University of Southern Mississippi

Gulf Park Campus

EcoCosts Final Design

Development Documentation

Documented by Jackie Diep

CSC 424 G001

Professor Michaela Elliott

19 April 2021

1.0 Scope

- 1.1 Identification

This document is written in application to the final version of EcoCosts to be covered in the Design Review taking place on April 14, 2021 between the members of the EcoCosts development team and Professor Michaela Elliott.

As the first release, this document refers to EcoCosts version 1.0.

- 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. EcoCosts is a stand-alone web application that is not currently intended for larger system integration.

- 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. Utilizes 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

❖ Red is utilized to signify parts of implementation cut from the final version
❖ Blue is used to signify partial implementation
❖ Yellow is used to signify full implementation

Rqmt ID	Rqmt Statement	Design Section #	Test Section #
1.0	Shall allow transaction input	1.0	Back-end: Pass
1.1	Shall allow user input and import	1.0	Back-end: Pass
1.2	Shall allow CSV importation	1.2	N/A

1.3	Shall maintain atomicity between transactions	1.3	Back-end: Pass
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	Back-end: Pass
3.1	Shall allow the user to track stocks by entering stock names	3.0	Back-end: Pass
3.2	Shall allow the user to search for appropriate NASDAQ short codes	3.0	Back-end: Pass
4.0	Shall allow users to set budgets	4.0	Back-end: Pass
4.1	Shall allow the users to create categories to separate portions of budget	4.0	Back-end: Pass
5.0	Shall have intuitive navigation and efficient presentation of information	5.0	Front-end: Pass
5.1	Shall implement a navigation bar to transition between pages	5.0	Front-end: Pass
5.2	Shall allow the user to login and register to save information	5.0	Front-end: Pass
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	Dedicated page only: - Front-end: Pass Home page: N/A
5.5	Shall implement a stock watching tool on both home and dedicated pages for tracked stocks	5.1	Dedicated page only: - Front-end: Pass Home page: N/A
5.7	Shall implement a graphical display of budget categories on both home and dedicated pages	5.1	Front-end: Pass
5.8	Shall present users with suggestions for budgeting strategies on demand	5.1	N/A

5.9	Shall implement a ledger to summarily present all transactional data present relating to the user	5.1	Front-end: Pass
5.10	Shall implement a stock ticker displayed below navigation and stock features	5.1	N/A
6.0	Shall initialize a database	6.0	Data-base: Pass
6.1	Shall handle failed connections	6.0	Data-base: Pass
6.2	Shall create structs for necessary tables	6.2	Data-base: Pass
6.3	Shall present only necessary information to the user	6.3	Data-base: Pass

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	Fully implemented
1.2	The Golang CSV package will be used to convert CSV files into transactional data	CSV importation cut due to a lack of time and
1.3	The Golang sync/atomic and database/sql packages will be used to maintain atomicity between transactions	Fully implemented
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	All charting and graphical representations of budget were cut due to a lack of time
3.0	The Yahoo Finance API shall be used to	Fully implemented

	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	Fully implemented
5.0	A .html and .css based front-end will be used in conjunction with back-end programs coded in Go to implement navigational functionalities.	Successfully implemented: ● Navigation bar ● Login & Registration Partially implemented: ● Stock searching ○ Only implemented on dedicated page Not implemented: ● Separate account details page
5.1	A .html and .css based front-end will be used in conjunction with back-end programs coded in Go to implement displays and charting.	Successfully implemented: ● Category display ● Transaction ledger Partially implemented: ● Stock watching ○ Only implemented on dedicated page Not implemented: ● Budgeting strategy suggestions ● Stock ticker below navigation bar and stock features
6.0	A database will be maintained and coded using Go, SQL, PostgreSQL, and pgAdmin 4	Fully implemented
6.2	Tables and categories are properly constructed using database.go and schema.sql files as well as a data.sql file for testing purposes.	Fully implemented
6.3	Database is written such that only information relevant to particular users and categories are output to the front-end	Fully implemented

All implementations that were not included or not fully implemented were purely time based decisions. There were no problems that we ran into that strictly rendered things impossible.

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. Test data will be used to verify correct outputs and acceptance of inputs.

- **5.2 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

- **1. Back-end Testing: All pass (Of what was implemented)**

It was decided early on that testing of the back-end would take place in four modules with separate tests. Functionalities would be tested module by module before then trying to combine with the front-end and doing an integration test with the entire program in place.

Modules:

1. Transaction
2. ~~Charting~~
3. Stock
4. Budgeting

Of these modules, everything that we decided to implement passed testing. There was nothing of the back-end that we coded or tested that did not eventually receive a successful implementation into the project.

**We take this as a sign that we did a good estimation of the amount of content that needed to be cut out before release.

The charting module in particular was practically completely cut from the project, as it handled all graphing and visual representations.

A small template of CSV importation was created very early on, but it was never fully coded, implemented, or tested.

- **2. Front-end Testing: All requirements pass, some developer personalizations fail**

The front-end was tested more in a visual method, where each page was scanned throughout implementation to ensure success. The only sections of the front-end that were not entirely successful were slight developer personalizations that were not particular requirements of the project.

For example, a slight statement of “Hello, (Name)” was seeing formatting issues near the top of the application that we never solved. Again, all front-end requirement testing was successful, and no coding was done for things that were not implemented.

Some graphical displays were not implemented due to being cut out from the back-end, and some displays of information were not included on all relevant pages due to time constraints.

- **3. Database Testing: All pass, Fully implemented**

Database testing was performed using the aforementioned data.sql file and test data. No errors arrived from testing, and the database is considered the most implemented part of our project. PgAdmin 4 and PostgreSQL 13 were used to manage and test the database.