Software Design: Management System (GotoGro-MRM) for Goto Grocery Inc.

Team: MSP_CL4_T1	
Name	**ID**
Enzo Peperkamp	102895415
Nelchael Kenshi Turija	103057559
Julian Codespoti	102997816
Alex Kyriacou	103059830
Marella Morad	103076428

Table of Contents

Design of the Software Components

Model Layer

In our current architecture, Supabase serves as the Model. It stores and manages all data, handles database operations, and enforces data integrity and consistency. Supabase consists of a cloud-hosted relational database service that will store all the database entities required for the system. This then interacts with the view/controller layer to create, read, update and delete records as well as retrieve relevant data from the database for The UI and report generation. This layer is also responsible for all the user access control and authentication within the application including 2FA management.

View/Controller Layer

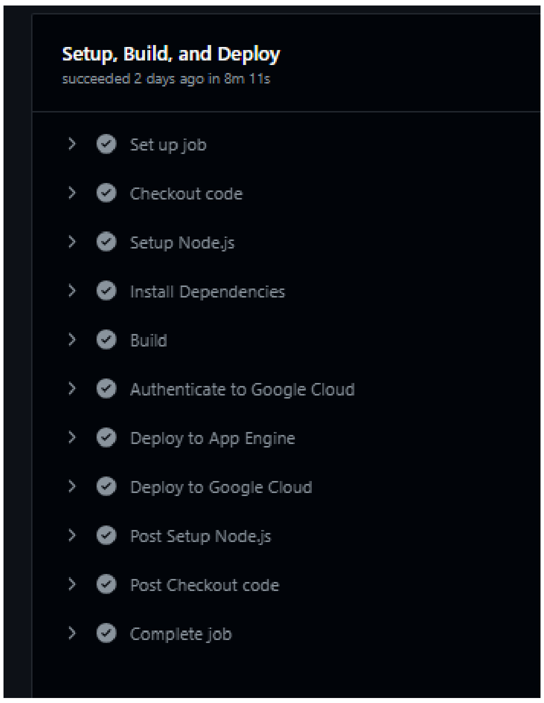
In our current architecture, we have a site powered with React JS hosted on Google Cloud platform that acts as a dual View and Controller layer. In a traditional MVC pattern, the controller and view act as separate entities, with the controller handling input, requests and updates and processing them for the view layer to be presented. This was not required for our project as with modern frameworks such as React JS, the controllers' responsibilities are now able to be handled among various components and libraries. These are able to handle all state management while also handling the presentation of the final site.

CI/CD Pipeline Architecture

In addition to the project architecture, we have also spent considerable thought on the architecture of our CI/CD pipeline. This has made the development of the GotoGro codebase seamless and deployment instant.

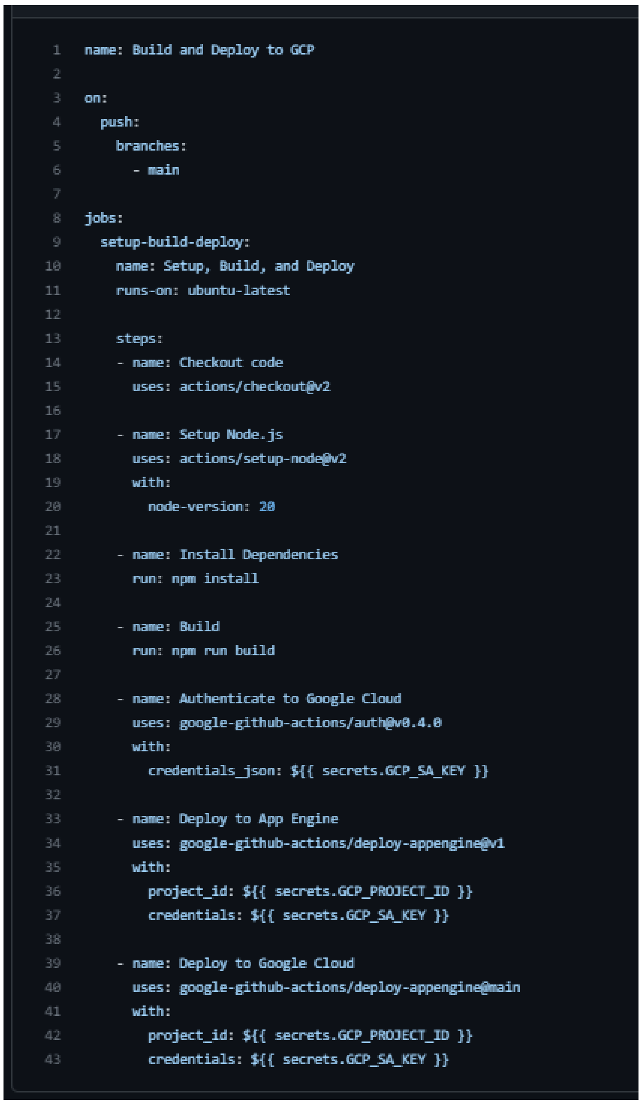
GitHub

We have used GitHub Actions alongside the GCP plugins to set up an automatic deployment pipeline to ensure that our production server always reflects the most up-to-date version of the master branch.



Setup, Build, and Deploy
succeeded 2 days ago in 8m 11s

```
Set up job
Checkout code
Setup Node.js
Install Dependencies
Build
Authenticate to Google Cloud
Deploy to App Engine
Deploy to Google Cloud
Post Setup Node.js
Post Checkout code
Complete job
```



```yaml
name: Build and Deploy to GCP

on:
  push:
    branches:
      - main

jobs:
  setup-build-deploy:
    name: Setup, Build, and Deploy
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Setup Node.js
      uses: actions/setup-node@v2
      with:
        node-version: 20

    - name: Install Dependencies
      run: npm install

    - name: Build
      run: npm run build

    - name: Authenticate to Google Cloud
      uses: google-github-actions/auth@v0.4.0
      with:
        credentials_json: ${{ secrets.GCP_SA_KEY }}

    - name: Deploy to App Engine
      uses: google-github-actions/deploy-appengine@v1
      with:
        project_id: ${{ secrets.GCP_PROJECT_ID }}
        credentials: ${{ secrets.GCP_SA_KEY }}

    - name: Deploy to Google Cloud
      uses: google-github-actions/deploy-appengine@main
      with:
        project_id: ${{ secrets.GCP_PROJECT_ID }}
        credentials: ${{ secrets.GCP_SA_KEY }}
```

Jira

In addition to our GitHub integration we have also set up Jira to assist with our development efforts. Within sprint 2 we will be able to synchronise with GitHub to perform some of the following actions automatically:

- When a pull request is merged, move the relevant issue to done
- When a branch is created, move the issue to in progress
- When a task is moved to in progress, create a branch in GitHub

By having this pipeline in place, it assists the team in both working on the project, as well as knowing with confidence what the state of the project is at any given time. This last point also ensures that our burndown chart stays accurate at all times as it automatically reflects the state of each task within the sprint.

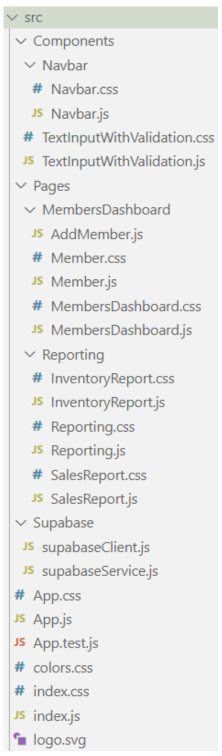
Design Justification using Design Principles

Naming Conventions:

Before starting to develop our web app, we agreed on some naming conventions. Naming conventions are essential for code readability and maintainability. It's important to have consistent and descriptive names for variables, functions, and components to make working on our product a lot easier and not require explanation from each team member every time they code something new.

Component Names:

We are following the camelCase naming convention to name our components, for example, `SalesReport`, `Reporting` and `MembersDashboard`.



```
src
  Components
    Navbar
      Navbar.css
      Navbar.js
    TextInputWithValidation.css
    TextInputWithValidation.js
  Pages
    MembersDashboard
      AddMember.js
      Member.css
      Member.js
      MembersDashboard.css
      MembersDashboard.js
    Reporting
      InventoryReport.css
      InventoryReport.js
      Reporting.css
      Reporting.js
      SalesReport.css
      SalesReport.js
  Supabase
    supabaseClient.js
    supabaseService.js
  App.css
  App.js
  App.test.js
  colors.css
  index.css
  index.js
  logo.svg
```

Variable and Function Names:

We made sure our variable and function names are descriptive and follow the PascalCase naming convention such as `searchMembersByName`, `updateMember`, and `softDeleteMember`. Having such descriptive functions and variable names makes our code self-explanatory and reduces the number of comments we need to add to explain it.

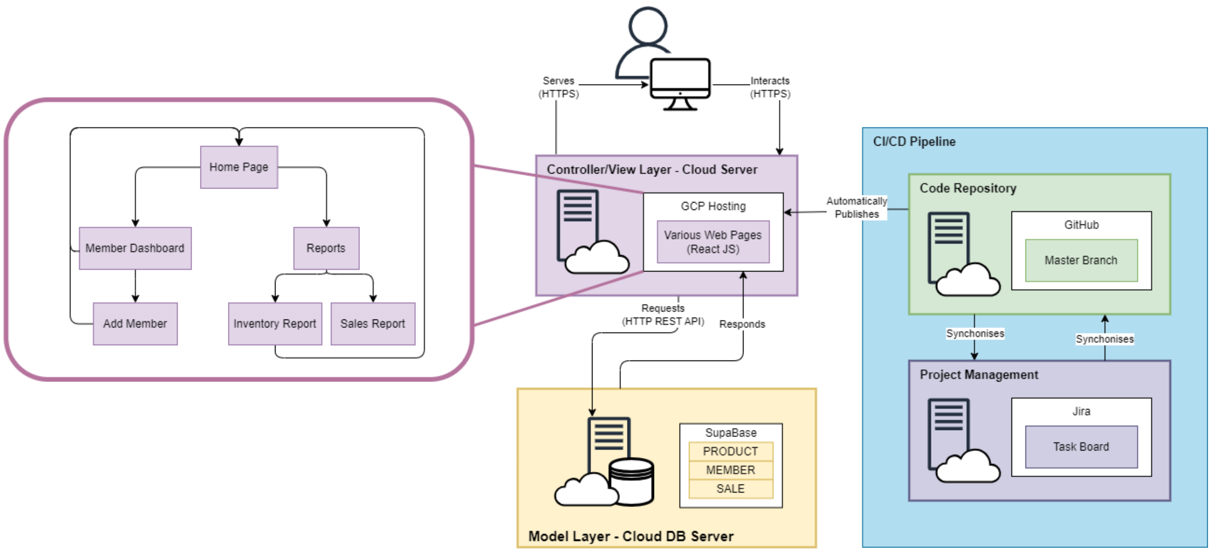
Code Organisation:

We've organized our code files into folders to enhance modularity, readability, and maintenance, facilitating developers' comprehension, modification, and software expansion. Beyond this, code organisation helps foster collaboration, scaling our project, and ensuring its long-term maintainability. This structure keeps our project flexible and efficient as it evolves and grows in complexity.

Design Principles:

Model-View-Controller Design Pattern:

The Model-View-Controller (MVC) design pattern is a software architectural pattern that separates an application into three interconnected components: Model, View, and Controller. As mentioned earlier, in our design, Supabase serves as the Model Layer, responsible for data storage, database operations, and user access control, while React JS, hosted on Google Cloud, serves as the dual-purpose View/Controller Layer. Unlike traditional MVC patterns, where controllers and views are separate, this design leverages modern React components and libraries to manage both user interface presentation and state control. Supabase acts as the backend database service, enforcing data integrity and handling user authentication, including 2FA management, while React takes care of site presentation and user interaction, blending the roles of controllers and views for an efficient and contemporary architectural approach. This is shown in the figure below.



Object-Oriented Principles:

We've structured our code to align with object-oriented principles such that:

- Each React component represents a cohesive unit of functionality
- Each component encapsulates its behaviour and rendering logic
- State variables and props are used to manage data flow and component interactions

Strong Cohesion and Weak Coupling:

We've organized our code to exhibit strong cohesion by structuring it into separate modules, each with a specific responsibility for handling particular functions. This approach enhances the comprehensibility and maintainability of our code. For instance, instead of consolidating all member functionality into a single Member component, we opted to divide it into three distinct components:

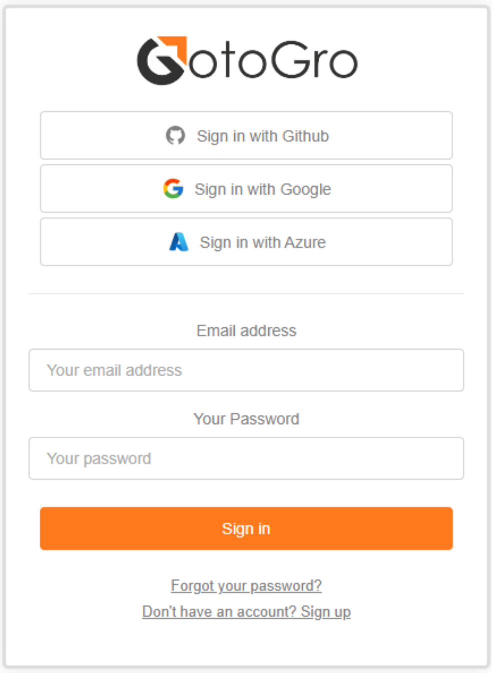
- MembersDashboard: Responsible for searching, retrieving, and displaying members.
- AddMember: Responsible for adding new members.
- Member: Responsible for editing existing members.

Furthermore, we aimed to achieve weak coupling, ensuring that these modules interact with one another with minimal dependencies. In simpler terms, they don't need to have an in-depth understanding of each other's internal workings to collaborate effectively. This concept is clearly exemplified in how our frontend components interact with Supabase APIs without necessitating extensive knowledge of how Supabase functions or how the database is structured. This approach streamlines the process of making modifications or adding new functionality without affecting other parts of the codebase.

Designing the Logo and UI

We made the decision to create a logo and a color palette for our web application. We began by crafting a suitable logo that embodies a sense of warmth and professionalism. After several design iterations, we settled on the logo displayed below. We have developed two variations of the logo to seamlessly integrate with both dark and light backgrounds.

For example:
Light background in our sign-up and sign-in pages:



Sign in with Github

Sign in with Google

Sign in with Azure

Email address

Your email address

Your Password

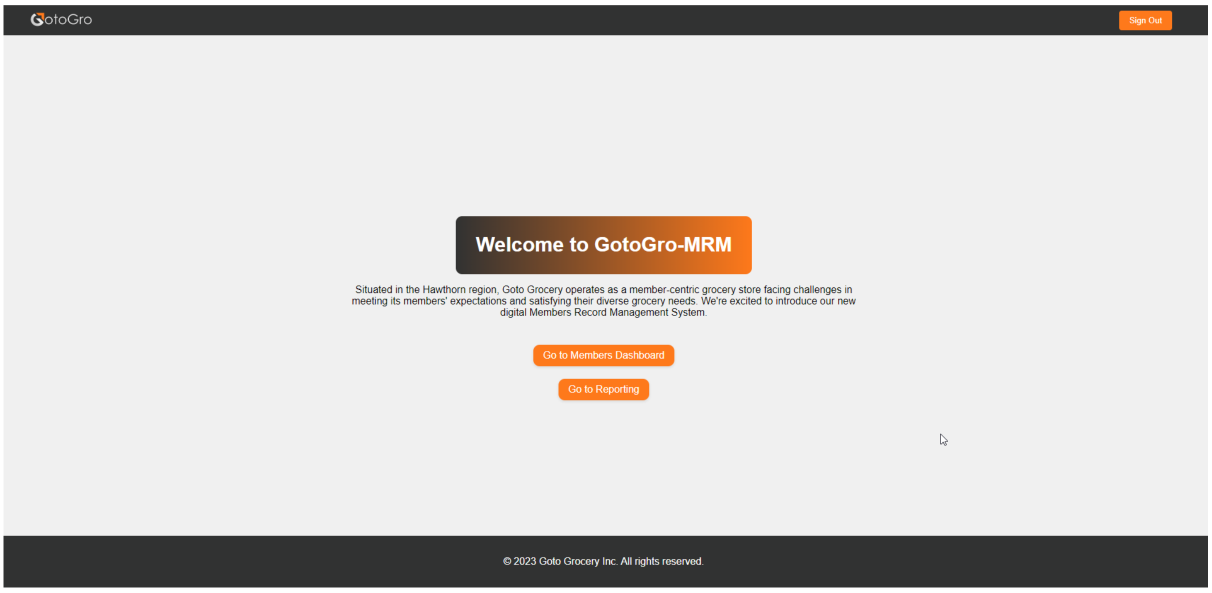
Your password

Sign in

Forgot your password?

Don't have an account? Sign up

Dark background in our navbar:



GotoGro

Sign Out

Welcome to GotoGro-MRM

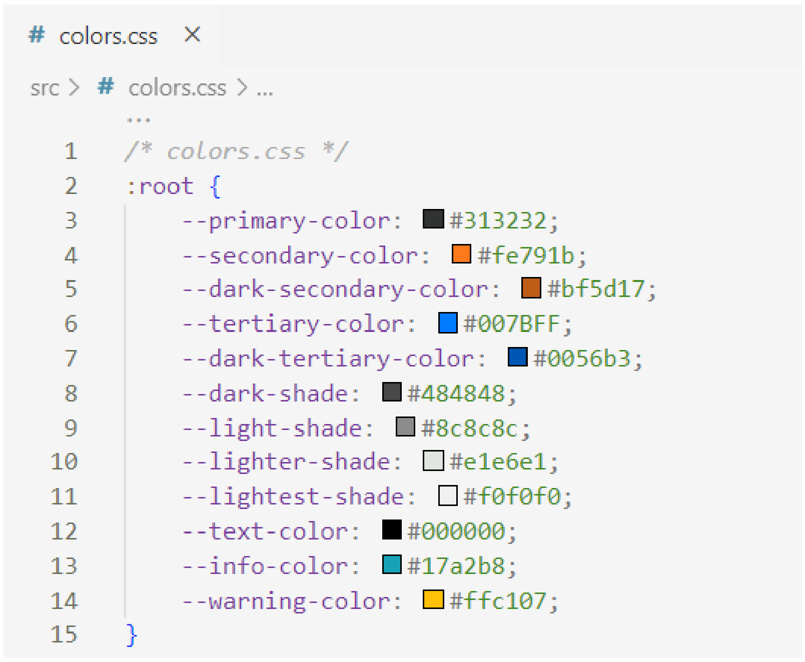
Situated in the Hawthorn region, Goto Grocery operates as a member-centric grocery store facing challenges in meeting its members' expectations and satisfying their diverse grocery needs. We're excited to introduce our new digital Members Record Management System.

Go to Members Dashboard

Go to Reporting

© 2023 Goto Grocery Inc. All rights reserved.

With our logo in place, we proceeded to define our colour palette, ensuring that visual consistency would be a hallmark of our project. We also paid meticulous attention to maintaining a high level of colour contrast to enhance website accessibility.



```css
/* colors.css */
:root {
    --primary-color: #313232;
    --secondary-color: #fe791b;
    --dark-secondary-color: #bf5d17;
    --tertiary-color: #007BFF;
    --dark-tertiary-color: #0056b3;
    --dark-shade: #484848;
    --light-shade: #8c8c8c;
    --lighter-shade: #e1e6e1;
    --lightest-shade: #f0f0f0;
    --text-color: #000000;
    --info-color: #17a2b8;
    --warning-color: #ffc107;
}
```