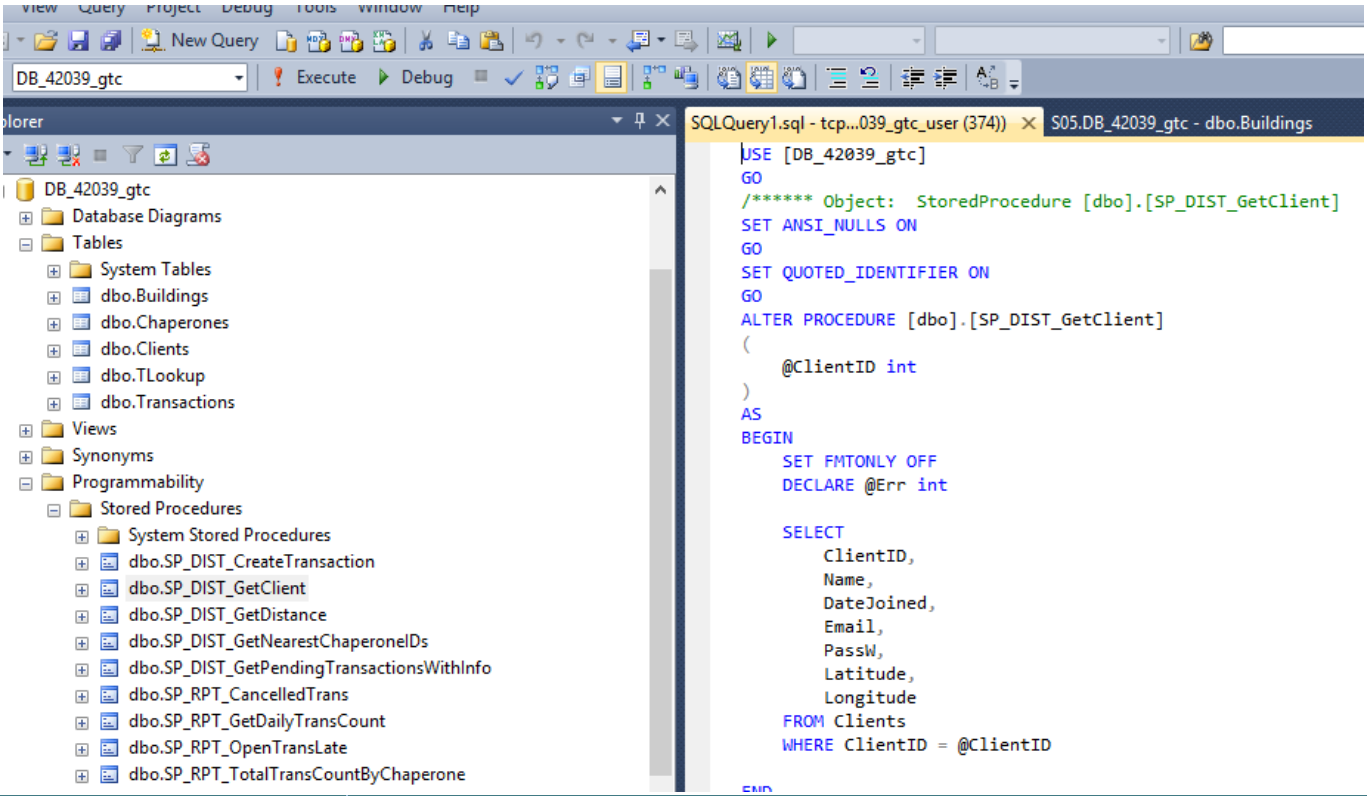
Although this SQL Server would not technically be included in the web portion of this project, I would like to briefly go over how we created our database and allow the Database team to



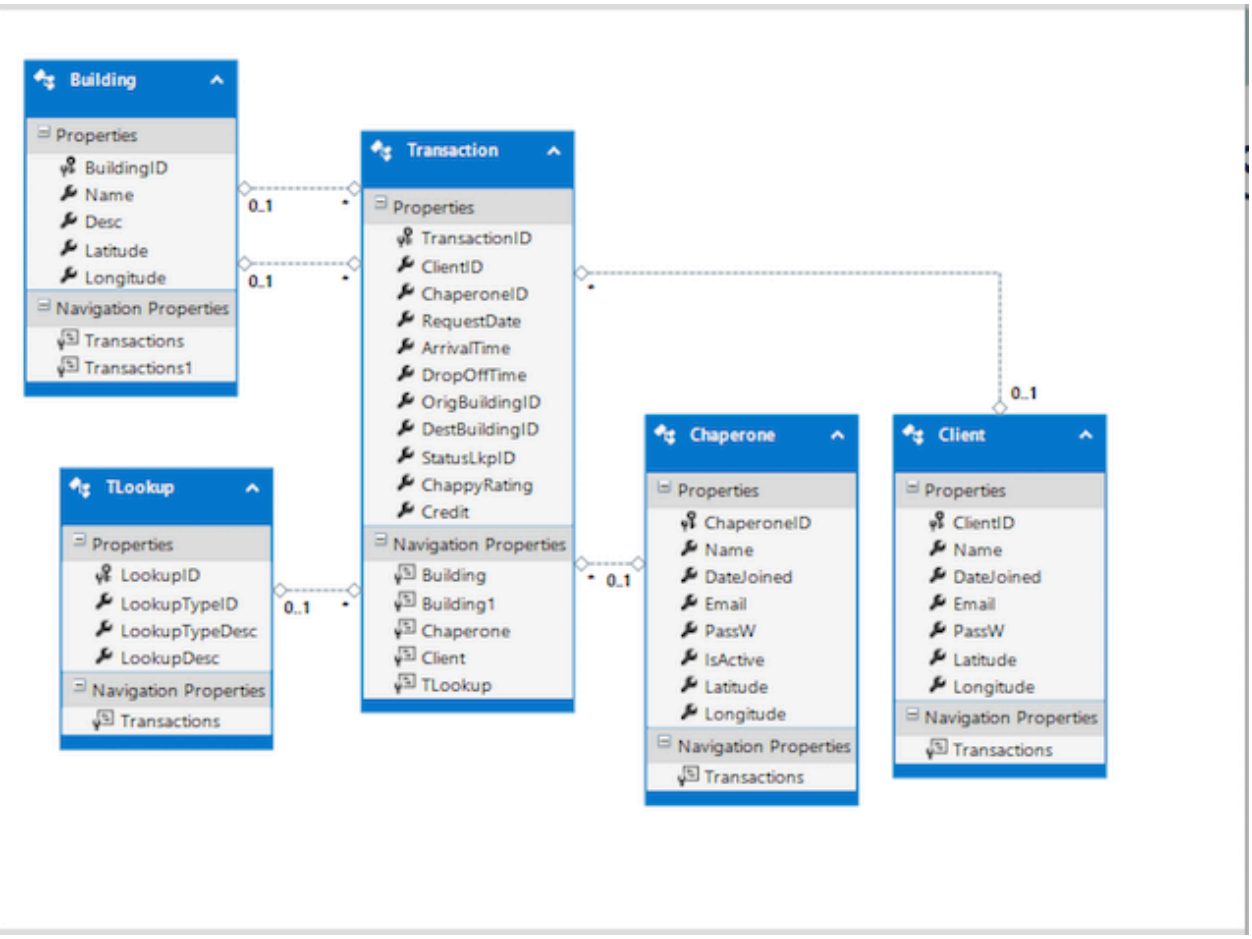
explain in more depth.

We first generated the tables by writing scripts and also were able to learn how to use the GUI tools to generate tables and create relationships. Above is an image of the layout of SQL Server. The management studio makes it easy to view the design of each table and allows you to easily alter the table. As you click different options it generates the scripts to perform the action desired.

After the tables were built and we were able to predict some of the data that would needed to be used for the applications and created stored procedures to specific data. The advantage of using stored procedures, is that the data being pulled is cohesive amongst all platforms, the queries are reusable, and the code is already pre-compiled for enhanced performance time. Below is a picture of one of the stored procedures we created.



Tables


Building
Properties
BuildingID
Name
Desc
Latitude
Longitude
Navigation Properties
Transactions
Transactions1
Transaction
Properties
TransactionID
ClientID
ChaperoneID
RequestDate
ArrivalTime
DropOffTime
OrigBuildingID
DestBuildingID
StatusLkpID
ChappyRating
Credit
Navigation Properties
Building
Building1
Chaperone
Client
TLookup
TLookup
Properties
LookupID
LookupTypeID
LookupTypeDesc
LookupDesc
Navigation Properties
Transactions
Chaperone
Properties
ChaperoneID
Name
DateJoined
Email
PassW
IsActive
Latitude
Longitude
Navigation Properties
Transactions
Client
Properties
ClientID
Name
DateJoined
Email
PassW
Latitude
Longitude
Navigation Properties
Transactions
0..1
*
0..1
*
0..1
*
*
0..1
0..1

MIDDLE TIER (WEB SERVICES)

The main operations of Chappy are executed by a Web service. The architecture of the service consist of five projects:

- **Chappy_DAL** – ORM for the database.
- **Chappy_Model** – DTOs for the service.
- **Chappy_Model.Extensions** – Helper classes to convert classes in the ORM to DTOs.
- **Chappy_Service** – Holds the main service interface and its implementation.
- **Chappy_Host** – Contains Logic that hosts the service on a remote IIS server.

Chappy_DAL

DAL stands for Data Access Layer. Our goal was to create a clean bridge between the Chappy database and the service methods. This was achieved by the Object Relational Mapper (ORM) provided by .NET called the Entity Framework. The entity framework creates classes that represent the databases itself and it classes that represent the tables within the database. Instead of using the ADO.Net library and explicitly opening a connection to the database and executing queries, the classes that the Entity Framework created executes these task for you. However, The Chappy web service will be communicating to other devices remotely via passing JSON objects. The objects that the Entity Framework create has too much code for them to be an ideal object to serialize and send. Also, these classes are auto generated. If one adds serialization code to the entities, it will be erased after an update to the database. This is where the project Chappy_Model comes in.

Chappy_Model

This project contains classes which are called POCOs – in relation to ORMs and DALs, POCO stands for 'Plain Old C# Objects'. POCOs are to contain only properties that represent the data – no behavior. For these to be serialized, each class will have an attribute on its declaration header called `DataContract`. And on each of the class's property definitions will have an attribute named DataMember. These objects will be used as the arguments that will be required by our service methods and the return value for these methods. The more formal name of these is Data Transfer Objects (DTO). Since most of these DTOs are analogous to the classes that the ORM created, we must map from the ORM entity classes to the DTOs and vice versa. This is where the project Chappy_Model.Extensions comes in.

Chappy_Model.Extensions

This project contains helper classes that convert entity objects to DTO and vice versa. These helper classes are static classes that contain methods that use a feature of the C# called extension methods. These methods can be called by the entity or DTO objects as if these methods where members of those object allowing a very streamline coding experience. We were then ready for the service implementation itself.

Chappy_Service

The Chappy_Service is comprised of two main parts – The Interface and the service implementation.

The interface is decorated with the attribute of a ServiceContract, which indicates that the service can be interacted by authorized clients. Each method declaration inside the interface is tagged with the attribute of OperationContract which indicates that the method being declared is required with the ServiceContract. In addition, each method declaration includes a WebInvoke Attribute. The WebInvoke attribute will use the RequestFormat and ResponseFormat of JavaScript Object Notation (JSon) so that our web service can interact with clients no matter the platform.
For every method declared in our interface, we must implement that method and write the method definition. In general each method makes use of Microsoft Entity Framework using Language Integrated Query (LINQ) to retrieve and update data, and calling stored procedures using the Chappy_DAL library.

Each method that returns an Object, returns a DTO version of the database object. As noted above in the Chappy Model, these Objects are used for the client to communicate to the service and vice versa.

Chappy_Host

Chappy will be hosted on an IIS server. To achieve this we created another project that holds the .svc file which as host definition containing the language and name of the service. And the project contains the web.config which is an xml file with settings that specify how to connect to the database and the rules on how to send and receive messages from the Chappy_Sevice endpoint.

Chappy API Methods

DTO_<bool> **ValidEmail**(DTO_<string> email)

DTO_Client **RegisterClient**(DTO_RegisterRequest clientEmail)

DTO_Chaperone **RegisterChap**(DTO_RegisterRequest chapEmail)

List<DTO_Building> **GetBuildings**()

DTO_Transaction **CreateTransaction**(DTO_CreateTransactionRequest TxRequest)

DTO_Transaction **GetTransaction**(DTO_<int> transactionId)

List<DTO_Transaction> **GetOpenTransactions**()

DTO_Transaction **AcceptTransaction**(DTO_AcceptTransactionRequest AcceptRequest)

DTO_Transaction **ArrivedTransaction**(DTO_<int> TransactionID)

DTO_Transaction **DropOffTransaction**(DTO_<int> TransactionID)

DTO_Transaction **AddRatingToTransaction**(DTO_RatingRequest ratingRequest)

DTO_Transaction **ClientCancel**(DTO_<int> TransactionID)

DTO_Transaction **ChaperoneCancel**(DTO_<int> TransactionID)

DTO_Transaction **ClientPanic**(DTO_<int> TransactionID)

DTO_Transaction **ChaperonePanic**(DTO_<int> TransactionID)

void **UpdateCoordinatesClient**(DTO_CoordinatesSetRequest coordRequest)

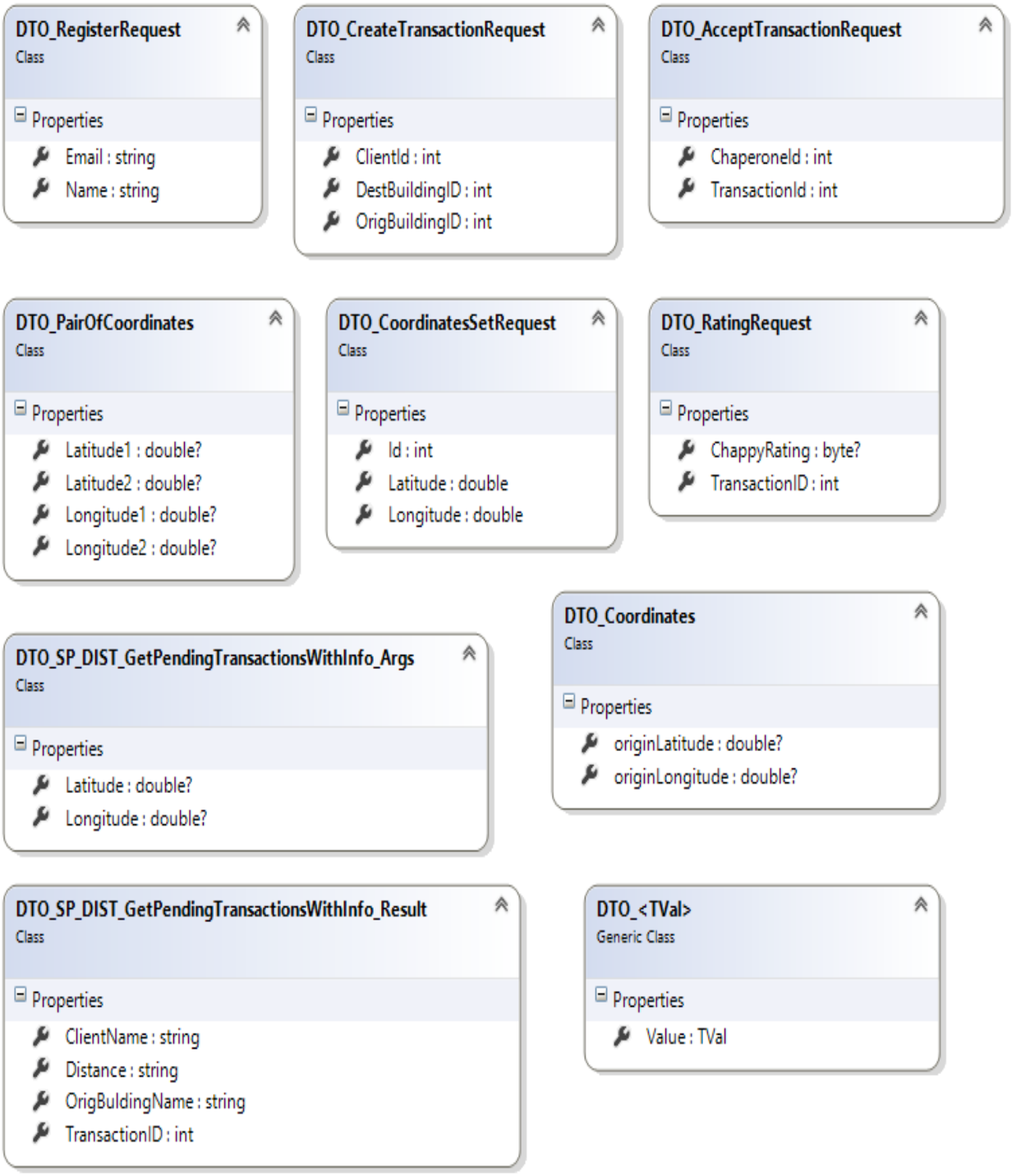
void **UpdateCoordinatesChaperone**(DTO_CoordinatesSetRequest coordRequest)

DTO_<double> **GetDistance**(DTO_PairOfCoordinates coordinates)

List<int> **GetNearestChaperoneIDs**(DTO_Coordinates coordinates)

List<DTO_SP_DIST_GetPendingTransactionsWithInfo_Result> **GetPendingTransactionsWithInfo**(DTO_SP_DIST_GetPendingTransactionsWithInfo_Args coordinates)

Chappy Complex Entity DTOs



Chappy_Model.Extension Code Snippet:

```
using Chappy_DAL;
using Chappy_Model;

namespace Chappy_Model.Extensions
{
    public static partial class DTO_Extensions
    {
        public static Chaperone ToDAL(this DTO_Chaperone dtoChaperone)
        {
            return new Chaperone
            {
                ChaperoneID = dtoChaperone.ChaperoneID ?? 0,
                Name = dtoChaperone.Name,
                Email = dtoChaperone.Email,
                PassW = dtoChaperone.PassW,
                DateJoined = dtoChaperone.DateJoined,
                IsActive = dtoChaperone.IsActive,
                Latitude = dtoChaperone.Latitude,
                Longitude = dtoChaperone.Longitude
            };
        }

        public static DTO_Chaperone ToDTO(this Chaperone chaperone)
        {
            return new DTO_Chaperone
            {
                ChaperoneID = chaperone.ChaperoneID,
                Name = chaperone.Name,
                Email = chaperone.Email,
                PassW = chaperone.PassW,
                DateJoined = chaperone.DateJoined,
                IsActive = chaperone.IsActive,
                Latitude = chaperone.Latitude,
                Longitude = chaperone.Longitude
            };
        }

        public static Chaperone Update(this Chaperone chaperone, DTO_Chaperone dtoChaperone)
        {
            chaperone.ChaperoneID = dtoChaperone.ChaperoneID ?? 0;
            chaperone.Name = dtoChaperone.Name;
            chaperone.Email = dtoChaperone.Email;
            chaperone.PassW = dtoChaperone.PassW;
            chaperone.DateJoined = dtoChaperone.DateJoined;
            chaperone.IsActive = dtoChaperone.IsActive;
            chaperone.Latitude = dtoChaperone.Latitude;
            chaperone.Longitude = dtoChaperone.Longitude;
            return chaperone;
        }

        public static Chaperone Update(this Chaperone chaperone, Chaperone otherChaperone)
        {
            chaperone.ChaperoneID = otherChaperone.ChaperoneID;
            chaperone.Name = otherChaperone.Name;
            chaperone.Email = otherChaperone.Email;
            chaperone.PassW = otherChaperone.PassW;
            chaperone.DateJoined = otherChaperone.DateJoined;
            chaperone.IsActive = otherChaperone.IsActive;
            chaperone.Latitude = otherChaperone.Latitude;
            chaperone.Longitude = otherChaperone.Longitude;
            return chaperone;
        }
    }
}
```

Chappy_Model.Extension Code Snippet:

```
using Chappy_DAL;
using Chappy_Model;

namespace Chappy_Model.Extensions
{
    public static partial class DTO_Extensions
    {
        public static Chaperone ToDAL(this DTO_Chaperone dtoChaperone)
        {
            return new Chaperone
            {
                ChaperoneID = dtoChaperone.ChaperoneID ?? 0,
                Name = dtoChaperone.Name,
                Email = dtoChaperone.Email,
                PassW = dtoChaperone.PassW,
                DateJoined = dtoChaperone.DateJoined,
                IsActive = dtoChaperone.IsActive,
                Latitude = dtoChaperone.Latitude,
                Longitude = dtoChaperone.Longitude
            };
        }

        public static DTO_Chaperone ToDTO(this Chaperone chaperone)
        {
            return new DTO_Chaperone
            {
                ChaperoneID = chaperone.ChaperoneID,
                Name = chaperone.Name,
                Email = chaperone.Email,
                PassW = chaperone.PassW,
                DateJoined = chaperone.DateJoined,
                IsActive = chaperone.IsActive,
                Latitude = chaperone.Latitude,
                Longitude = chaperone.Longitude
            };
        }

        public static Chaperone Update(this Chaperone chaperone, DTO_Chaperone dtoChaperone)
        {
            chaperone.ChaperoneID = dtoChaperone.ChaperoneID ?? 0;
            chaperone.Name = dtoChaperone.Name;
            chaperone.Email = dtoChaperone.Email;
            chaperone.PassW = dtoChaperone.PassW;
            chaperone.DateJoined = dtoChaperone.DateJoined;
            chaperone.IsActive = dtoChaperone.IsActive;
            chaperone.Latitude = dtoChaperone.Latitude;
            chaperone.Longitude = dtoChaperone.Longitude;
            return chaperone;
        }

        public static Chaperone Update(this Chaperone chaperone, Chaperone otherChaperone)
        {
            chaperone.ChaperoneID = otherChaperone.ChaperoneID;
            chaperone.Name = otherChaperone.Name;
            chaperone.Email = otherChaperone.Email;
            chaperone.PassW = otherChaperone.PassW;
            chaperone.DateJoined = otherChaperone.DateJoined;
            chaperone.IsActive = otherChaperone.IsActive;
            chaperone.Latitude = otherChaperone.Latitude;
            chaperone.Longitude = otherChaperone.Longitude;
            return chaperone;
        }
    }
}
```

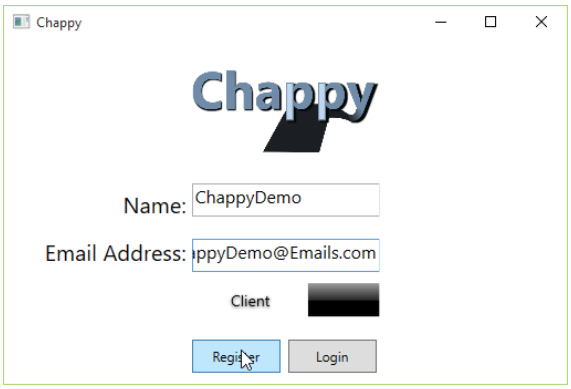
Chappy_Host Web.config Snippet

```
<system.serviceModel>
  <bindings>
    <webHttpBinding>
      <binding name="StreamedRequestWebBinding"
               openTimeout="10:15:00"
               receiveTimeout="10:15:00"
               sendTimeout="10:15:00"
               bypassProxyOnLocal="true"
               hostNameComparisonMode="WeakWildcard"
               maxBufferSize="2147483647"
               maxBufferPoolSize="2147483647"
               maxReceivedMessageSize="2147483647"
               transferMode="StreamedRequest"
               useDefaultWebProxy="false">
        <readerQuotas maxStringContentLength="2147483647"
                      maxArrayLength="2147483647" />
      </binding>
    </webHttpBinding>
  </bindings>
  <behaviors>
    <endpointBehaviors>
      <behavior name="webBehavior">
        <webHttp />
      </behavior>
    </endpointBehaviors>
    <serviceBehaviors>
      <behavior name="ServiceBehvr">
        <serviceMetadata httpGetEnabled="true"
                         httpsGetEnabled="true" />
        <serviceDebug includeExceptionDetailInFaults="false" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <services>
    <service name="Chappy_Service.ChappyService"
             behaviorConfiguration="ServiceBehvr">
      <endpoint address=""
                behaviorConfiguration="webBehavior"
                binding="webHttpBinding"
                bindingConfiguration="StreamedRequestWebBinding"
                contract="Chappy_Service.IChappyService"></endpoint>
      <endpoint address="mex"
                binding="mexHttpBinding"
                contract="IMetadataExchange"></endpoint>
    </service>
  </services>
  <protocolMapping>
    <add binding="basicHttpsBinding"
         scheme="https" />
  </protocolMapping>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true"
                             multipleSiteBindingsEnabled="true" />
</system.serviceModel>
```

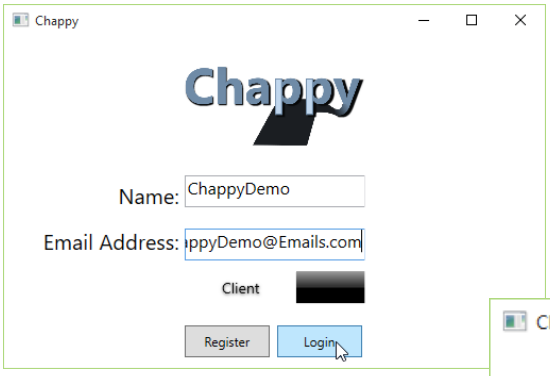
FRONT TIER (UI)

Windows Desktop

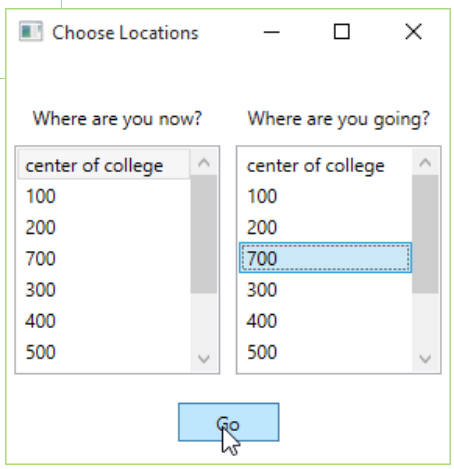
Register Client



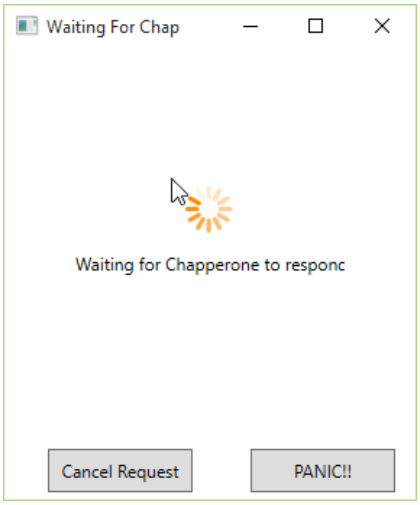
Login Client



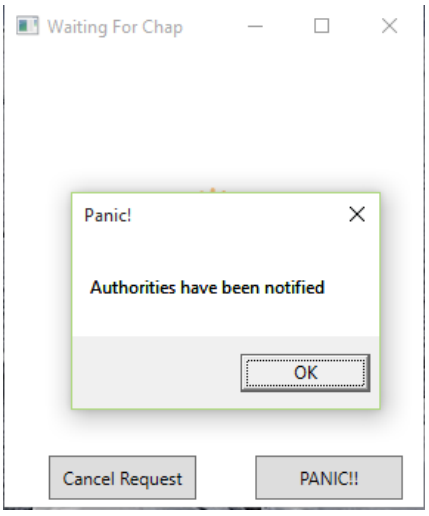
Choose Origin/Destination



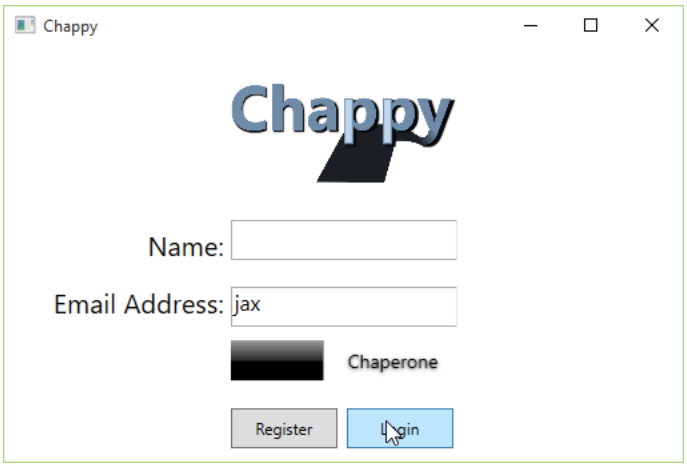
Wait for Chaperone



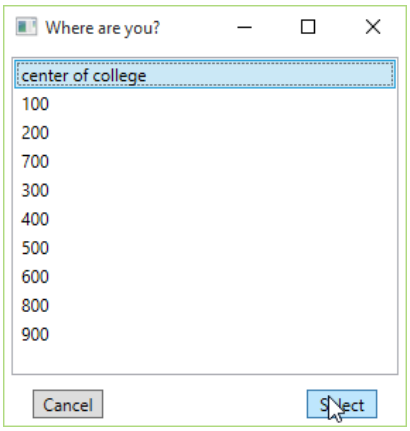
Panic!



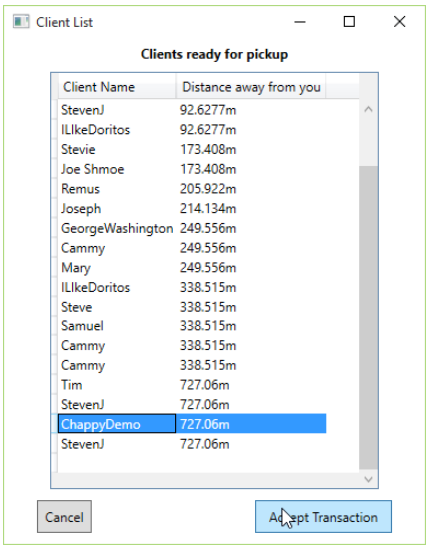
Chaperone Login



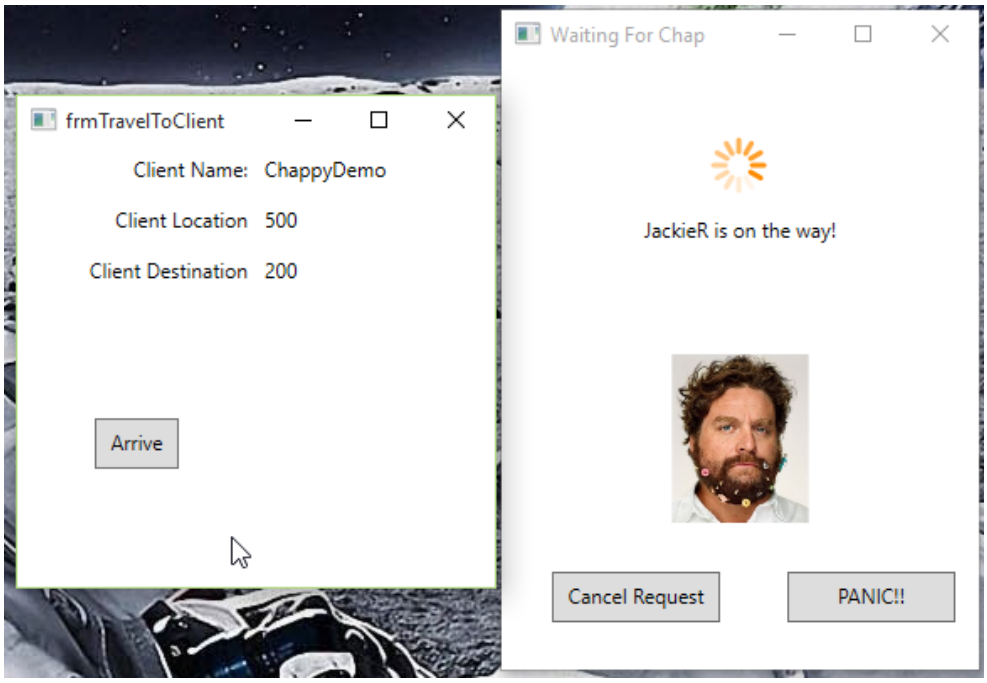
Select starting location



Choose Client



Client accepted, shown on both screens



Web Site http://chappyadmin.azurewebsites.net/

Website Overview

This Chappy website is for administrative purposes only. It currently displays reports about Chaperones, Clients, Buildings and Open Transactions. Users are able to view reports containing all records or narrow into the detail of one specific record.

The website is located at:
http://chappyadmin.azurewebsites.net/

Construction

This site is an ASP.NET 5 Web Application built using Visual Studio 2015 and uses MVC to support the basic CRUD operations (create, read, update, delete). The site has been designed to be responsive, using Twitter Bootstrap, and the layout will adjust based on the user's device.

Since most tables have more columns than will fit on one screen in a smaller device, they have been made responsive as well. This allows the user to scroll horizontally through the table in order to view all fields and have only the table scroll, instead of the entire screen.

Future Enhancements

The ability create, edit and deactivate record for all sections.

Sample Code

Below are samples of the code for the Chaperone pages:

ChaperoneID	Name	Email	PassW	DateJoined	IsActive	Latitude	Longitude	
1	StevieBeez	buzzdabe@b.com	buzz	10/17/2015 5:06:00 AM	1	80.974407	-84.069027	Details
2	JohnnyElway	denver@elway.com	football	10/17/2015 5:08:18 AM	0			Details
3	Cole	chelmut2313@student.gwinnetttech.edu	1234	10/22/2015 8:19:03 PM	1	33.964407	-84.069027	Details

Chaperone Controller

```
// GET: Chaperone
public ActionResult Index()
{
    ViewData["Message"] = "None at this time.";

    ChappyService.DTO_Chaperone[] obj = ServiceClient.GetChaperones();
    return View(obj);
}
```

Chaperone View

Populates page with Chaps

```
@foreach (var item in Model)
{
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.ChaperoneID)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Email)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.PassW)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.DateJoined)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.IsActive)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Latitude)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Longitude)
        </td>
        <td>
            @Html.ActionLink("Details", "Details", new { id = item.ChaperoneID })
        </td>
    </tr>
}
```

Selected Chaperone

Clicking the link on the Chaperone Index page gets the ID of the selected Chap and passes it to Details in the controller.

ChaperoneID	Name	Email	PassW	DateJoined	IsActive	Latitude	Longitude	
1	StevieBeez	buzzdabe@b.com	buzz	10/17/2015 5:06:00 AM	1	80.974407	-84.069027	Details

```
<td>
    @Html.ActionLink("Details", "Details", new { id = item.ChaperoneID })
</td>
```

ChaperonController

```
// GET: Chaperone/Details/5
public ActionResult Details(int id)
{
    ViewData["Message"] = "ImageURL is not yet included.";

    ChappyService.DTO_Chaperone obj = ServiceClient.GetChaperone(new ChappyService.DTO_Ofint { Value = id });
    return View(obj);
}
```

Selected Chaperone Page Source (Partial)

```
<dl class="dl-horizontal">
    <dt>
        @Html.DisplayNameFor(model => model.ChaperoneID)
    </dt>

    <dd>
        @Html.DisplayFor(model => model.ChaperoneID)
    </dd>

    <dt>
        @Html.DisplayNameFor(model => model.Name)
    </dt>

    <dd>
        @Html.DisplayFor(model => model.Name)
    </dd>
```

Selected Chaperone Web Page

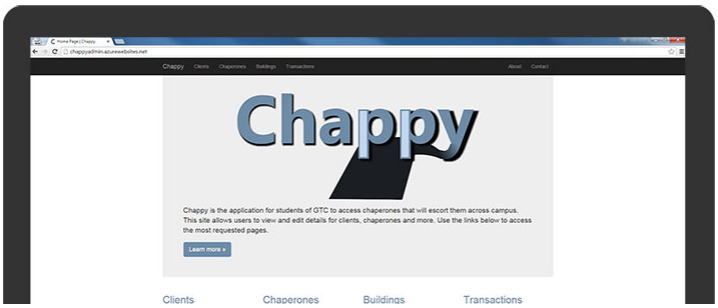
ChaperoneID	1
Name	StevieBeez
Email	buzzdabe@b.com
PassW	buzz
DateJoined	10/17/2015 5:06:00 AM
IsActive	1
Latitude	80.974407
Longitude	-84.069027

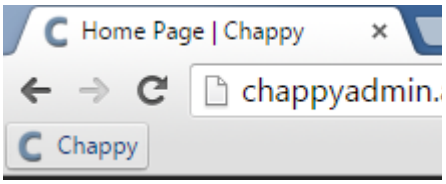
Responsive Design Features

Menu changes when the device size is too small to legibly view the full-width menu.

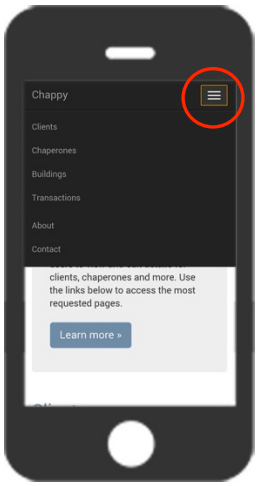
Branding

The Chappy logo is prominently displayed on the home page and a favicon add in order to easily locate the tab that Chappy is open on as well as to quickly locate in their bookmarks folder or bookmarks bar.

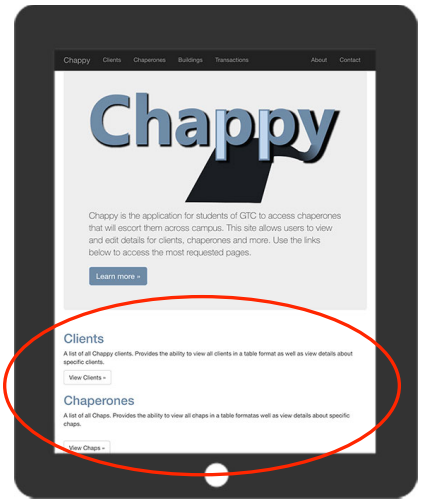




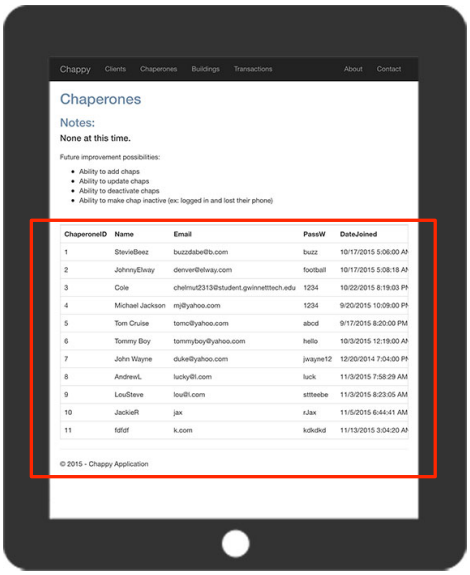
Responsive Menus

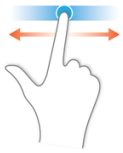


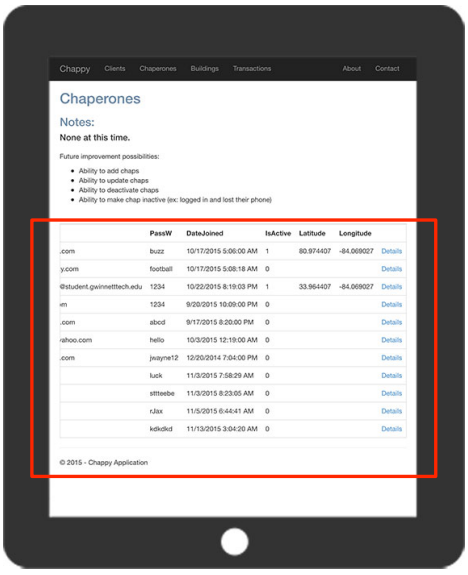
Responsive Layout



Responsive Tables



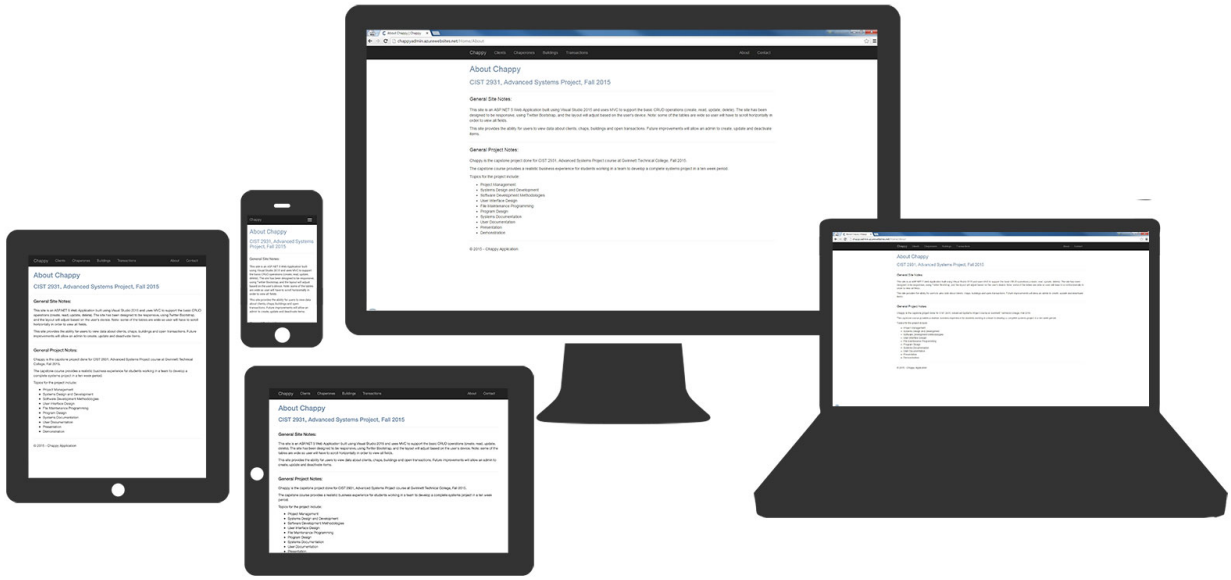




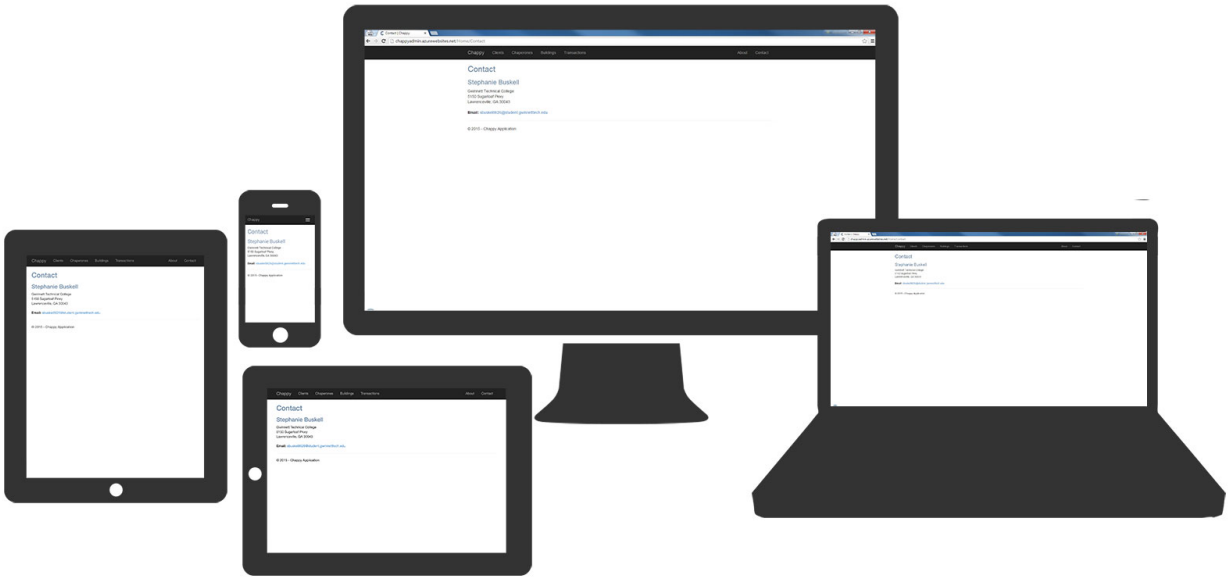
Screenshots

Home

About

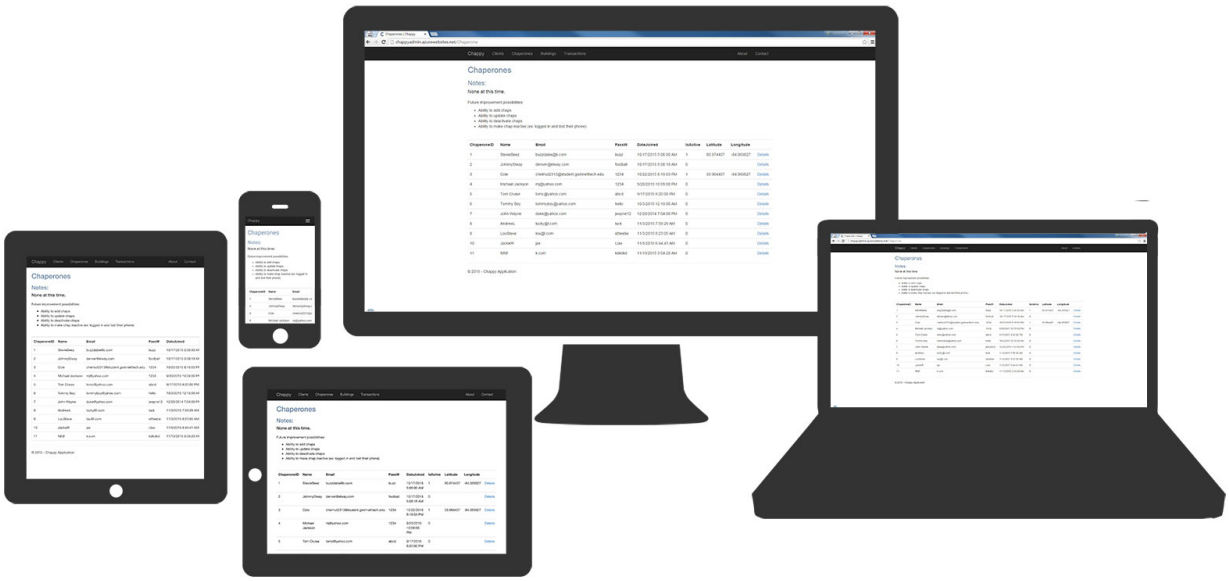


Contact



Chaps

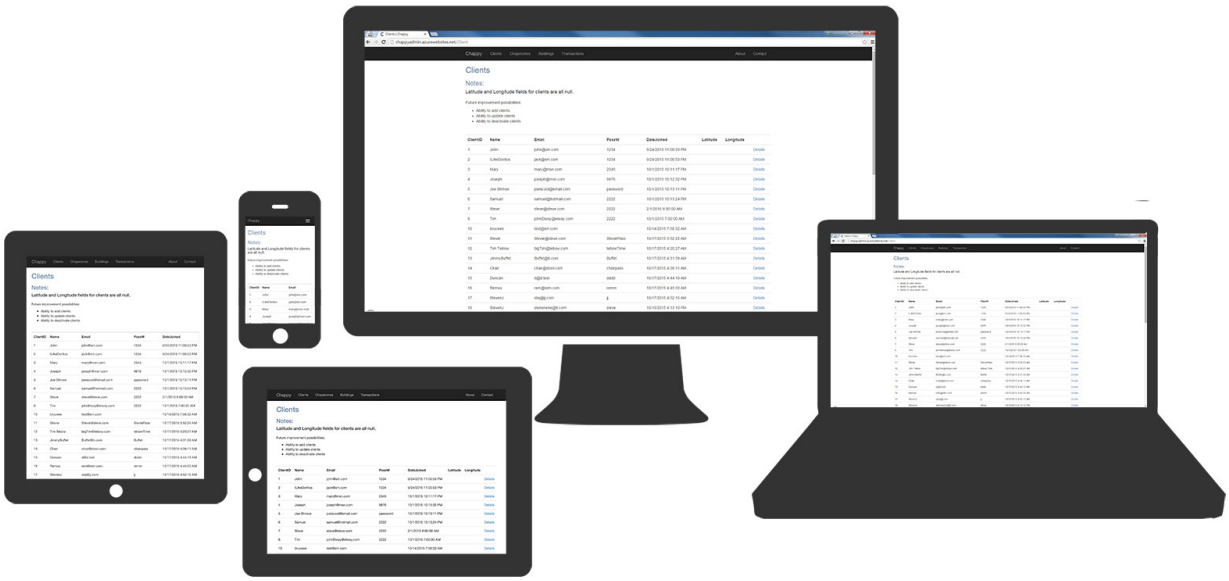
All Chaps



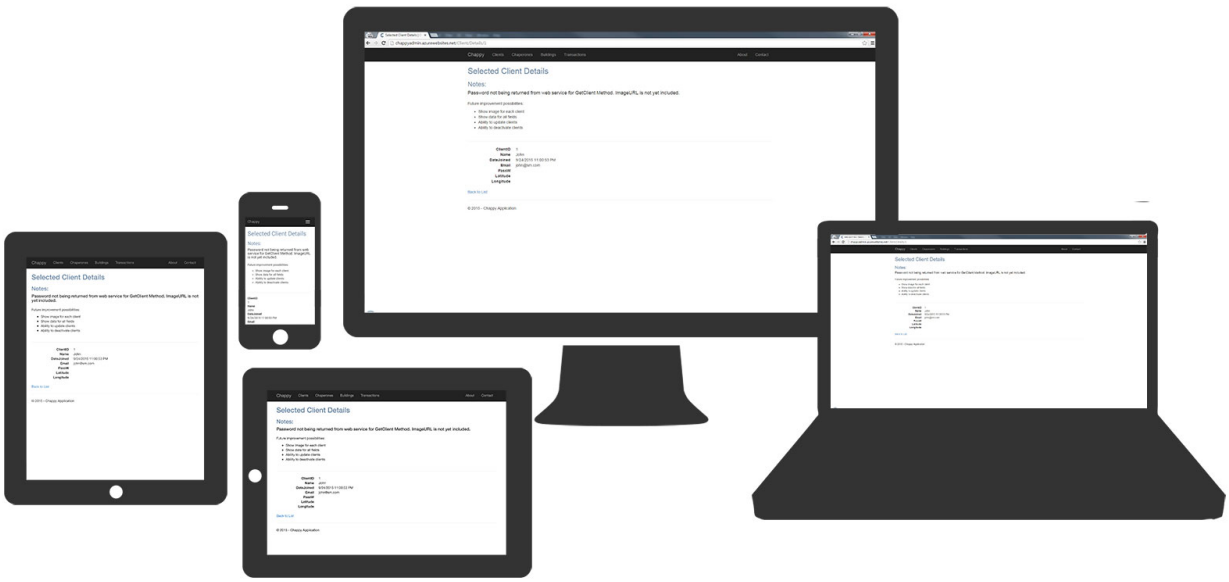
Chap Details

Clients

All Clients

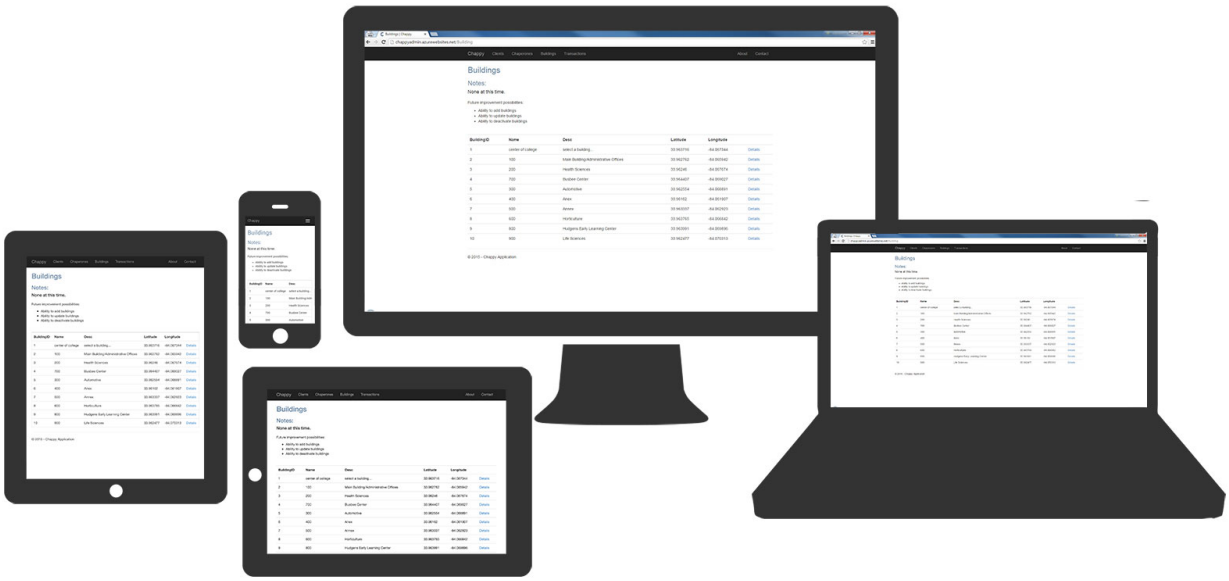


Client Details



Buildings

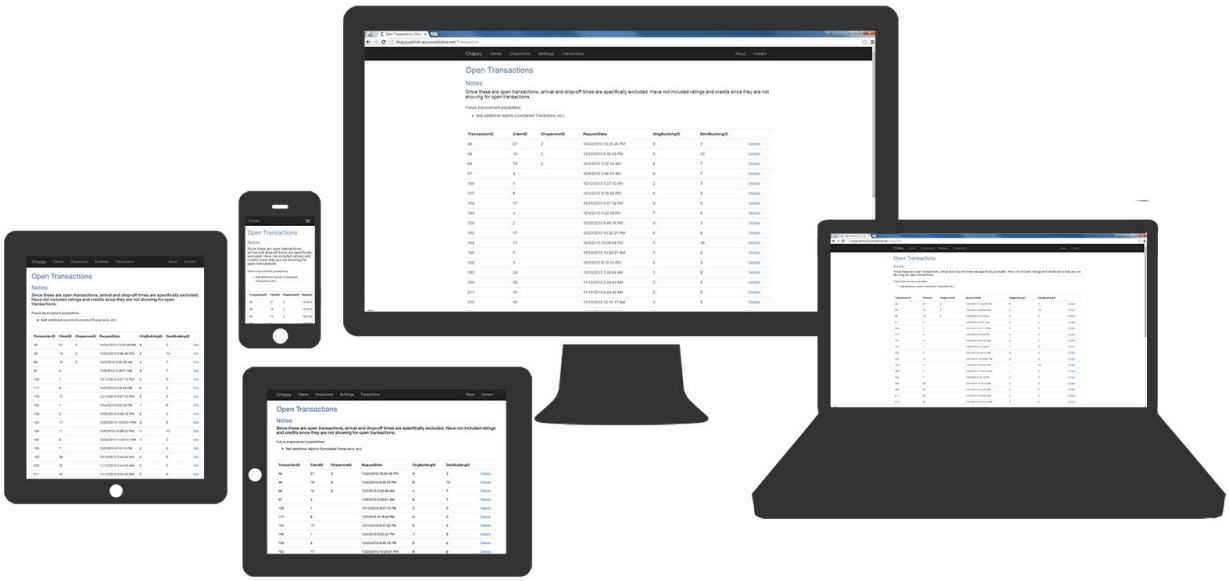
All Buildings



Building Details

Transactions

All Open Transactions

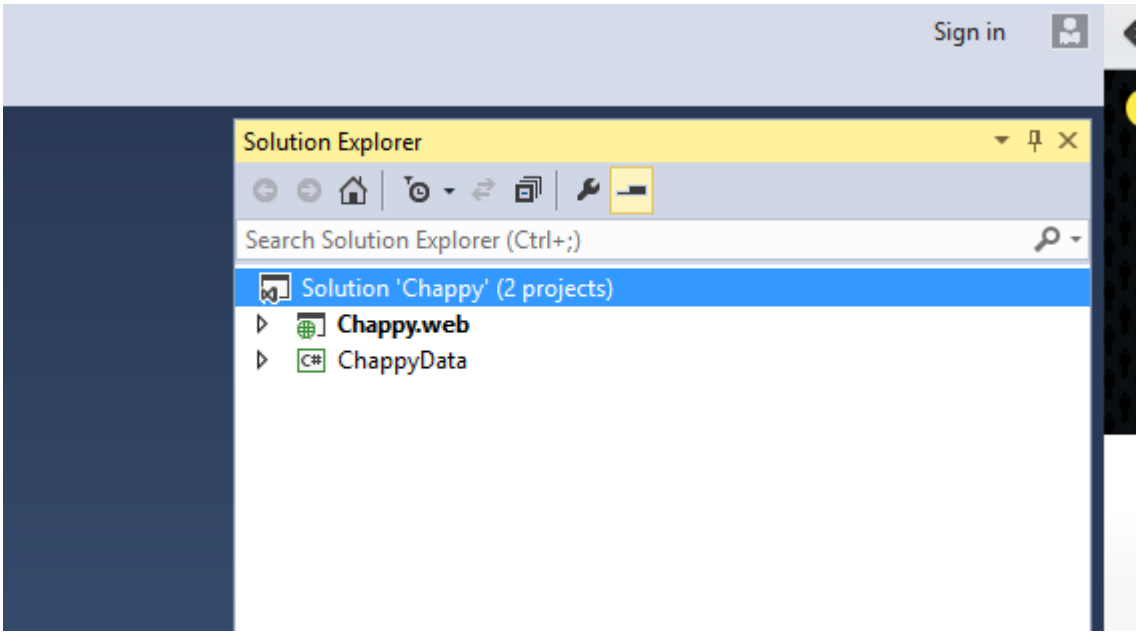


Transaction Details

MVC 5.0 supporting Web Site

Microsoft Visual Studio

We used Microsoft Visual studio as our IDE. Using visual studio creates a clean and organized environment to create projects. This solution only consisted of two projects but Visual studio enabled us to use all of the technologies and frameworks needed to create our web application. Below is a picture of our

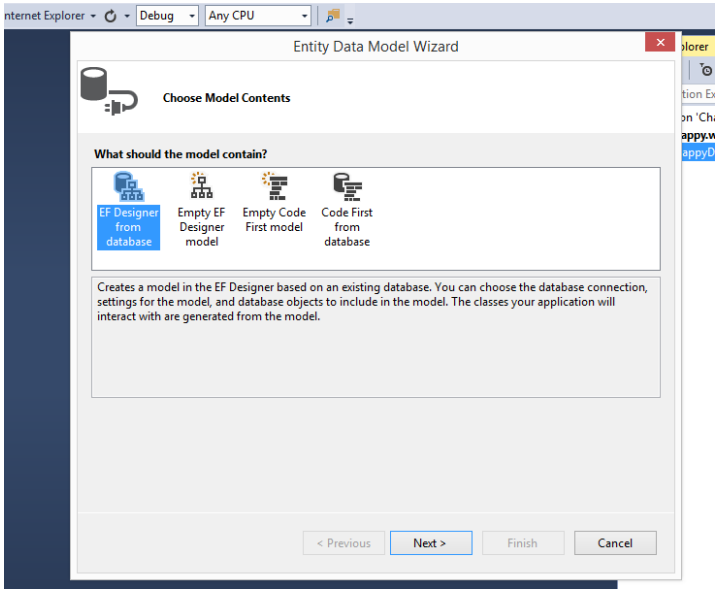


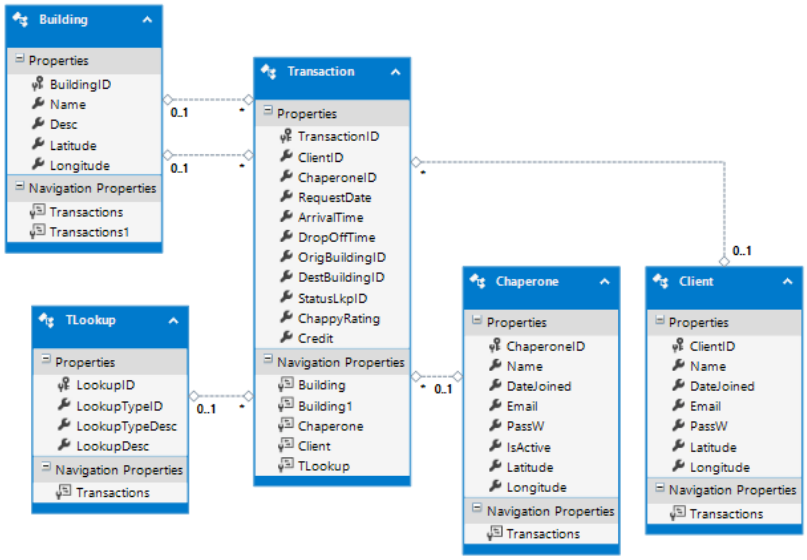
projects and the layout in visual studio.

As you can see it is a fairly simple layout. We have included two projects: ChappyData and Chappy.web. Chappy data is the project we have used to set up entity framework and access the database. Chappy.web is our MVC web application that uses ChappyData as a reference to access the data.

Entity Framework

Entity Framework is an open source object-relational mapping framework. We used entity framework in ChappyData. Visual Studio could not make this easier to set up. When you simply add an ADO.NET Entity Data Model and a wizard, pictured to the right, walks you through the options to connect to your database. Assuming that the database and relationships are correctly created, Entity framework then generates all the classes needed from the tables along with their relationships. Below is an image of the EDMX diagram that shows all of the tables and



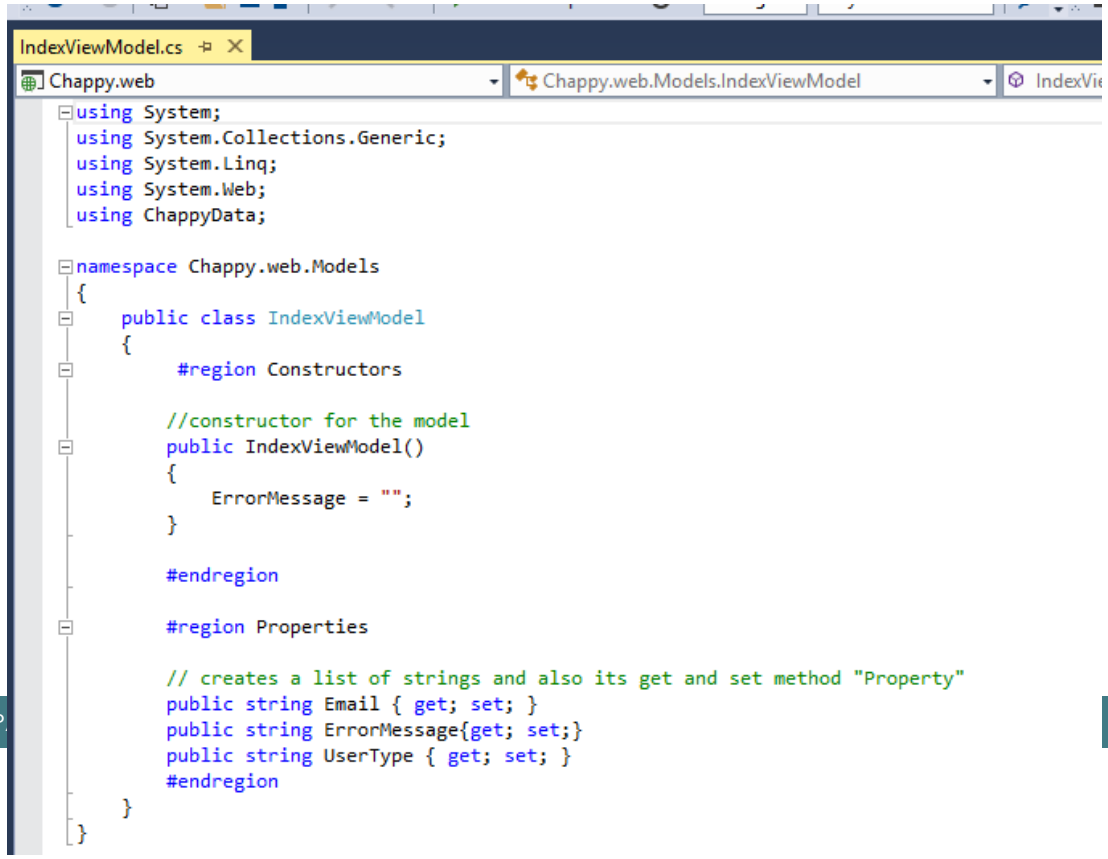


their corresponding relationships.

ASP.Net MVC

MVC is a framework for building web application. The idea of MVC is to separate logic in a way to keep the code easily readable and organized. MVC stands for model, view, and controller, which are the three logic layers. The model is primarily used as the business layer and used in this project to supply the views with the dynamic data needed. The view is the front end layer and incorporates html, CSS and JQuery to create a GUI for the user. The controller is used in this application to navigate throughout the application, validate data and update the database.

Below is the model for the index page of Chappy. The index page for Chappy was a very simple log in screen, so there was not much data required. The model is written just as any C# class is and contains empty properties that will be populated in the



index view login form.

Below is the Index view that populates the previous model. As you can see most of this page just consist of HTML tags like any other page. At the top we included a reference to the model class that we will be using. We then populate the model by using a razor form and assign input values to the properties by using @Model.*Property* . When the form is submitted we send the post to the “Index” controller action of the “Home” controller which

Index.cshtml

```
@model Chappy.web.Models.IndexViewModel
@Styles.Render("~/Content/Site.css")
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Index</title>
</head>
<body>
    <div class="homeIndexWrapper">
        <div class="indexContent">
           @using (Html.BeginForm("Index", "Home", FormMethod.Post, new { @class= "login_loginForm", @id ="login" } ))
           {
               <h4 class="login_form_headers">Log Into Chappy</h4>
               <div>
                   <label class="login_form_labels">Email:</label>
                   @Html.TextBoxFor(m => m.Email, new { @class = "indexEmailInput" })
               </div>
               <div>
                   <label class="login_form_labels">User Type:</label>
                   @Html.RadioButtonFor(m => m.UserType, "Client", new { @class = "indexRadio" }) Client
                   @Html.RadioButtonFor(m => m.UserType, "Chaperone", new { @class = "indexRadio" }) Chaperone
               </div>
               if (Model.ErrorMessage != "")
               {
                   <div class="error_Message">@Model.ErrorMessage</div>
               }
                   <p class="login_form_rows center">
                   <button type="submit" class="signUpBtn">Log In</button> <button type="button" id="indexSignUpBtn" class="signUpBtn">SignUp</button> <a href='
               </p>
           }
       </div>
    </div>

    <script type="text/javascript" src="~/Scripts/jquery-1.10.2.min.js"></script>
    <script type="text/javascript" src="~/Scripts/jquery-ui-1.8.24.min.js"></script>
    <script type="text/javascript" src="~/Scripts/ChappyScripts/Index.js"></script>
```

automatically sends the populated model by model binding.

Below is the Index controller actions. There is an overloaded controller action so when the Index is called without a model it simply returns an empty model to supply an index view. After the form post it send a model as a parameter to it calls the second index with a populated model. We now have all the input data and are able to perform the necessary actions here and when finish

```
[HttpGet]
public ActionResult Index()
{
    //how to connect to web service
    //ChappyServiceClient svc = new ChappyServiceClient();
    //DTO_Ofint clientId = new DTO_Ofint();
    //clientId.Value = 2;
    //DTO_Client client = svc.GetClient(clientId);

    IndexViewModel model = new IndexViewModel();
    return View("Index", model);
}
//After form is submitted overloaded controller action is called with the model as a parameter
[HttpPost]
public ActionResult Index(IndexViewModel model)
{
    DB_42039_gtcEntities db = new DB_42039_gtcEntities();

    if (model.Email != null)
    {
        if (model.UserType == "Chaperone")
        {
            Chaperone user = db.Chaperones.FirstOrDefault(c => c.Email == model.Email);
            if( user == null)
            {
                model.ErrorMessage = "No Chaperone with that Email not Found";
                return View("Index", model);
            }
            else
            {
                int userId = user.ChaperoneID;
                Session["userType"] = model.UserType;
                Session["userId"] = userId;
                ChaperoneHomeViewModel chaperoneModel = new ChaperoneHomeViewModel();
                return View("ChaperoneHome", chaperoneModel);
            }
        }
        else
```

we can insatiate different model and return it with a different view to navigate to different pages.

Although commented out, we show how we can use our Webservice created by our classmates to use the web method GetClient by sending a clientId.

CSS

We used Cascading Styling sheets to make the webpages more appealing. You can change the appearance and location of different HTML elements by using Classes and ID's. Below is a portion our CSS used throughout the web application.

```
/*---------Home Index (LOGIN PAGE)----------*/
.homeIndexWrapper { width: 100%; height: 100%; background-color: #e0d3d3; padding-top: 100px;  }
    .indexContent {  margin-top: 200px; margin-left: auto; margin-right: auto; width: 500px; border: 2px solid; border
    .indexContent a { text-decoration: none; padding-left: 90px; }
         .login_loginForm { padding-left: 70px; }
            .indexEmailInput { width: 250px; margin-bottom: 10px; }
            .login_form_headers{ font-size: 20px;  }
            .error_Message { text-align: center; color: red;}


/*---------User Registration (SIGNUP PAGE)----------*/
.registrationWrapper { width: 100%; height: 100%; background-color: #e0d3d3; padding-top: 100px;  }
    .registrationContent { margin-top: 200px; margin-left: auto; margin-right: auto; width: 500px; border: 2px solid;
        .registration_form { padding-left: 88px; }
            .register_form_headers { font-size: 20px;}
            .register_form_labels { display: inline-block; width: 100px;}
            .signUpBtn { padding: 10px 15px; background: #4479BA; color: #FFF; -webkit-border-radius: 4px; -moz-border
            .error_Message { text-align: center; color: red; }


/*---------Client Home (Clients HomePage after login)----------*/
.clientHome_Wrapper { width: 100%; height: 100%; background-color: #333131; padding-top: 20px;  }
    #userNameBanner { font-size: 30px; color: #e0d8d8; }
    .clientHome_Content { width: 700px; margin-left: auto; margin-right: auto; }
        .buildingsLabels { display: inline-block; width: 200px; }
        .btnStyle { cursor: pointer; }
        #tabs-3, #tabs-2, #tabs-1 { padding-left: 100px;}


/*---------Chaperone Home (Chaperone HomePage after login)----------*/
.chaperoneHome_Wrapper { width: 100%; height: 100%; background-color: #333131; padding-top: 20px;  }
    #userNameBanner { font-size: 30px; color: #e0d8d8; }
```

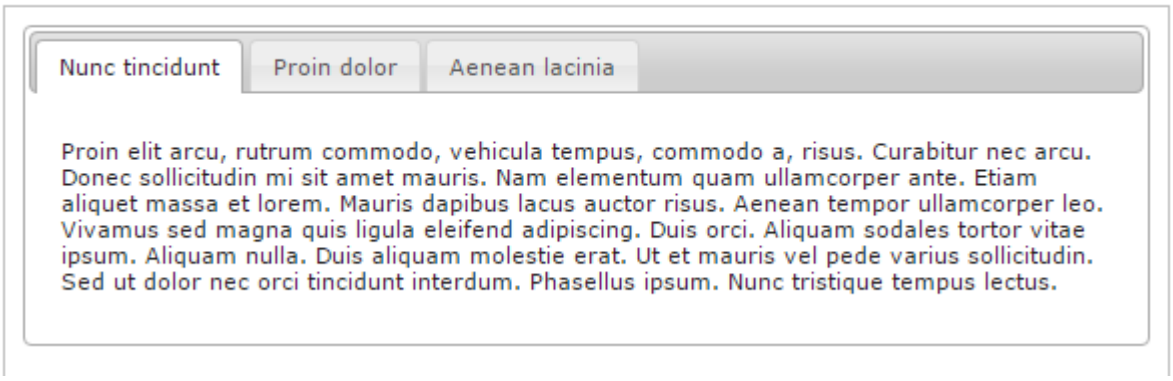
JQuery

We incorporated JQuery UI in our web application to display a view in a more appealing way. By adding a reference to JQuery UI it automatically applies CSS and javascript functionality to your website to create the tabs at the bottom.

lientHome.js | Site.css | HomeController.cs | Index.cshtml

```
$(document).ready(function () {

    $('#signOut').click(function () {
        window.location.href = "/home/Index";
    });

    $('#updateProfile').click(function () {
        window.location.href = "/home/UpdateClientInfo";
    });

    $(function () {
        $("#tabs").tabs();
    });


});
```

A single content area with multiple panels, each associated with a header in a list.



Android Development Document

***Purpose**: To outline the main problems encountered during android development of the Chappy application and their associated workarounds.*

- Problem 1:

Q: How do we connect to the Service methods that the middle tier created?

A: To accomplish this, a class titled "Connection" was created. This class extends the android API's "AsyncTask" class. Utilizing this class had two main benefits:

1. To establish a connection to the server that won't take too long for the user.
2. To allow updating of the UI without worrying about thread access that could quite possibly crash the entire application. For example: (Trying to access the main application thread after you've manually created a new thread to connect to the server)

Inside the "Connection" class, we had to create an instance of Java's "HttpUrlConnection" class to connect to the server. After that, it was a matter of setting the connection to use the Http web "Post" method, and telling it to use JSON as the data transfer type. Finally, to connect to specific service methods, the "Connection" class's execute method (inherited from "AysncTask") took in two strings:

1. String one: The url of the Chappy_Host.svc location along with the name of the method.
2. String two: This string lets the "Connection class know which connection method to call when the execute() method is invoked. (e.g.: "GetBuildings")

Inside of each of the "Connection" class' connection methods, the connection is either wrote to using Java's "OutputStreamWriter" class, read from using java's "BufferedReader" class, or both.

- Problem 2:

Q: How do we translate the raw JSON returned from the connection or write JSON to the connection?

A: Reading JSON: Read the data into an instance of "StringBuilder". Next, if the connection method only returns one JSON object, use android's "JSONOBJECT" class with the StringBuilder variable in the constructor to retrieve each field from the string. (e.g.: obj.getInt("ClientID"))

If the connection returns a collection of json objects, do the same, but split the string into an array of strings at every closing curly brace ("}"). Next, remove all of the slashes, brackets, and quotation marks. Lastly, append each closing curly brace back on into each string index before attempting to read each JSON object's data.

Writing JSON: If the service method expects a primitive data type, the JSON format must look something like "{"Value":1}". On the other hand, if the parameter wants a class object, the JSON format must include each of the class's properties represented as individual JSON objects. (e.g. :{"ClientID":1}{"Name":"Ricky"}etc..)

- Problem 3:

Q: How do we share resources between the different views of the application?

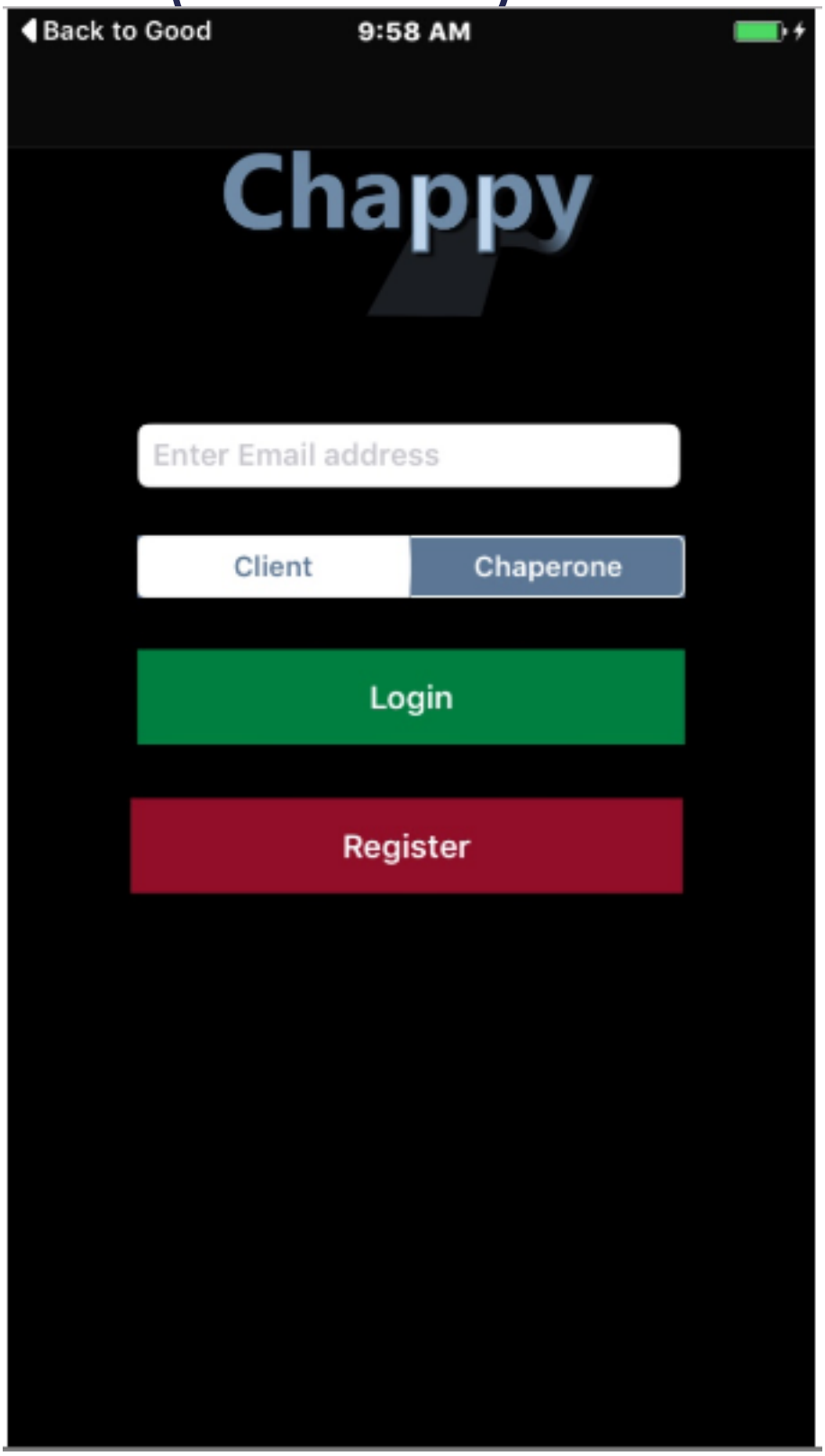
A: To do this, a Singleton class was created titled, "ActiveSession". This class stores data such as the active Client class, active Chaperone Class, and connection parameter strings.

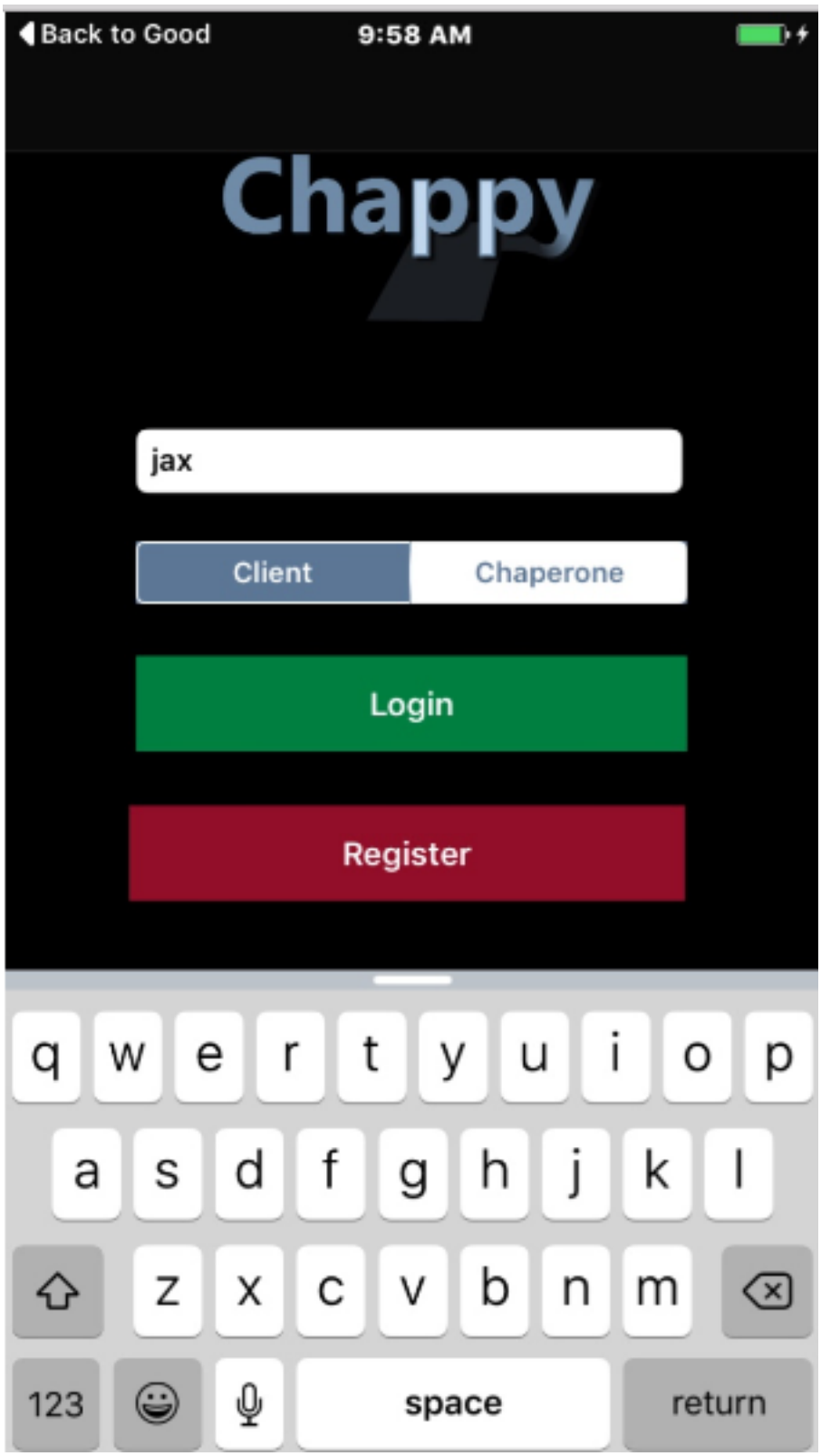
- Problem 4:

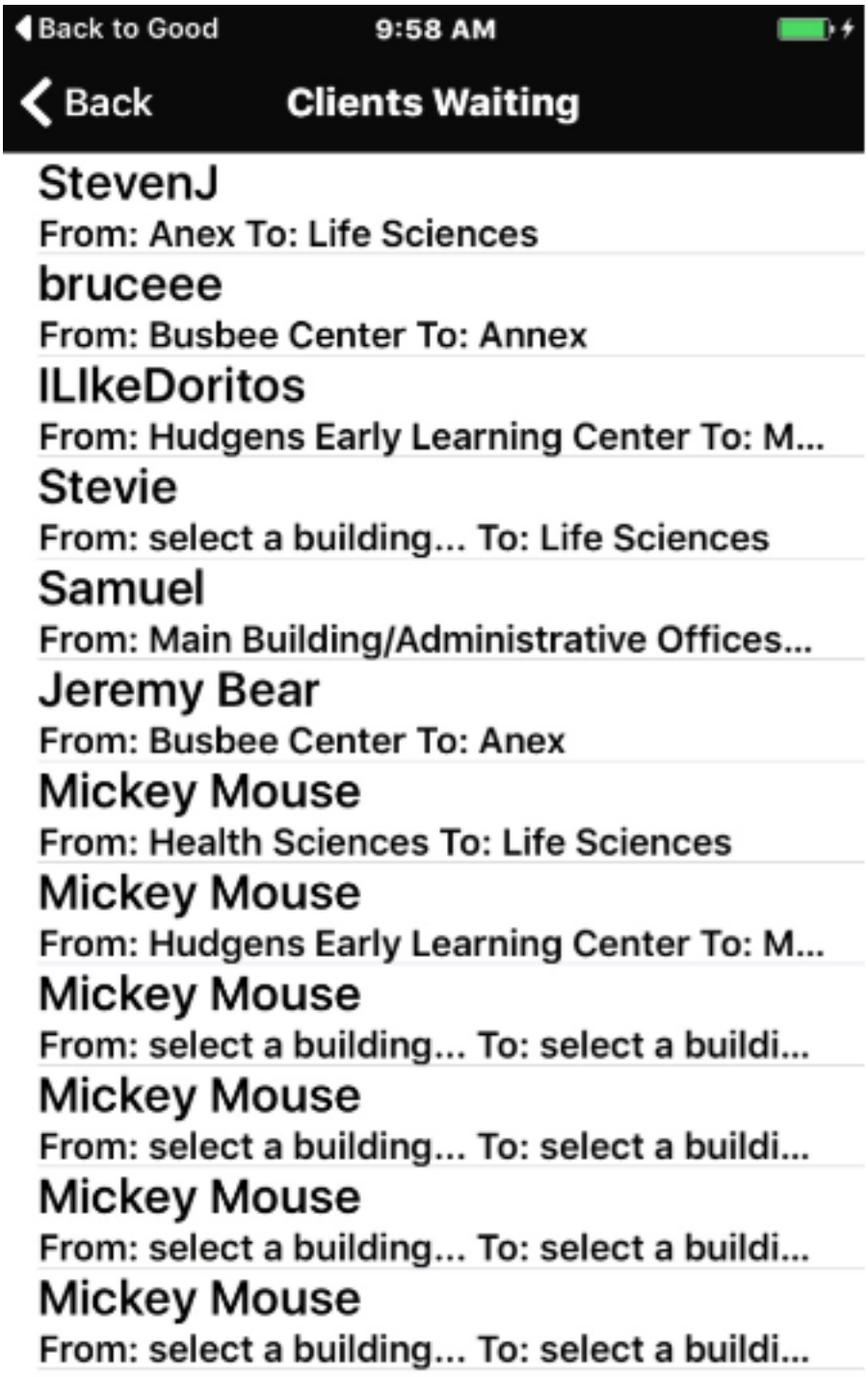
Q: How do we get the GPS location of a client or chaperone?

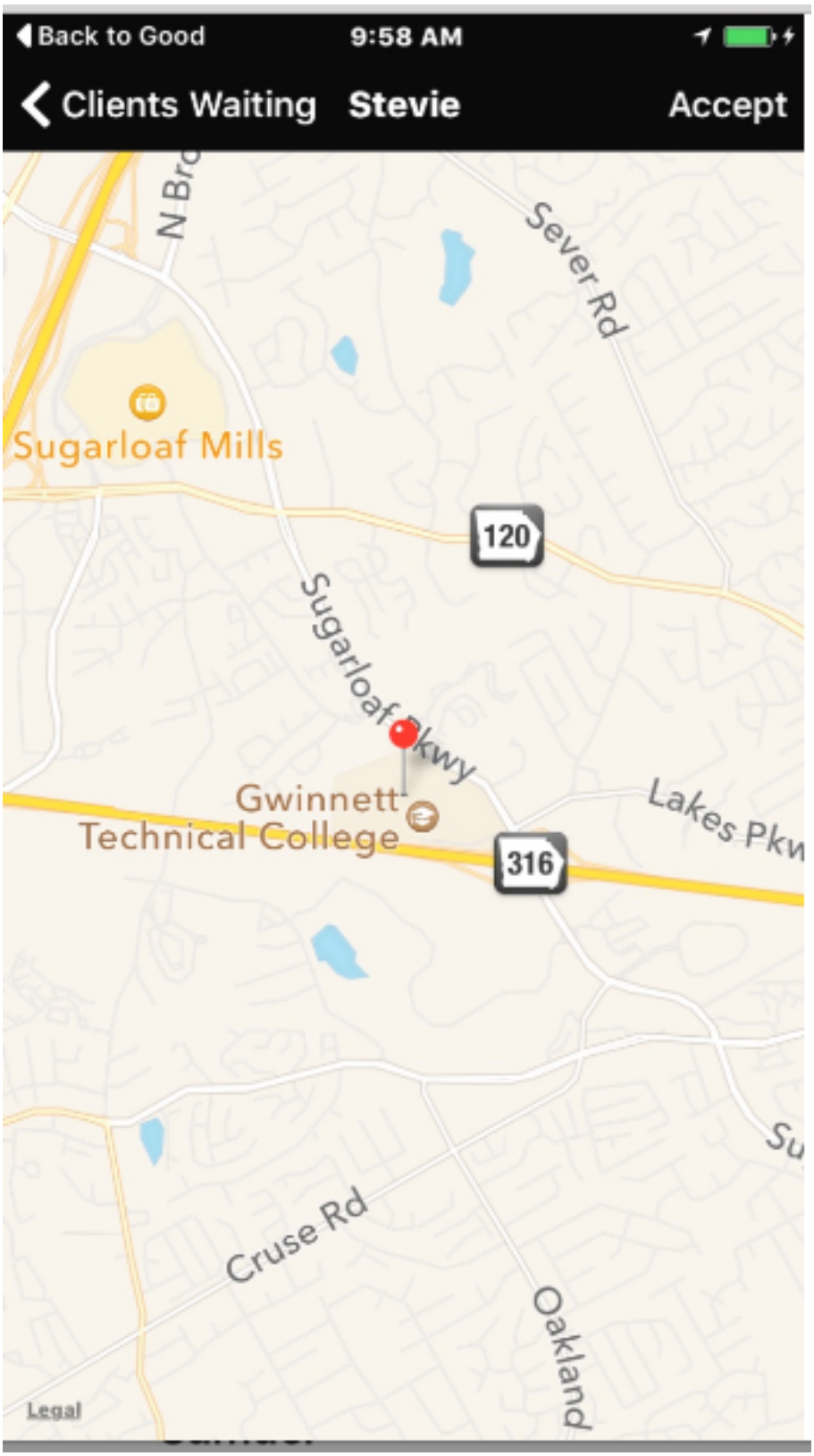
A: Google's API was used for this problem. It was found that Google's GPS functionality is slightly more accurate, and updates locations more frequently than android's native GPS methods.

IOS (IPHONE)

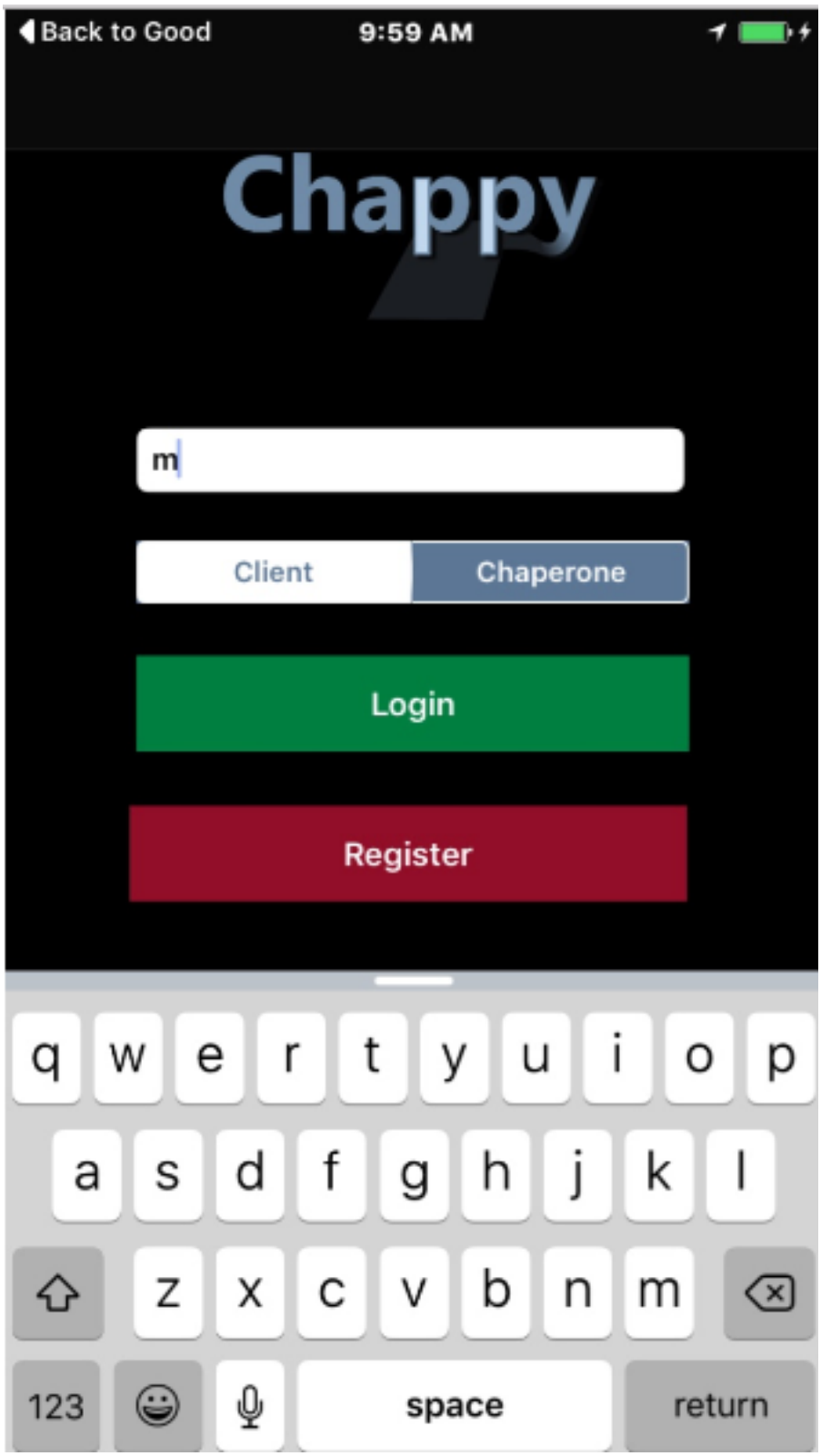

Back to Good
9:58 AM
Chappy
Enter Email address
Client
Chaperone
Login
Register


Back to Good
9:58 AM
Chappy
jax
Client
Chaperone
Login
Register
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
return

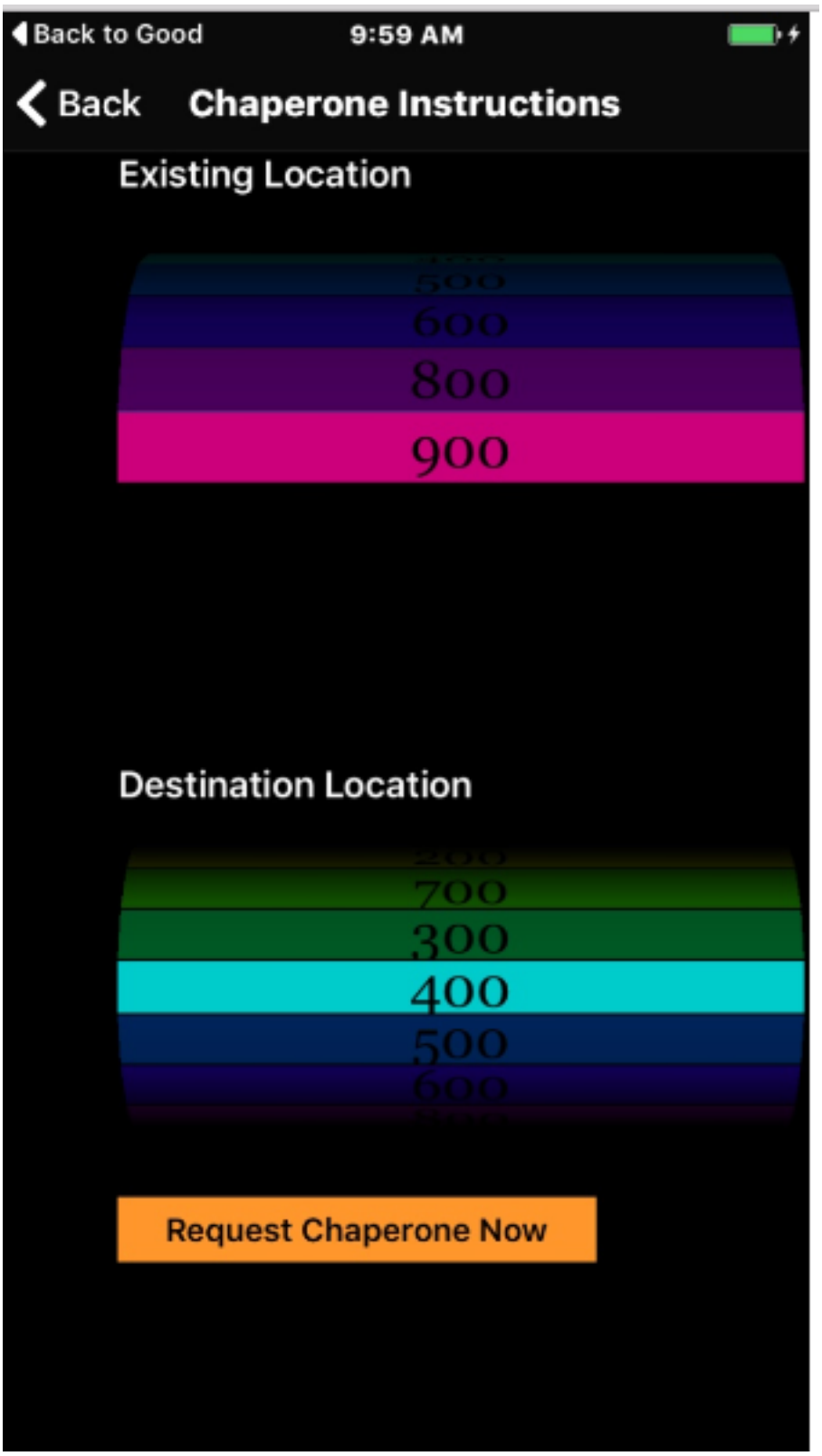

Back to Good
9:58 AM
Back
Clients Waiting
StevenJ
From: Anex To: Life Sciences
bruceee
From: Busbee Center To: Annex
lLIkeDoritos
From: Hudgens Early Learning Center To: M...
Stevie
From: select a building... To: Life Sciences
Samuel
From: Main Building/Administrative Offices...
Jeremy Bear
From: Busbee Center To: Anex
Mickey Mouse
From: Health Sciences To: Life Sciences
Mickey Mouse
From: Hudgens Early Learning Center To: M...
Mickey Mouse
From: select a building... To: select a buildi...
Mickey Mouse
From: select a building... To: select a buildi...
Mickey Mouse
From: select a building... To: select a buildi...
Mickey Mouse
From: select a building... To: select a buildi...

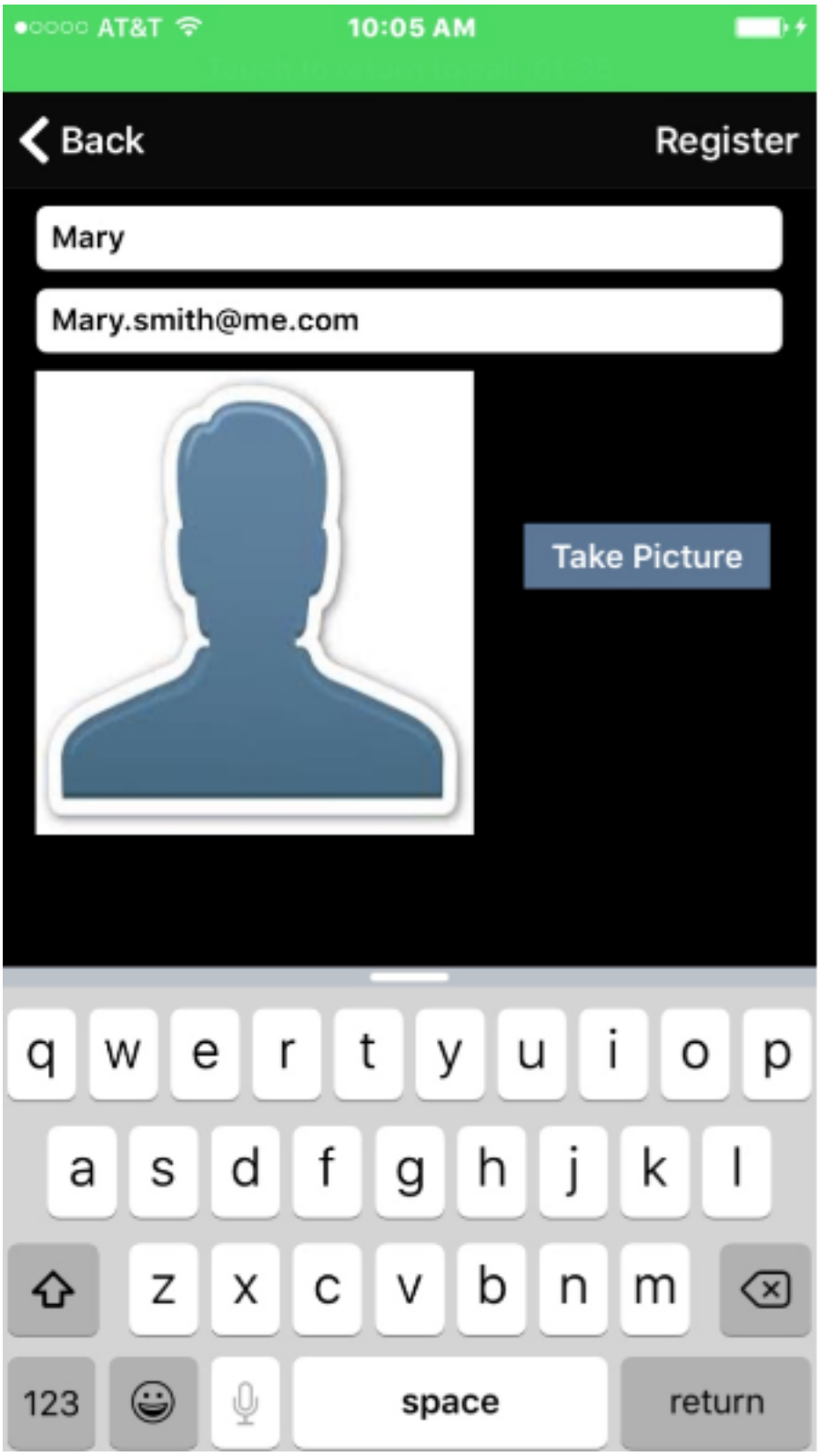

Back to Good
9:58 AM
Clients Waiting
Stevie
Accept
Sever Rd
Sugarloaf Mills
120
Sugarloaf Pkwy
Gwinnett
Technical College
316
Lakes Pkwy
Cruse Rd
Oakland
Legal


Back to Good
9:59 AM
ILIkeDoritos
Arrived at Client

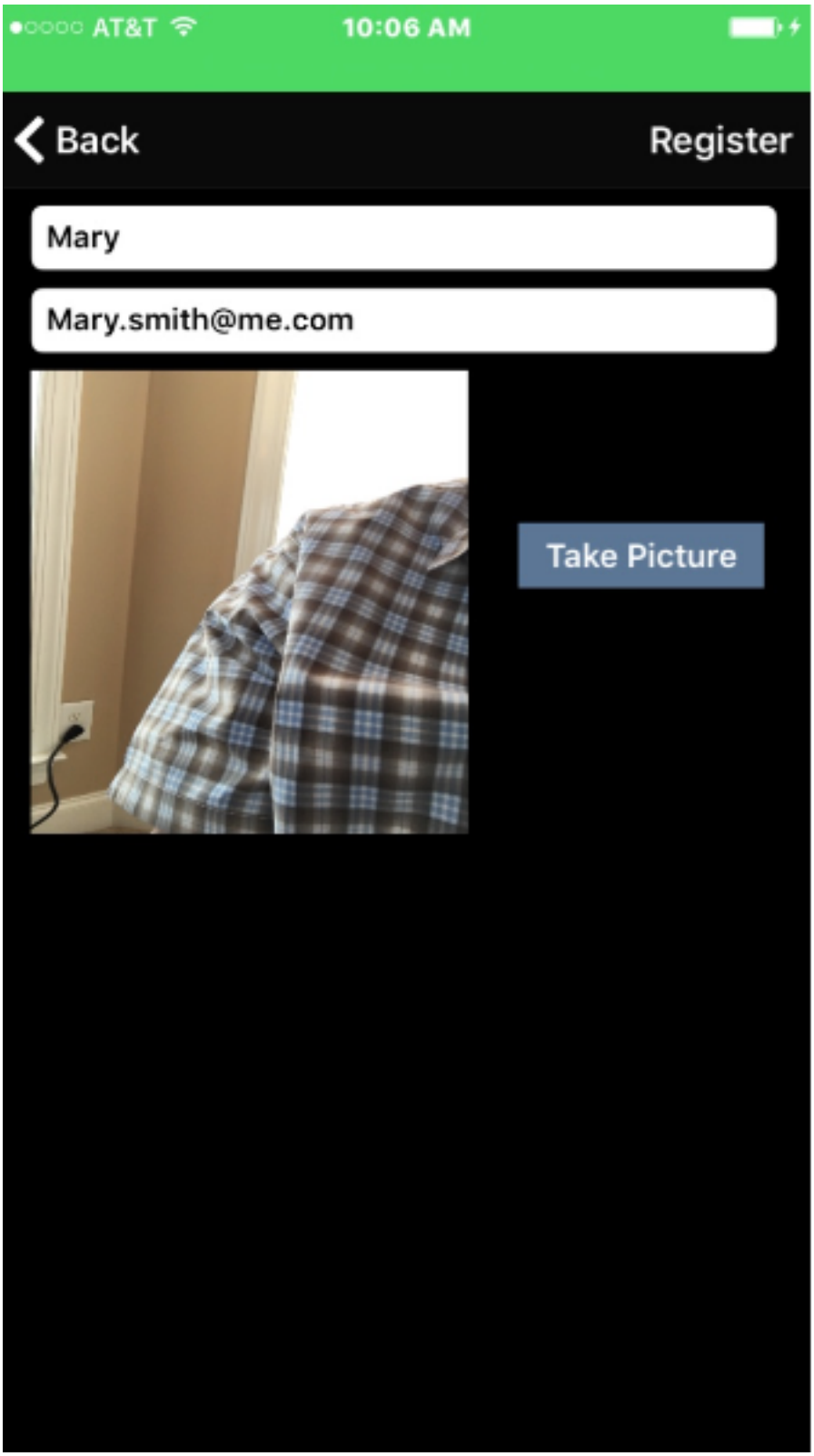


Client Login


Back to Good
9:59 AM
Back
Chaperone Instructions
Existing Location
500
600
800
900
Destination Location
700
300
400
500
600
Request Chaperone Now



Register Client


AT&T
10:06 AM
Back
Register
Mary
Mary.smith@me.com
Take Picture

Chappy App Co. Financial Reports

Assumptions

Chappy App Co. Assumptions Year 1			
Assumptions:			
	Sales Price	$1/Trip	
		Year 1	Year 2
	Client Adoption	10%	15%
	Chaperone Adoption	10%	15%
	Students On Campus	6000	
	Client Usage Per Month	10	
	Average Trip Time	15	
	Number of Monthly Trips per Chap	50	
	Chaperone Trip Revenue	80%	
	Chappy App Employee Wages	$10/hr	
	Initial Adoption by Clients	600	
	Average Monthly Spending by Client	$10	
	Initial Adoption by Chaperones	600	

Start-Up Costs

Chappy App Co. Start-Up Costs Year 1		
Costs:	**Reoccurring**	**One-Time**
Equipment/Hardware		$4,500
Licensing Cost		$850
Insurance	$2000/yr	
Business Filing Fees	$100/yr	
Market Research		$1,200
Lawyer Fees		$3,000
Office Supplies		$500
Advertising & Marketing		$2,500
Office Furniture		$200
Rent Deposit		$500
Total Start-Up		$13,250.00

Balance Sheet

Chappy App Co. Balance Sheet Year 1				
		Assets		
Current Assets:				
Cash			$10,000	
Accounts Receivable		$25,661		
Less:	Reserve for Bad Debts	0	25,661	
Prepaid Insurance			2,000	
	Total Current Assets			$37,661
Fixed Assets:				
Furniture and Fixtures		200		
Less:	Accumulated Depreciation	50	150	
Hardware		4,500		
Less:	Accumulated Depreciation	3,000	1,500	
Office Supplies		500		
Less:	Accumulated Depreciation	375	125	
	Total Fixed Assets			1,775
Other Assets:				
Goodwill			0	
	Total Other Assets			0
Total Assets				$39,436
		Liabilities and Capital		
Current Liabilities:				
Accounts Payable			$2,500	
Sales Taxes Payable			0	
Accrued Wages Payable			800	
Unearned Revenues			0	
	Total Current Liabilities			$3,300
Total Liabilities				3,300
	Total Current Liabilities			$3,300
Total Liabilities				3,300
Capital:				
Owner's Equity			0	
Net Profit			36,136	
Total Capital				36,136
Total Liabilities and Capital				$39,436

Income Statement

Chappy App Co. Income Statement Year 1			
Revenue:			
	Gross Sales		$25,661.14
	Less: Sales Returns and Allowances		$0.00
	Net Sales		$25,661.14
Cost of Goods Sold:			
	Add: Direct Labor/Wages	$20,800.00	
		$20,800.00	
	Cost of Goods Sold		$20,800.00
	Gross Profit (Loss)		$4,861.14
Expenses:			
	Advertising	$2,500.00	
	Charitable Contributions	$100.00	
	Company Credit Card Fees	$50.00	
	Equipment Depreciation	$1,000.00	
	Insurance	$2,000.00	
	Miscellaneous	$100.00	
	Hosting Services	$150.00	
	Office Expenses	$200.00	
	Permits and Licenses	$850.00	
	Professional Fees	$100.00	
	Rent	$600.00	
	Telephone	$20.00	
	Travel	$0.00	
	Utilities	$650.00	
	Total Expenses		$8,320.00
	Net Operating Income		-$3,458.86
Other Income:			
	Trial Contract with Uber	$10,000.00	
	Total Other Income		$10,000.00
Net Income			$6,541.14

Forecasting

Chappy App Co.
Sales Forecasting

****Sales Forecasting based on a Single Client****

Year 1

	January	February	March	April	May	June	July	August	September	October	November	December	Total
Total Client Trips Per Month:	10	11	12	13	15	16	18	19	21	24	26	29	214
Total Chaperone Earnings Per Month:	$8.00	$8.80	$9.68	$10.65	$11.71	$12.88	$14.17	$15.59	$17.15	$18.86	$20.75	$22.82	$171.07

Year 2

	January	February	March	April	May	June	July	August	September	October	November	December	Total
Total Client Trips Per Month:	29	33	38	43	50	57	66	76	87	100	115	133	827
Total Chaperone Earnings Per Month:	$22.82	$26.25	$30.19	$34.71	$39.92	$45.91	$52.80	$60.71	$69.82	$80.30	$92.34	$106.19	$661.96

****Sales Forecasting based on Initial Adoption****

Year 1

Average Number of Trips Per Client Annually:	214
Average Amount of Annual Earnings Per Chaperone:	$171.07
Average Number of Trips for all initally adopted Clients:	128306
Average Amount of Annual Earnings for all Chaperones:	$102,644.56

Year 2

Average Number of Trips Per Client Annually:	827
Average Amount of Annual Earnings Per Chaperone:	$661.96
Average Number of Trips for all initally adopted Clients:	496471
Average Amount of Annual Earnings for all Chaperones:	$397,176.72