Frappy

# Project Overview

This project aims to build a website for ordering different kinds of Frappuccinos.

This website aims to replace clutter and confusion found among other mainstream competitor websites, by offering a simplistic UI that gets straight to the point to allow an excellent user experience. The website will support default Frappuccino options, as well as the ability to customize the default options further, or allow customers to create their own Frappuccino from a list of ingredients to their liking.

# Team Organization

Scrum Master: Dylan Spencer (May change over the course of the project)

Developers: Spencer Peterson, Dalyn Dalton, Jacob Fitzgerald, Dylan Spencer

# Software Development Process

The development will be broken up into five phases. Each phase will be a little like a Sprint in an Agile method and a little like an iteration in a Spiral process. Specifically, each phase will be like a Sprint, in that work to be done will be organized into small tasks, placed into a “backlog”, and prioritized. Then, using on time-box scheduling, the team will decide which tasks the phase (Sprint) will address. The team will use a Scrum Board to keep track of tasks in the backlog, those that will be part of the current Sprint, those in progress, and those that are done.

Each phase will also be a little like an iteration in a Spiral process, in that each phase will include some risk analysis and that any development activity (requirements capture, analysis, design, implementation, etc.) can be done during any phase. Early phases will focus on understanding (requirements capture and analysis) and subsequent phases will focus on design and implementation. Each phase will include a retrospective.

| **Phase** | **Iteration** |
| --- | --- |
| 1. | Phase 1 - Requirements Capture |
| 2. | Phase 2 - Analysis, Architectural, UI, and DB Design |
| 3 | Phase 3 - Implementation, and Unit Testing |
| 4 | Phase 4 - More Implementation and Testing |

We will use Unified Modeling Language (UML) to document user goals, structural concepts, component interactions, and behaviors.

# Communication policies, procedures, and tools

Discord - Main source for communication. Used for group calls, collaboration, and general idea sharing and questions. Discord will also serve as a place for accountability, where each member will use the platform to report their work on the project and ask for any help with any blockers if they have any.

Standups - Standups are currently slated to be held 3 times a week on Monday, Wednesday, Friday. On days that group meetings are held during class time, standups will take place in person. When lecture is held on those days, current status for each member will be reported on Discord.

Git - Version control system for tracking changes of any project files from documentation to source code. This also provides accountability as each developer's name is associated with any changes that are made to files in the repository.

Github - Website for hosting the git repository. This is used for tracking tickets, allows communication and collaboration among team members, and provides a decent interface for viewing current and past source code changes or documentation changes.

Tools - The project will be using a custom stack consisting of React, Django, and Postgres (RPD), where React provides a front end framework for building web and mobile applications and allows us to fetch page specific javascript allowing for a simple to use SPA. Django provides the REST API, user authentication, and serves static files via a reverse proxy with NGINX and postgres is a robust relational database

# Configuration Management

See the README.md in the Git repository.

# Risk Analysis

* User Authentication
  + Likelihood - Low
  + Severity - High
  + Consequences - If user authentication fails, users will not be able to login to the site to access their account or order drinks.
  + Workarounds - None. Users must be verified in order to login to the system and use the app.
* Database Structure
  + Likelihood - Low
  + Severity - Medium
  + Consequences - Bad database structure can lead to longer load times, loss of data, and large storage space.
  + Workarounds - None. Good database structure is important to provide a good user experience, as well as less issues during development.
* User Experience
  + Likelihood - Low - Medium
  + Severity - Medium
  + Consequences - If users have a lackluster experience, they may choose to use a competitor's app over ours.
  + Workarounds - Provide incentives to continue to use the app. Such as track how many drinks a customer has bought and give them a free drink if they reach a threshold.
* Queue System Issues
  + Likelihood - Low - Medium
  + Severity - Medium - High
  + Consequences - Customer orders could possibly fail to get registered in the queue, or dropped from the queue prematurely which will lead to a negative user experience
  + Workarounds -
* Hosting
  + Likelihood - Low
  + Severity - Medium
  + Consequences - Hosting the app and being able to serve it to customers is an integral part of the project. Without being able to serve the web app, customers will not be able to order drinks.
  + Workarounds - Use a reputable hosting service.