CASH (Cash And Savings Helper)

Leong Zhong Wei Nicholas, Loo Choon Boon,
Ngan Zisheng Nigel & Tan Hui Min
CS5224 Cloud Computing
AY2020/21 Semester 2
Department of Computer Science
National University of Singapore

Problem Statement

As the saying goes, "It's never too early to plan for retirement". According to a survey conducted during the "circuit breaker", two in three working Singaporeans do not have sufficient savings to last them beyond half a year if they were to lose their jobs (OCBC Bank, 2020). A study showed that 4 in 5 Singaporeans underestimated the amount they require for retirement by 32% (OCBC Bank, 2021) in spite of Singapore's lauded Central Provident Fund system which contributes significantly to retirement planning.

Motivation & Objective

When asked to make crucial life decisions, we are often advised to think what a future version of ourselves 10, 30 or even 50 years down the road would have done.

The CASH web app aims to help Singaporeans visualize what their future versions would look like financially, and match them to partners who provide services to achieve their goals. Coupled with high levels of interactivity and dashboard capabilities, this web app aims to give Singaporeans full control to forecast and eventually realize their financial goals.

Value Proposition

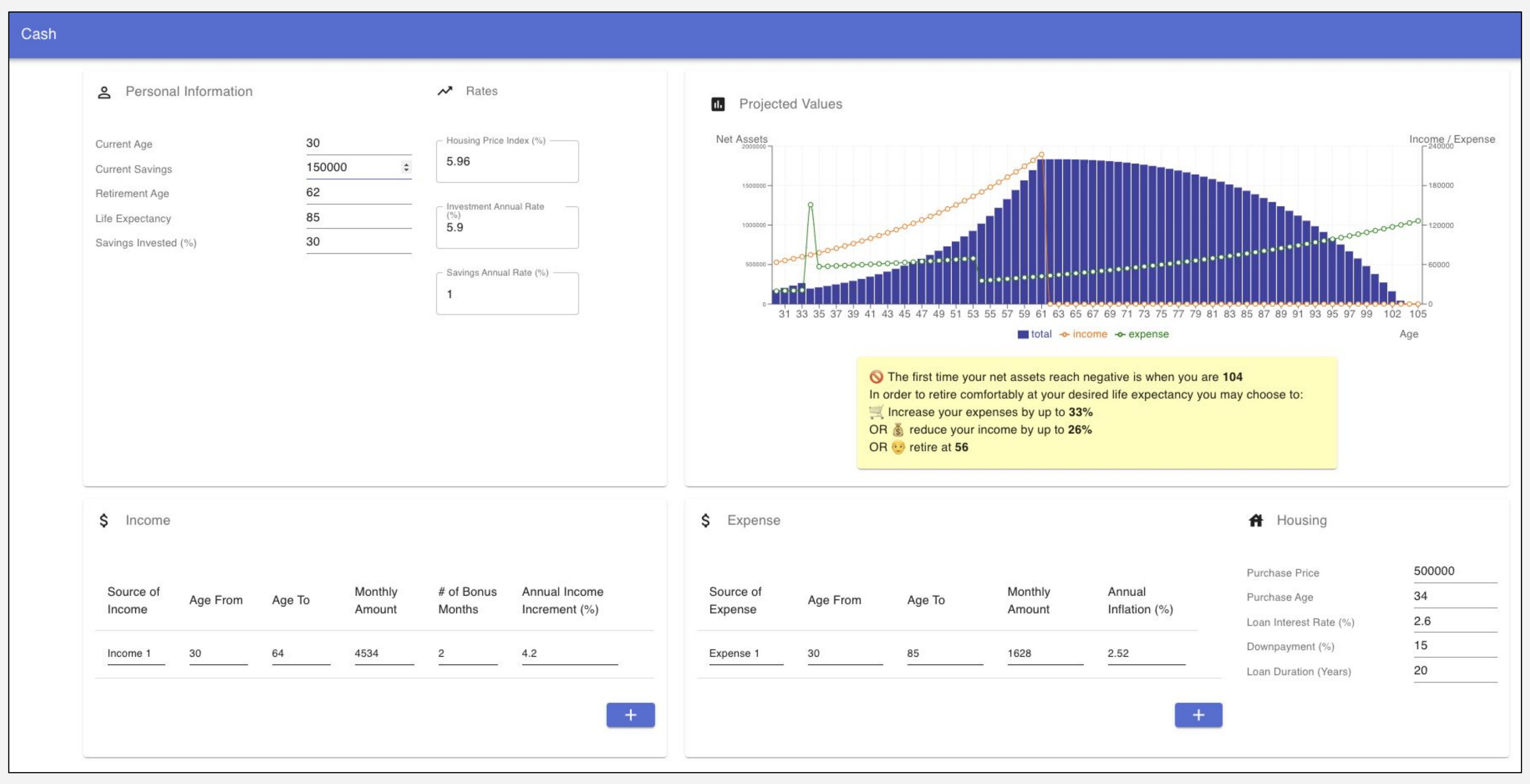
Target audience: Mass market working adults in Singapore	Easy-to-use interface which will help to improve their financial literacy and facilitate them to make better financial decisions
Partners: trading brokerages, interior designers, banks, etc	Match buyers to relevant services/products

Approach

Our approach was to develop a Software-as-a-Service (SaaS) based on three key principles:

1. **Simple**
Without any financial knowledge, users can change the inputs (personal information, rates, income, expense and housing), to visualize different outcomes. All computations are transparent to the user and done in the backend. In the future we plan to add on more inputs and information to guide users.
2. **Personal**
Emphasizing 'personal' in personal finance. Everyone's circumstances are different and our application is built for configurability and customization, to cater for a variety of scenarios such as multiple streams of varying income, taking a gap year and see how it affects your retirement goals?
    a. Do you invest significantly better than others or have lower expenses?
3. **Dynamic**
CASH allows users to set parameters of their financial lives dynamically and visualize the results instantly well into their retirement. This will allow them to identify gaps from their desired financial outcomes, triggering positive behavioural changes to their financial decisions. In the future we plan to include recommendations and online resources to further help this group of people.

Prototype (https://www.cash.cs5224cash.site/)


Cash

Personal Information

Current Age	30
Current Savings	150000
Retirement Age	62
Life Expectancy	85
Savings Invested (%)	30

Rates

Housing Price Index (%)	5.96
Investment Annual Rate (%)	5.9
Savings Annual Rate (%)	1

Projected Values

🚫 The first time your net assets reach negative is when you are **104**
In order to retire comfortably at your desired life expectancy you may choose to:
🛒 Increase your expenses by up to **33%**
OR 💰 reduce your income by up to **26%**
OR 🙂 retire at **56**

Income

Source of Income	Age From	Age To	Monthly Amount	# of Bonus Months	Annual Income Increment (%)
Income 1	30	64	4534	2	4.2

Expense

Source of Expense	Age From	Age To	Monthly Amount	Annual Inflation (%)
Expense 1	30	85	1628	2.52

Housing

Purchase Price	500000
Purchase Age	34
Loan Interest Rate (%)	2.6
Downpayment (%)	15
Loan Duration (Years)	20

Upon first load, CASH displays values based on average Singapore data derived from SingStat, Ministry of Manpower, data.gov.sg, etc. Users can input data in the Personal Information, Income and Expense cards to dynamically visualize outcomes and view personalised recommendations.

Implementation

Front-end (FE) Design
React.js was used to build the the main interface, dashboards and widgets to display financial statistics. React was chosen as it is a mature FE framework with strong community support for building single page applications. Our FE is hosted on AWS Amplify, because of its ability to deploy static sites and single page apps with a Git-based workflow, allowing us to build deployment pipelines, and use tools like authentication and admin dashboards out of the box. AWS Amplify also runs on Docker, meaning we could leverage on containerization to ensure portability of our software.

Back-end (BE) and Data Pipeline Design
The BE application was implemented using Flask, a Python framework for developing web APIs. These APIs power the logic of the application's main functionalities, such as collecting information, performing financial calculations to produce output for the dashboards and insights. The logic for deriving the metrics and insights from a user of our application can be described by a simple formula:

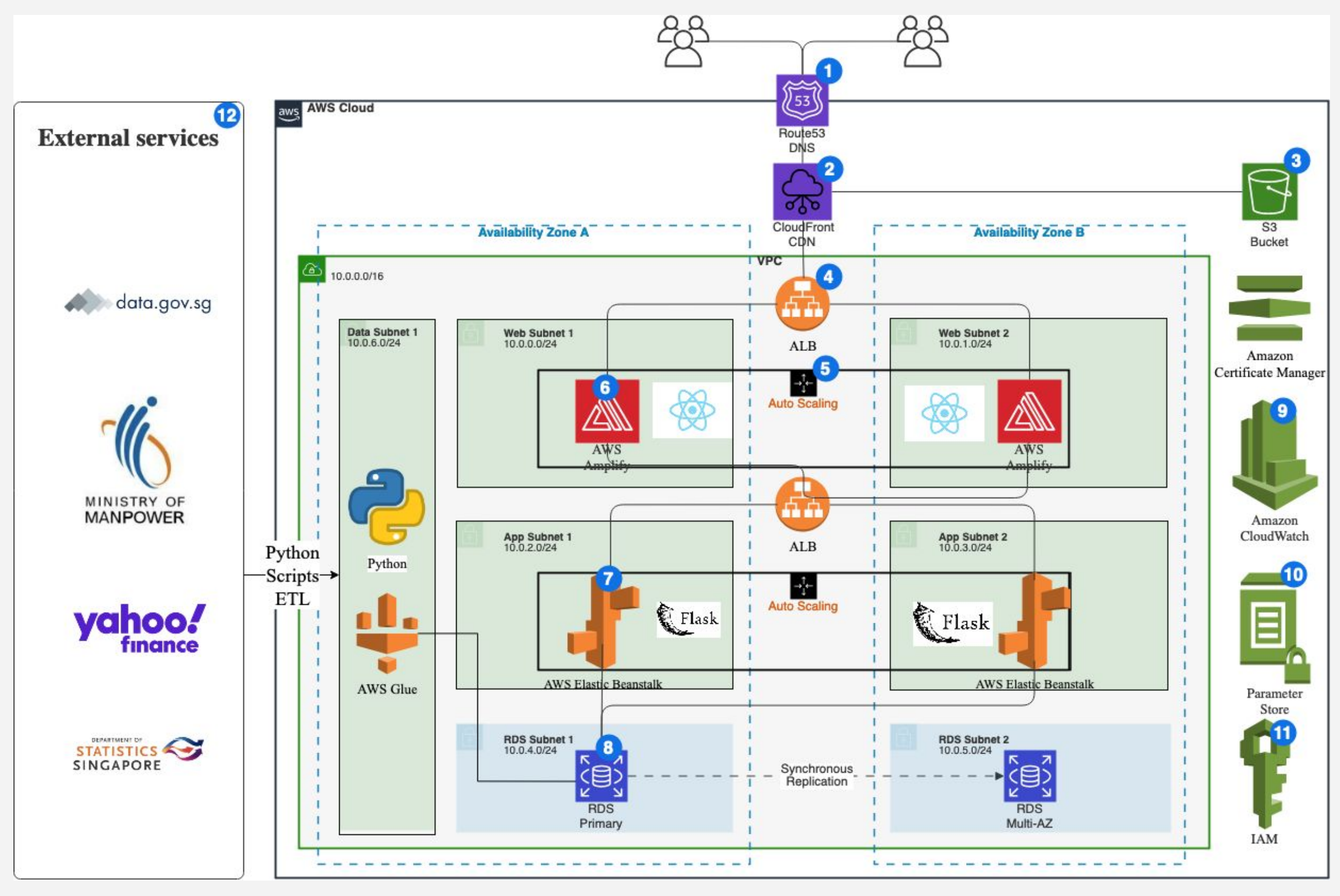
$$Y=f(X, Z)$$

where Y is the set of financial metrics and insights to be generated for the user, X being the set of input values from the user, Z is additional data and insights from 3rd party sources like SingStat, MOM, data.gov.sg and Yahoo Finance, and f represents a function comprising a custom proprietary algorithm we developed. Currently this function is a deterministic algorithm using a series of formulas and financial calculations through the numpy library and we are exploring the use of machine learning to enhance our insights. Our BE provides APIs for the FE to call to export csv reports that are saved in S3, with a pre-signed URL and an expiry date which the user can use to download the reports. The bucket also comes with a life cycle policy to clean up reports after X days to reduce S3 costs of storing the reports.

Privacy is ensured as only a logged in user can see his/her own data, and these user preferences and filters are persisted to a database so that the users would not need to re-enter this information on subsequent visits. The database of choice will be MySQL, a popular open-source RDBMS used in many companies. As the application has a login feature, the database will also be storing sensitive information such as passwords. Proper steps must be made to ensure the security of the application. This includes the use of methods such as salting and bcrypt hashing with a suitable cost factor to ensure the password cannot be easily reverse engineered.

We used Python to build the data pipeline and scripts to extract external data into our database. The ETL scripts can be executed on AWS Glue, a serverless and managed ETL PaaS with in-built autoscaling functionalities and without having to manage the underlying infrastructure.

Application Infrastructure
The BE infrastructure is hosted on AWS Elastic Beanstalk as it is provides a managed solution for parts of the infrastructure such as load balancers and auto-scaling groups, and allows us to leverage containerization to package our application with Docker runtime. The Elastic Beanstalk for the BE REST APIs uses burstable t3.medium EC2 instances. as there might be periods where there may be spikes in CPU demand due to the complex financial calculations. There will be policies in place to scale the application instances up or down in the servers based on CPU usage.



Storage
Our database is hosted on AWS RDS with MySQL due to its automatic backups, multi-AZ support and hot standby for high availability, read replicas and automated upgrades. AWS S3 is used for storing and serving static assets such as images and reports for users. Due to Singapore's data protection requirements, both the RDS which contains user input and S3 buckets which contains user generated content are hosted in Singapore. To minimize network latency between our database and application, the application EC2 instances are located in the same region. To reduce downtime and keep availability high, our AWS application infrastructure is configured across multiple availability zones. To secure the RDS data, we use encryption keys from AWS Key Management Store (KMS) to encrypt data-at-rest.

Network and Security
We deployed an application load balancer (ALB) in front of the Elastic Beanstalk instances, in the public subnet. The Elastic Beanstalk instances are located in the private subnet for security reasons, and we only allow the HTTP ports from the instances to be open to the ALB. AWS CloudFront which is a CDN service will also be sitting in front of the S3 assets to ensure fast content delivery to users and provide other capabilities such as rate limiting. Route 53 is used for DNS resolution to route the user to the actual application and CloudFront CDN when the user accesses our application through the URL.

Monitoring, Maintenance and Development
AWS CloudWatch Rules is used to monitor the uptime for our applications. IAM is used to control access management for the users, while EC2 Instance Profiles is used on the Elastic Beanstalk instances to ensure least privileges on the servers, as well to minimize the impact in the unlikely event our servers are compromised. Our website will be protected by SSL with AWS Certificate Manager attached to the ALB that enables SSL auto renewal, while AWS Parameter Store helps to store our application secrets and keeps them safe with encryption using AWS KMS. Development is done using GitHub as our version control system, and using CI tools like GitHub Actions for continuous integration and continuous deployment.

Revenue Model

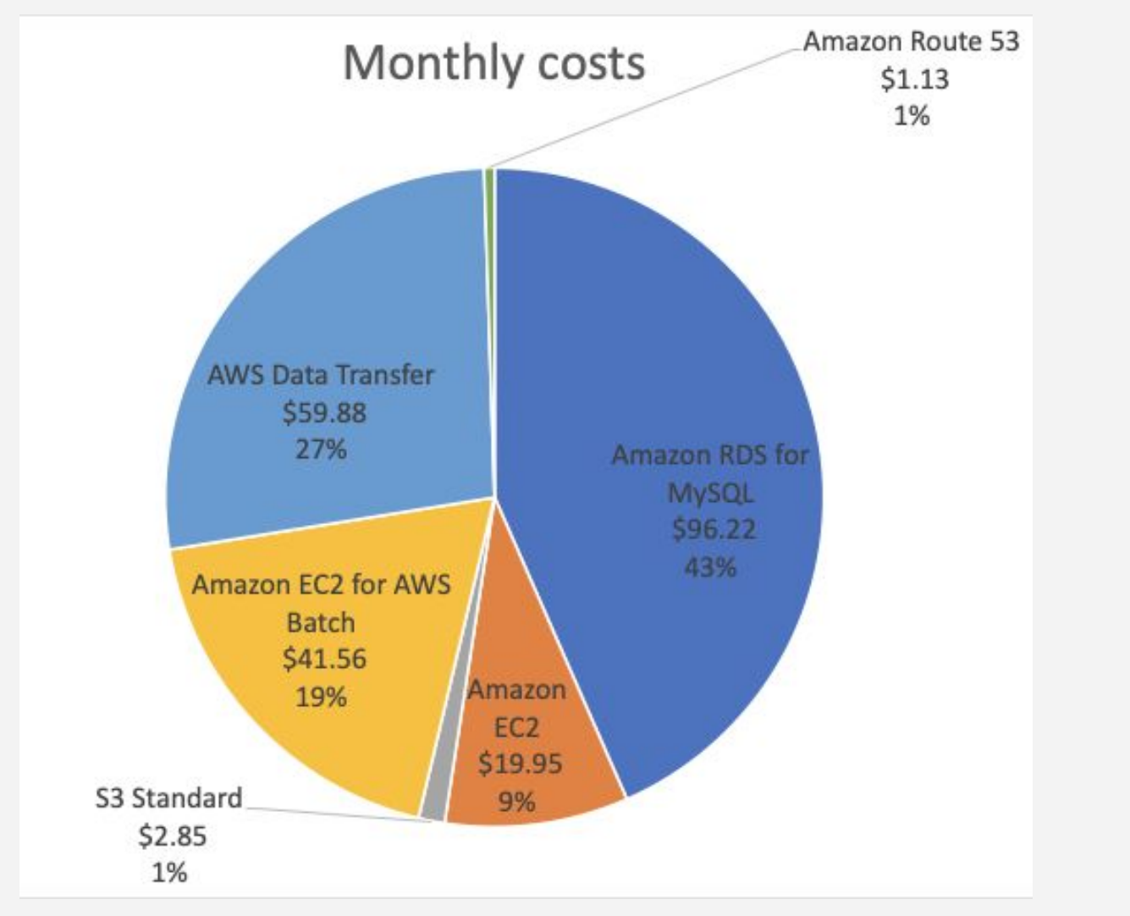
Partner commissions
Our business developments efforts will focus on attracting and onboarding partners onto our platform to sell their services. We will charging a fee to partners for successful sales

Advertisements
As a source of secondary revenue, we will host paid advertisements on our website through Google AdSense to derive additional revenue from cost per click, cost per thousand impressions, etc.

Cloud Costs

Based on our comparison of cost estimates for on-premise and cloud:

- Our total cost would be 44% lower for cloud ($10,447) than on-premise ($18,708)
- Our upfront cost would be 84% lower for cloud ($2,470) even if we only compare it to the cost of servers ($15,000).
- Our monthly fees would amount to only $222 (please see breakdown on the right) for cloud.


Monthly costs
Amazon Route 53 $1.13 1%
Amazon RDS for MySQL $96.22 43%
Amazon EC2 $19.95 9%
S3 Standard $2.85 1%
Amazon EC2 for AWS Batch $41.56 19%
AWS Data Transfer $59.88 27%