University of Southern Mississippi

Gulf Park Campus

EcoCosts Preliminary Design

Development Documentation

Documented by Jackie Diep

CSC 424 G001

Professor Michaela Elliott

10 March 2021

1.0 Scope

- 1.1 Identification

This document is written in application to the version of EcoCosts preliminary to the Design Review taking place on March 11, 2021 between the members of the EcoCosts development team and Professor Michaela Elliott.

As no coding has taken place, this version is referred to as EcoCosts version 0.

- 1.2 System Overview

EcoCosts was an application envisioned and decided upon by our development lead, Christian Schmachtenberger. It is defined as a "Web application used for budgeting" that also has elements of stock management. The intent of EcoCosts is to create an easily accessible but highly personalizable budgeting system that anywhere from the beginner budgeter to the financially savvy investor may find daily use in, hence the decisions to use an easily navigable website and implement stock management aspects. In version 1.0, EcoCosts is intended as a stand-alone web application that is not currently intended to be part of a larger system.

- 1.3 Document Overview

This document will serve as both a brief overview for potential newcomers or outside individuals of the EcoCosts development process and a log of incremental progress and decisions made throughout the development of EcoCosts. The primary focus will be on the requirements of the application and the process of implementation through design.

2.0 Referenced Documents

1. Intending to utilize the Yahoo Finance API for relevant stock short codes.
2. pgAdmin 4 / PostgreSQL usage instructions
3. Golang package listings

3.0 Requirements

- Windows and Linux operating systems
- Use a website based system to promote accessibility
- Widget based home page that separates into more detailed sections
    - Four detailed modules separating from home page

Rqmt ID	Rqmt Statement	Design Section #	Test Section #
1.0	Shall allow transaction input	1.0	N/A
1.1	Shall allow user input and import	1.0	N/A
1.2	Shall allow CSV importation	1.2	N/A
1.3	Shall maintain atomicity between transactions	1.3	N/A

2.0	Shall display graphical representations of various features	2.0	N/A
2.1	Shall display a cash flow analysis chart	2.0	N/A
2.2	Shall display charting of budget diversification	2.0	N/A
2.3	Shall display categorical separation of user expenses in a pie chart format	2.0	N/A
3.0	Shall contain stock management implementations	3.0	N/A
3.1	Shall allow the user to track stocks by entering stock names	3.0	N/A
3.2	Shall allow the user to search for appropriate NASDAQ short codes	3.0	N/A
4.0	Shall allow users to set budgets	4.0	N/A
4.1	Shall allow the users to create categories to separate portions of budget	4.0	N/A
5.0	Shall have intuitive navigation and efficient presentation of information	5.0	N/A
5.1	Shall implement a navigation bar to transition between pages	5.0	N/A
5.2	Shall allow the user to login and register to save information	5.0	N/A
5.3	Shall implement a separate account details page	5.0	N/A
5.4	Shall implement a stock searching tool on both home and dedicated pages	5.0	N/A
5.5	Shall implement a stock watching tool on both home and dedicated pages for tracked stocks	5.0	N/A
5.7	Shall implement a graphical display of budget categories on both home and dedicated pages	5.0	N/A
5.8	Shall present users with suggestions for budgeting strategies on demand	5.0	N/A
5.9	Shall implement a ledger to summarily present all transactional data present relating to the user	5.0	N/A

5.10	Shall implement a stock ticker displayed below navigation and stock features	5.0	N/A
6.0	Shall initialize a database	6.0	N/A
6.1	Shall handle failed connections	6.0	N/A
6.2	Shall create structs for necessary tables	6.0	N/A
6.3	Shall present only necessary information to the user	6.0	N/A

4.0 Design

EcoCosts is designed with Golang on the back-end. This should ensure that no regressions may occur as Golang is backwards compatible. PostgreSQL 13 shall be used for the database, and the version shall be kept static to combat regression. Extensive use of the Golang libraries will be used to implement many of the requirements described before (primarily requirements 1 - 4 and 6).

Requirements section 5 is intended to be mostly .html and .css based.

The primary operating systems intended for the project are Windows and Linux.

*Design statements will be expanded upon as coding beings.

Dsgn ID	Dsgn Statement
1.0	POST requests will be used to allow users to input and import into the database
1.2	The Golang CSV package will be used to convert CSV files into transactional data
1.3	The Golang sync/atomic and database/sql packages will be used to maintain atomicity between transactions
2.0	Front-end html and css will be used along with the go-chart library to implement all charting and graphical representations of information
3.0	The Yahoo Finance API shall be used to fetch json information and allow searching
4.0	Structs in Go along with the various SQL functionalities similar to Design Section 1.0 will be used to create and set budgets on categories
5.0	A .html and .css based front-end will be used in conjunction with back-end programs coded in Go to implement necessary displays, trackers, and navigational functionalities as well as the login and registration implementation.
6.0	A database will be maintained and coded using Go, PostgreSQL, and pgAdmin 4

5.0 Test Plan

- 5.1 Pseudo Data SQL file

A data.sql file will be created that will store SQL commands to create a test database that will be the same on all developers computers. This will ensure that mixed data will not cause issues to go unnoticed or issues to occur during revisions. Of course, more data will be chosen at the final testing stage to ensure that edge cases do not go unnoticed.

- 5.3 Integration

Project designs will be grouped into modules that handle particular tasks such as the “budgeting” module versus the “stocks” module. Each module will be tested individually to narrow down the arrival of bugs and simplify integration.

Each specific task of the modules will be tested individually with a basic command line front-end before being tested a second time with the intended .html and .css based front-end for proper integration. In conjunction with the data.sql file created before, this should ensure proper integration of the final product as anything beyond highly specific regression testing can be largely avoided due to Golang’s built in backwards compatibility and the static version PostgreSQL 13.

Appendix A. Test Results

As no coding has occurred at this time, no test results are available.