Frappy Time

Project Overview

This project aims to build a fully functional frappuccino shop.

Designed with customer satisfaction in mind, the frappuccino shop will sell a wide variety of fraps along with the ability to customize your order to create a unique drink. A login verification system will be the backbone of dividing customer, cashier, barista, and manager experiences, all of which will experience different views of the frappuccino shopping experience. A fully functional inventory and payroll system will be ingrained within the application. The online app will be capable of running through a browser on a PC, all while being hosted on a local device owned by the developers.

Team Organization

Team Lead: Jason(Subject to change following team agreement)

Designers: Isaac Airmet, Jason Jackson, Zach Jordan, Jacob Thompson

Developers: Isaac Airmet, Jason Jackson, Zach Jordan, Jacob Thompson

QA Testers: Isaac Airmet, Jason Jackson, Zach Jordan, Jacob Thompson

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

Tools:

Discord - Main platform for communicating with team members. Daily messaging and video meetings will occur on Discord.

GitHub - Main platform for project files. All primary app files will reside within our Github backend and frontend repos, and consequently will also be the place where updates/version-control procedures are maintained.

Communication Policies, Procedures:

Questions and/or concerns within the group should first be expressed on Discord to the team. If the team can not address the situation, the TA and/or Professor may be brought into the discussion.

Daily StandUps will occur on Discord, where each team member will write a few lines describing where they are at on assigned tasks and if they need any assistance. Group Video Calls will be scheduled the day before meetings with the Professor along with at the start of each sprint. Other group calls will be created as needed based on current work requirements.

The Scrum Master will be changed during each sprint. The Team Lead can be changed when a group majority votes in favor of another team member.

If a team member does not stay on track for completing his work, they will first be confronted by fellow team members to see if any help is required. If repeated attempts to help the individual complete his work fail, higher authorities such as the TA and Professor will be contacted at the team's discretion.

Risk Analysis

- Database Failure
  - Likelihood - Low
  - Severity - Extremely High
  - Consequences - Improper handling of data leading to lost user data, inventory, and products
  - WorkAround(s) - No workarounds available within the scope of the project. Daily database backups are possible, but beyond the scope of the project. A database failure

would have permanent effects without recoverability of lost data. Careful reviews of database changes are the best preventive measure.

- Login/Verification Failure
  - Likelihood - Low
  - Severity - High
  - Consequences - Users can't log in to their accounts and therefore are blocked from the app, or users accidentally are permitted into accounts of higher access then intended which compromises app stability/security.
  - WorkAround(s) - Careful reviews of login/verification process changes along with comprehensive testing are the best preventative measures available.
- UI Failure
  - Likelihood - High
  - Severity - Med
  - Consequences - A wide range of consequences could be created from UI bugs depending on their specific nature. Failures ranging from minor graphical bugs to the complete inability to navigate the app are possible.
  - WorkAround(s) - No direct workaround available as the UI is the singular method by which individuals interact with the application. Preventative measures including thorough PR reviews/QA testing are crucial.
- Hosting Failure
  - Likelihood - Low
  - Severity - Extremely High
  - Consequences - The complete failure of the app. If the app is not being hosted properly, it can completely crash.
  - WorkAround(s) - Host through a verified remote hosting platform. For the scope of this project, however, this will not be done.
- Credential Encryption Failure
  - Likelihood - Low
  - Severity - High
  - Consequences - Without proper encryption, bad actors would have an easier time breaking through initial securities and stealing customer data.
  - WorkAround(s) - There are no direct workarounds if encryption fails, but preemptive measures include more secure database and github procedures along with training all individuals who have access to project files.

Configuration Management

See the README.md in the Git repository.