# 

# Table of Contents

[**Table of Contents**](#_qfiddj4ubvu0) **1**

[**Proposal**](#_d2oef51pixot) **2**

[**Goals of the Project**](#_bh1i6tum7lue) **3**

[**ER Diagram**](#_1zc7fiqzzeid) **4**

[**Summary**](#_ar9hqyfsjpba) **5**

[**Future Improvements**](#_cna31kucfpp4) **7**

[**Software Install**](#_pd7x116fopuv) **8**

[**Environment Setup**](#_dxjh72v5zqol) **8**

[**Github Integration**](#_thkm5ihp8jw6) **11**

# 

# 

# Proposal

The intent of the SMS Food service application is to assist with the collection and distribution of food products from businesses to distributors and consumers. The app will allow for the reduction of waste from businesses by utilizing excess or near expiration food in order to feed low-income families at little to no cost. The application will be designed to allow businesses to notify, via text message, distributors and consumers of available excess food in detail and allow for them to reserve/buy the food. The customers will then be able to pick up the reserved or purchased food within a reasonable time. This application can personalized for Sharefest in the future with additional features such as region specific notification, more detailed listing of the available product, and greater administrative control. Upon successful development and implementation of this application, Sharefest will be able to assist low-income families with alleviating financial stress related to food, provide proper nourishment to those same families, and assist businesses with decreasing food waste.

# 

# 

# Goals of the Project

•Create a working application which allows pantries to submit food items that can be views and picked up by consumers and distributors.

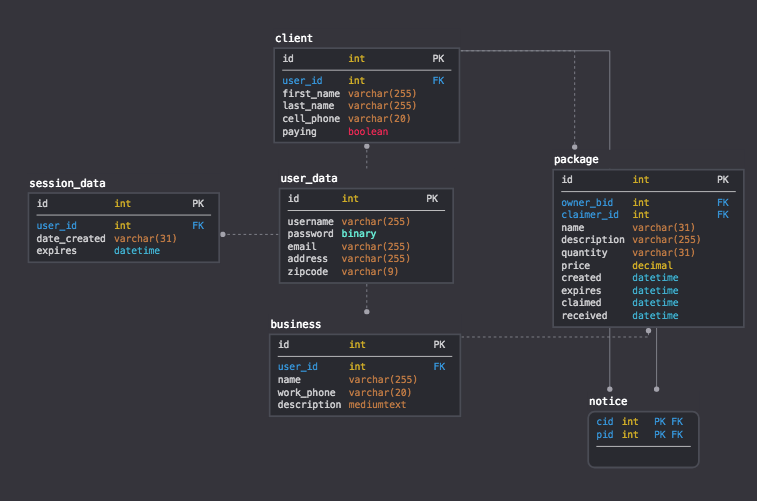
•Create a web-application to keep records of transactions made by all parties.

•Fix problems that could potentially arise when application is in production/live.

# 

# 

# ER Diagram



# 

# 

# Summary

Stefan: Initially the android application did not work with the API and the database server. There were a number of code changes that needed to be made in order to integrate the three components and ensure that they are working together. After the code has been changed to improve connectivity, the android application needed improvement to make it user friendly. There are three parts to the android application which were improved the business, the client and registration side. The business side of the application had parts which needed improvement which are the food posting, posted food items, and account information. All of the parts were improved by taking out the unnecessary aspects which cluttered the android activities(pages). The client side was improved by adding more information when the client chooses to view packages that are submitted by the business clients. The registration part of the android application was improved by removing a number of labels which cluttered the layout of the android application. Background, graphics and fonts were improved to give the android application a more user-friendly feel. There are still a number of improvements that need to be made in order for the application to fulfill its potential.

Jay: Towards the beginning of this project we decided our roles and mine was the web server side. I began learning Django however after finishing sprint 1 we were notified at week 9 about a previous repository with the same project in mind. With only 5-6 weeks left to work on this project not much could have been done. I studied on how a C#ASP.NET server works to be able to contribute to this project. Most of the things I have done has to do with verification in sending data to the database. Although that needs some more work. I fixed some of the previous issues the project had. Made simply worded documents to help future contributors get started on it. Meet with professor Lewis twice in this semester to review our goals and needs of the project. He wanted mostly a simple web site and app that would be able to send data to the database, we accomplished that goal.

Zach: During the beginning of the project, I had decided to take on the role of the database. The main goal for this was to turn the current database into a MySQL database. To do this a began to learn more MySQL then I previously did. This took a couple of week to hoan before I actually got into the project. We ended up adding more variables into the database after further conversations with the team. During the last couple of weeks, the team and I really focused on fine-tuning the app. We made the appearance look better by editing the XML files in the android studios platform. The final things to do with the app are to continue to fine tune it and start writing documentation. The guys ran me through the conversation with Prof. Lewis and Gary. The app seems very important and we will do our best to get the app where it needs to be. The app in terms of being done is probably about 85% done.

Forrest: During the division of roles it was decided that I would work on the website application. During our discussions about the direction we wanted to take the project, it was decided that we would focus on the android application, however, we wanted to make a very basic website to demonstrate future functionality. I decided to focus on account creation and signup, and succeeded in establishing a page for both clients and businesses to sign-up. My initial focus during the semester was on sprints one and two, which means that for weeks 1-5 I spent most of my time researching different tools with which to write the website code, and for weeks 6-11 I spent my time designing the website layout and helping my group members with anything that was needed. When we received access to the previous semester’s repository I learned that they had not used any of the tools that I had researched. As a result of my inexperience with web design and lack of time, I kept the website very basic, assisted my group members with what they needed, and planned possible future improvements.

# Future Improvements

|  |  |
| --- | --- |
| **Platform:** | **Improvement:** |
| Android Application & Database | Add a third user for the middle man between the users and providers |
| Android Application | Develop code to separate users and partners into communities |
| Android Application | Develop code where the provider will be able to choose if they want to alert everyone or just their community on submitted packages |
| Web Application | Implement code to where the user and providers will be able to view and post food packages |
| Web Application and Android Application | Develop code where a provider will be able to post a number of items under one package |
| Web Application and Android Application | Develop code where the users will be able to select certain items and quantities from a package with a number of items |
| Android Application | Send mass notifications to users who are able to pick up packages |
| Android Application | Develop news feed where people can post about events that food drives |

# Software Install

Android Studio, C#ASP.NET Visual Studio, MySQL community server, postman, MySQL Workbench (recommended)

**Android Studio** – download this from developer.android.com. Installation is straight forward. No need to download anything else for this

**Visual Studio** – download and install Visual studio community 2017 from visualstudio.com. While installing make sure you check the box for C#ASP.NET during the installation.

If you already have Visual Studio installed launch Visual Studio go to ToolsàTools and Features and then install ASP.NET & Web Development

**MySQL Community** **Server** - For a lightweight installation, download MySQL Community Server from https://dev.mysql.com/downloads/mysql/ and extract the compressed folder to the location you want the MySQL server files to be located. I would recommend downloading to the C:\ drive and name is MySQL Directory. (This will refer to the file location as [MySQL Directory]

**Postman** - Postman can send arbitrary HTTP requests and receive HTTP responses, which is useful for testing the Web API without building a user interface. Download the installer from https://www.getpostman.com/apps and run it.

**MySQL Workbench (recommended)** – download and install from https://dev.mysql.com/downloads/workbench/. This will be used to be able to view the SQL tables in a much easier way.

# Environment Setup

**Android Studio:**

Once the project is created, select Run and then Run ‘app’ or Debug ‘app’ from the Navigation Bar. This will take you to a menu to select your deployment target. If you have an Android device plugged in via USB, you can choose this. Otherwise you can configure a virtual device. If you have not configured a virtual device before, you are prompted to select a model, RAM to use and more (the project was primarily tested emulating a Galaxy Nexus). When you select "OK" an emulator will appear and open the app. Add and edit files through the Android Studio IDE, which is straightforward.

The Android Studio project on the repository may contain user-dependent and/or binary files which are not intended to be tracked, such as in the .idea and .gradle directories. Avoiding adding these files in the first place is a good idea, but if they are pulled and Android Studio starts acting up (e.g. saying the Android SDK version must be selected before running the app), deleting .idea and .gradle, opening Android Studio again, and pressing the “Sync Project with Gradle Files” button helps.

**Visual Studio:**

Visual Studio is used to develop the Web API (FoodServiceAPI) and Web interface (CapProj) projects.

The Web API project uses the ASP.NET Core 2.0 framework, which is open-source and cross-platform. The Web interface project uses the ASP.NET Framework, distinct from ASP.NET Core 2.0.

Open the solutions pulled from the repository. For running your project, select **IIS Express** from the navigation bar or type Ctrl+F5 (for running without debugging) or F5 on your keyboard. If you are using a **Mac, IIS Express** will not be available so just run it with whatever is there.

Running the project from Visual Studio launches the IIS Express server on localhost. The Web interface project opens the default browser to the current page being edited. The API project opens the default browser and automatically sends a request to a route specified in the project properties, currently configured to the default **api/values** route. **That route does not exist, so the response should be a 404.**

The Web API is set to listen for external requests on **127.0.0.1:50577**.

Note: The **Web API project must be opened and run with administrative rights** in order to avoid an error. It needs admin rights because it is configured to have IIS Express accept requests to ‘127.0.0.1’ instead of the default ‘localhost’, which IIS Express treats as two distinct host names during routing. ‘127.0.0.1’ is used because a ‘10.0.2.2’ request sent from the Android emulator is translated to ‘127.0.0.1’, not ‘localhost’. An Invalid Hostname error (along with a status code 400) would occur otherwise. Oddly enough, sending requests to ‘localhost’ directly (i.e. from Postman or the Web server) works OK.

**MySQL Server Initial Setup:**

**[MySQL Directory] : This place where you installed**

Open cmd.exe (may need admin rights for this initial command) and cd to [MySQL Directory]/bin. (**This is where you saved the MySQL download**) Execute the command:

mysqld --initialize-insecure

This sets up the MySQL server, creating the root@localhost user with no password (which is okay for local development), storing information in [MySQL Directory]/data, and creating a Windows Eventlog registry key. The command may take a while. It never has to be run again unless your reinstall MySQL Community Server.

**Starting the MySQL Server**

To run the MySQL server, cd to [MySQL Directory]/bin and execute mysqld (admin rights no longer needed). The console will start mysqld.exe in the background and block while it is running, though you can exit cmd.exe without shutting down mysqld.exe.

To log into the running MySQL server as root@localhost, open up another cmd instance in the bin directory and execute:

mysql -u root -p

If root has no password, either leave out -p or enter a blank password when prompted.

Once logged in, the cmd prompt should be "mysql>". Here you can enter SQL queries, such as SHOW DATABASES; to show all the databases on the server. **To make it easier you can use SQL Workbench to view database tables**. The semicolon is essential, otherwise MySQL will prompt for additional lines until a semicolon is entered. If, even with a semicolon, MySQL is still prompting for new lines, stopping and restarting both mysqld.exe and mysql.exe seems to fix the issue.

Enter SHUTDOWN; to stop the MySQL server. Enter EXIT; to log out.

If you want to set the password for the user you are logged in as, execute:

SET PASSWORD = ‘password’;

**MySQL Database Setup:**

To run the MySQL script that creates a database on the MySQL server, do the following:

● Start the MySQL server (launch mysqld.exe from cmd while in the [MySQL Directory/bin])

IF YOU HAVE MySQL WORKBENCH SKIP THE NEXT STEPS AND GO TO **MYSQL WORKBENCH SETUP**

● Log into the MySQL server as a user with admin privileges, such as root (mysql -u root -p)

● Execute SOURCE [Path to Script] (This is the location of the SQL Script from the project folder, in the database folder)

[Path to Script] is relative to the working directory you were in when you logged into the MySQL server.

**MySQL Workbench Setup:**

In order to run the database, you must use the cmd to start it. (see the first step in the section SQL Database setup). After starting the database server open MySQL Workbench. In the opening screen next to MySQL Connections, click on the + button. Name the connection whatever you want. Leave everything else default. Once you have done this you will see an option under MySQL Connections with your new Connection. Click on that. Once you are connected to the database go to Fileà Run script. The script you will use is the one that you get from github. It is in the database folder (sms\_food\_service.sql). **If you do not see Run Script, that means the server is not currently running.** Now you can see the database tables by look at the Navigator section to the left. Under Schemas you should see sms\_food\_service. Expand it to see more information. Find tables and expand that as well. This will be the list of tables. To view a table, hover over it and click on the rightmost icon within the name. It looks like a table.

**Connect Web API to MySQL Database:**

The Web API project is configured to connect to the MySQL database via the MySql.Data.EntityFrameworkCore package used to implement a database context that can be used with LINQ. The connection string is in appsettings.json.

Before running the project, make sure mysqld.exe is running on the port specified in the connection string (can set temporarily with the -P [port] option). The default mysqld.exe port is 3306.

**Postman:**

After launching Postman, you can configure and save requests. For example, to create a request to the api/user/login route, click New, click Request, set the request’s name and select a request collection, and click the Save button. Set GET to POST and enter “http://127.0.0.1:50576/api/user/login” for the URL. Go to the Body tab, select the ‘raw’ bullet, change Text to JSON (application/json), and enter the parameters in JSON format:

{

“username”: “testbusy”,

“password”: “dontsteal”

}

Click Send to send the request and wait for a response, shown at the bottom of the window once received. In this case, you should get a session token on success. Click Save to save changes to the request.

**Import Postman Collection:**

In the repository is a file (FoodServiceAPI/sms\_food\_service.postman\_collection.json) that defines a collection of configured requests for all the API routes. To import these into Postman, click the Import button next to the New button and choose that file.

Playing with these requests is probably a good way to grasp how communication with the API does and does not work, and they are handy to have for quick tests.

# Github Integration

**Getting Started and Git Workflow options:**

This depends on our development flow. If we are using the Fork and Pull Model, one person will need to create the master repository on Github. Others will fork the master and clone that fork onto their local machine. They will need to link the clone to the upstream repository using : git remote add upstream http://github.com/nameofrepo and check for changes before making their own. In this model you then add changed files using: add filename or add \* to add all. Then commit -m "Some message with a note about changes.". After that, on Github you submit a pull request.

If using the Shared Repository Separate Branches model, we will need to create a repository add all of the contributors and clone a copy to our local machines. We will need to create branches for our specific features using: git checkout -b my-new-feature-branch. Adding and committing are the same as described above. Then execute, git push origin my-new-feature-branch. There will be an option to submit a pull request. If it is approved by the rest of the team, make sure everything is still up to date to avoid merge conflicts and then execute the following three commands: 1. git checkout master 2. git pull origin master 3. git merge --no-ff my-feature-branch. Then delete the branch from local and master repositories.

**Android Studio:**

This is much easier. The process will be the same for creating and sharing or forking a repository as listed above. However all changes can be managed via the GUI VCS menu. You just have to select preferences, version control, and Github to configure your github account.

There are also multiple tutorials and documents online such as: https://www.youtube.com/watch?v=\_d4fFFAJKVA

**Visual Studio:**

The process is similar to Android Studio. Microsoft Docs provides a nice guide on how to do this. All we have to do is follow it: https://docs.microsoft.com/en-us/vsts/git/gitquickstart?tabs=visual-studio

**MySQL:**

For now, the only thing we will commit is a database-creation script, which will also insert example data. Later, a user-creation script may be committed that creates the MySQL user that the ASP.NET server will log in as. Early tests can use root@localhost.