EcoCosts Design Review

Christian Schmachtenberger, Hunter Johnson, Jackie Diep, Joshua McGehee, James Rinehart, Durlab Man Sherchan [1]

March 11, 2021

[1] assigned by Dr. Michaela Elliot

Introduction

Financial freedom is difficult. With EcoCosts, our clients will be able to easily plan out their budgets to achieve the ability to free the burden of their finances. The structure of the application puts key financial analysis in the hands of the masses.

Back-end

- Transaction input
- Double-entry bookkeeping
- Cash flow analysis
- Categories
- Stock
- Budget strategies

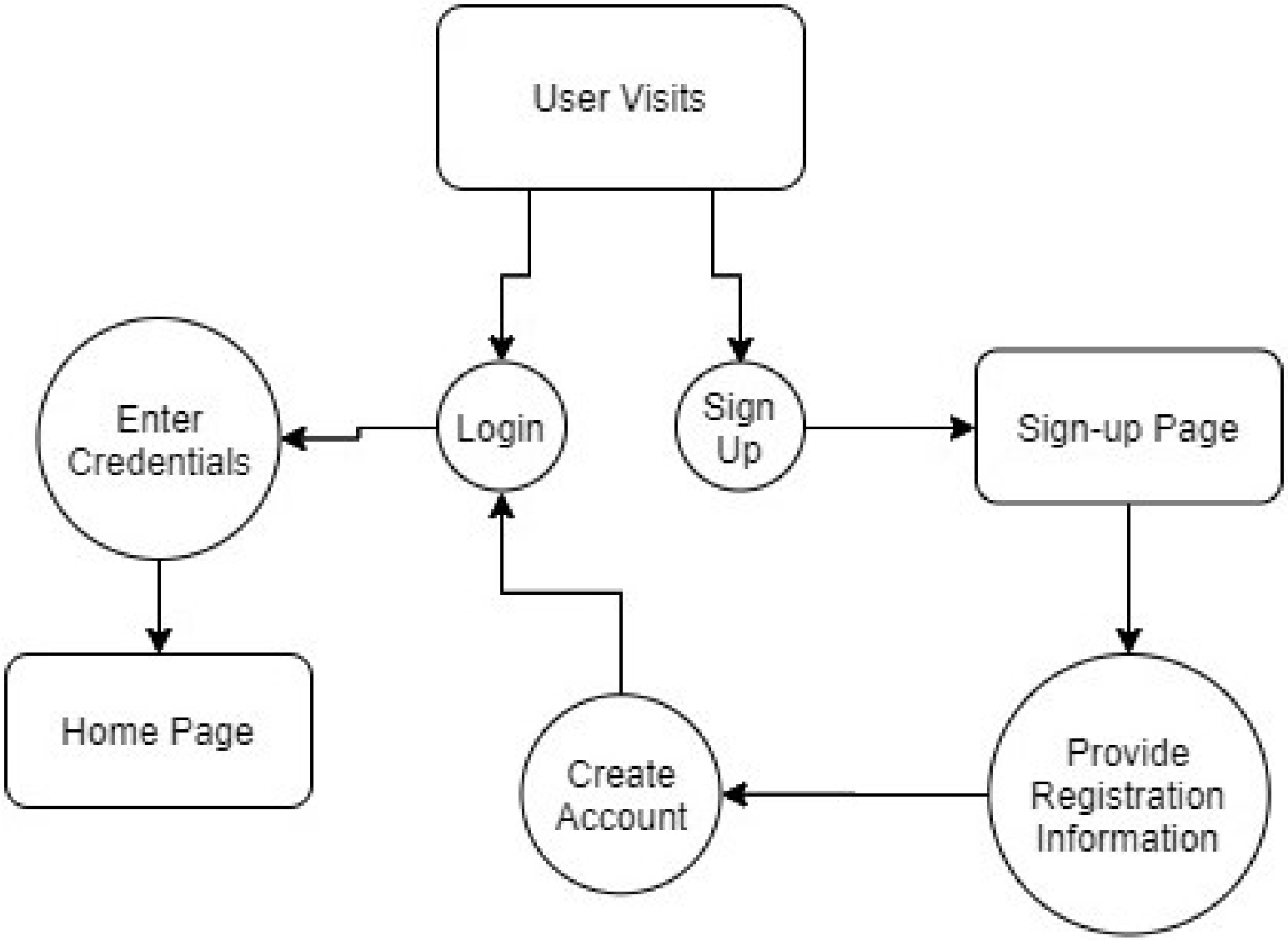
Front-end

- Navigation
- Login & Registration
- Account details
- Stock search
- Stock watcher
- Budget tracker
- Suggestion & tips
- Categories
- Ledger
- Stock ticker

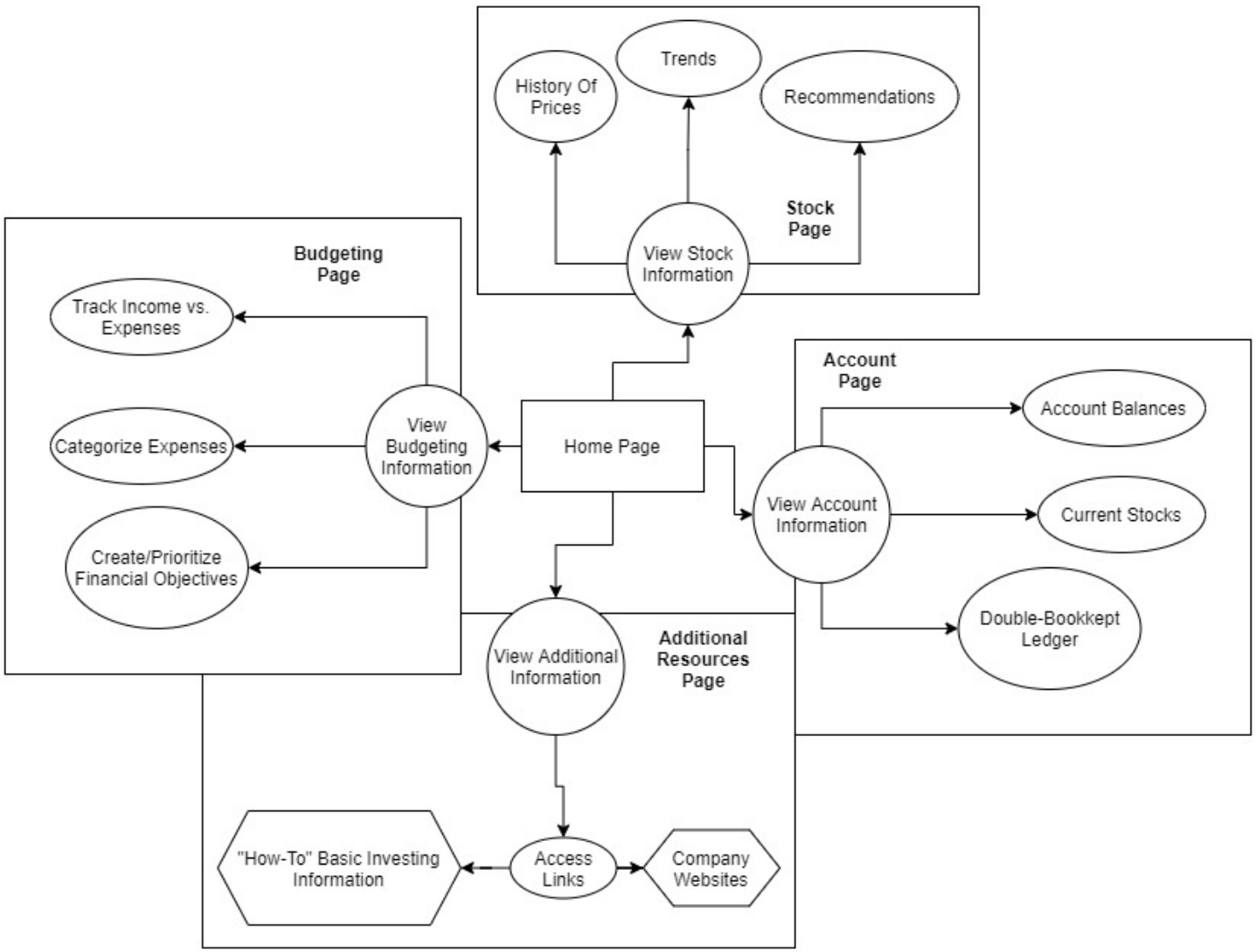
Database

- Relational schema
- Entity-relationship diagram
- Sample SQL queries
- Transaction-based system for atomicity

State Diagram: Landing page



State Diagram: Home page



Fulfill front-end

- *Navigation shall* take user to separate pages via navigation bar.
- *Login & Registration shall* allow the user to register.
- *Account Details shall* be separate from navigation to other pages.
- *Stock Search shall* be present on home page & dedicated page.
- *Stock Watcher shall* be present on home page & display user selected stocks
- *Budget Tracker shall* be present on home page & dedicated page.
- *Suggestions shall* be present on all pages & demonstrate budgeting strategies.
- *Categories shall* be available for sorting different budgets.
- *Ledger shall* show all transactional data.
- *Stock Ticker shall* be below the navigation and display stocks.

Fulfill back-end

- *Transaction module shall*
  - take user input and import into database using `POST` requests.
  - convert a CSV in transaction data using first-party Golang `csv` package.
  - check atomicity between transactions using Golang `sync/atomic` and `database/sql` packages.
- *Charting module shall*
  - create cash flow analysis charts using front-end designs.
  - create pi-charts showing budget diversification using front-end designs.
  - show portfolio diversity.
  - show categorically separation of user expenses.

Fulfill back-end

- *Stock module shall*
  - fetch json from Yahoo API for relevant stock short codes.
  - allow the user to enter stock names and search for the appropriate NASDAQ short code.
  - chart history of stock since user added to the budgeting application and appropriate profit or loss since inception.
- *Budgeting module shall*
  - allow the user to set budgets on certain categories.
  - create categories.
  - call upon charting module to display a category chart.

Fulfill database

- *Database Module shall*
  - initialize the database.
  - handle failed connections.
  - create structs for necessary tables.
- *Database shall*
  - ensure atomicity.
  - use transactions to handle failures.
  - fetch only necessary columns and rows.
  - use proper views.
- *Team shall*
  - plan the database using relational schema.
  - document the relationships between tables.
  - create test data (See Slide 12).

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.

Environment

A virtual machine will be set up to ensure a similar testing environment using a Linux distribution. The Golang version will stay static throughout the project; however, no regressions should occur due to Golang, since the project is backwards compatible on all minor revisions. PostgreSQL will also stay the same version to combat regression.

Integration

The project will feature modules that cover specific tasks as to combat any integration testing that would come from changing modules. Bugs are bound to happen so isolating them to a specific module will help with integration. Furthermore, the preliminary design will cover a modules' specific tasks tied to certain functions and data collection.

Challenges

- Lack of financial knowledge
- Use of Stock API
- Lack of stock knowledge
- Lack of back-end and front-end knowledge within team

Questions