National University of Singapore (NUS)

CS3219: Software Engineering Principles and Patterns

Favours4Uni

Project Report

Repository:
https://github.com/CS3219-SE-Principles-and-Patterns/cs3219-project-ay2122-2122-s1-g30

Web Application:
https://favours4uni.web.app/

GROUP 30

A0204683E Nicholas Giancarlo Canete
A0196650X Jordan Yoong Jia En
A0204838B Cheok Su Anne
A0201841U Augustine Kau Zhi Cong

Table of Contents

1. Background

Anecdotally in Singapore, we often note that Singaporeans tend to lack civic-mindedness and a sense of helping others without anything in return. Many public figures, such as veteran diplomat and Professor Tommy Koh have lamented and echoed similar sentiments.

On the other hand, from our experiences as students entering the Singapore workforce as professionals, students and apprentices in a myriad of professional and vocational fields, ranging from web development, to arts and design, to vocational skills like hairstyling, professional repairs and many others. As a beginner or apprentice in these fields, it can be very difficult for students to pick up the confidence to venture into these areas in a more professional setting as beginners often feel daunted by their inexperience and the competitiveness of these fields.

These two observations about Singapore have led us to devise a possible solution to the pertinent social issues delineated above. We look to create a social environment in NUS where we can amalgamate skills training with the notion of giving without return in order to promote the values of kindness and compassion, all while giving Singaporeans the space to hone their craft, network and forge friendships at the same time.

2. Purpose

We aim to create a web application that allows students to request favours, which can be categorised into casual favours and professional favours. Casual favours are mainly everyday tasks, such as getting food, a ride to another part of campus or borrowing an umbrella on a rainy day. On the other hand, professional favours involve tasks that require a certain level of competence or skill, such as web development, video editing, product testing, participation for research projects or help with school-related matters.

By allowing NUS students to voluntarily offer their assistance to these favours, we hope to promote kindness and greater support among the NUS community. Furthermore, this serves as a platform for students to hone their skills by taking on professional favours.

3. Individual Contributions

Name	Technical Contributions	Non-technical Contributions
Augustine	Frontend - Page Implementation (Lead), UI/UX Design, Styling, Integration, Authentication	App ideation, Developer Documentation: Architecture & Patterns (partial), Design Choices and Implementation (partial)
Jordan	Frontend - Page Implementation, UI/UX Design, Integration, Authentication Deployment - CI/CD, GCP Setup	App ideation, Developer Documentation: Architecture & Patterns (partial), Design Choices and Implementation (partial), Reflections
Nicholas	Backend - Favours APIs, API gateway, Users API, Dependencies and Firebase Setup	App ideation, user story-driven ideation, Agile sprint setup, Developer Documentation: Architecture & Patterns (partial), Design Choices and Implementation
Su Anne	Backend - Favours API, Users API, Chats API, Leaderboard Service, User Validation	App ideation, Project requirements, Developer Documentation: Architecture Diagram, Design Choices and Implementation (partial), Improvements and Enhancements

4. Project Requirements

Our team decided to go with option 1 and we developed a web application, *Favours4Uni*.

Favours4Uni is an application that allows students to exchange favours for one another, out of goodwill or as an opportunity to develop various vocational and professional skills for free, in a low-risk and friendly setting.

Do take note of the following naming conventions used:
Logged in users: Users logged into their accounts.
Not logged in users: Users not logged into their accounts or do not have an account.
Users: Includes both logged in and not logged in users.
Expired favours: Favours that have passed the deadline and are yet to be completed.

4.1 Functional Requirements

Requirements prefixed with ‘F’ indicate functional requirements.

4.1.1 User Management

F1.1 The application should allow users to register their credentials through Firebase.
F1.2 The application should allow users to log into their accounts.
F1.3 The application should allow users to log out of their accounts.
F1.4 The application should allow logged in users to view other users’ profiles.
F1.5 The application should allow logged in users to upload their profile photo.
F1.6 The application should allow logged in users to edit their profiles.

4.1.2 Favour Management

F2.1 The application should allow logged in users to create favours.
F2.2 The application should allow logged in users to delete favours that they have created.
F2.33 The application should allow logged in users to set the status of their favours posted as ‘Urgent’ or ‘Not Urgent’.
F2.4 The application should allow users to view all favours sorted by type (Casual/Professional).
F2.5 The application should allow logged in users to request to complete a favour.

F2.6 The application should allow logged in users to accept favour requests.
F2.7 The application should allow logged in users to reject favour requests.
F2.8 The application should allow logged in users to mark completed favours as completed.
F2.9 The application should allow logged in users to view their posted favours.
F2.10 The application should allow logged in users to view their past favours (favours that are expired or completed by others).
F2.11 The application should allow logged in users to view favours they are helping with.
F2.12 The application should allow logged in users to view favours they have completed for others.
F2.13 The application should allow users to view favours sorted by either the time posted or the favour deadline.

4.1.3 User Communications Management

F3.1 The application should allow logged in users to create a new chat with another user.
F3.2 The application should allow logged in users to chat with the user who posted the favour.
F3.3 The application should allow logged in users to view all of their chats.

4.1.4 Statistics Tracking Management

F4.1 The application should allow users to view the leaderboard which shows the top 4 users that completed the most favours (all-time and monthly).

F4.2 The application should allow users to view their own and other users' achievement page which shows the statistics of the user such as:

- Username, email, telegram handle, bio
- List of favours completed

4.2 Non-Functional Requirements

Requirements prefixed with 'NF' indicate functional requirements.

4.2.1 Performance

NF1.1 The application should be supported on Microsoft Internet Explorer, Google Chrome, Mozilla Firefox and Apple's Safari.
NF1.2 The application should be able to run on Windows OS and Mac OS.

4.2.2 Security

NF2.1 The application should only allow authenticated users to create and delete favours, view other users' profiles, request to complete favours and chat with other users.

4.3 Application Features

The requirements listed in 4.1 and 4.2 are exhibited in the features listed below.

1. Users are able to register and login with their email.
2. Logged in users are able post casual favours for other users to view/request.
   Casual favours are favours that tend to issues that arise in our everyday lives. Some examples includes:
   a. Need help to get lunch as not time to get due to classes/not feeling well.
   b. Need someone with an umbrella to go to Utown as it is raining.
   c. I have just misplaced my wallet in the vicinity of Utown. Need help finding it.
   d. I am taking the module CS3219 alone, I need a study buddy to study together.
   e. I am having an exam (MA1521) in 2 hours but my GC ran out of battery. Does anyone staying at PGP have an extra GC that I could borrow?
3. Logged in users are able to post professional favours for others users to view/request.
   Professional favours are favours that are educational or help the person that posted the favours in terms of experience. Some examples include:
   a. Just started to learn how to apply manicures. Looking for people that are interested to let me try on their nails and provide feedback.
   b. Just coded a website. Looking for people that are willing to give the website a try, provide feedback, perform User Acceptance Testing and Automated Testing
   c. I am a student researching sleep cycles. Looking for 40 people to take part in a user study for me to gather data for my research.
4. Logged in users are able to delete posted favours.
5. Logged in users are able to set the deadline of the favours (favours expiry date) and the status of the favours (Urgent/Non-urgent) when creating a favour.
6. Users are able to browse a list of favours posted by other users.
7. Logged in users are able to contact the user that posted the favour (chats feature).
8. Logged in users that post favours are able to mark completed favours as done.
9. Logged in users are able to view a list favours sorted by the following category:
   a. Posted: Favours posted by the logged in user that are not completed and not expired.
   b. Past: Favours posted by the logged in user that are expired or completed.

    c. Helping: Favours posted by other users that the logged in user is helping to complete, and the favour has yet to be expired.
    d. Fulfilled: Favours posted by other users that the logged in user has completed on time.
10. Users are able to view a leaderboard to show the top 4 users that have completed the most favours. There is an all-time leaderboard and a monthly leaderboard.

5. Developer Documentation

5.1 Architecture Diagram

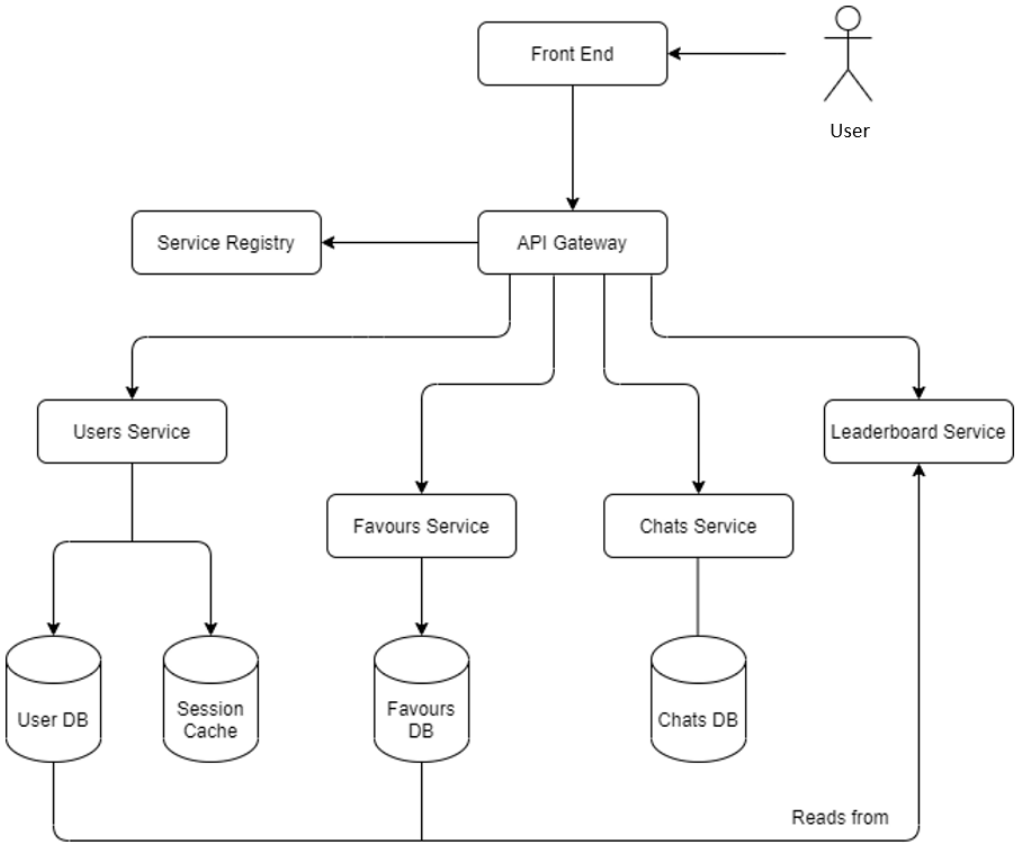


Figure: Favours 4 Uni Architecture Diagram

5.2 Design Patterns and Architecture

5.2.1 Backend

Favours4Uni utilises a microservice architecture, arranging the application as a collection of loosely-coupled services. The application consists of the four microservices:

1) *Users services* - This service orchestrates authentication and authorisation of users to perform particular operations. They control user management operations such as logins, registration, validation, HTTP requests on user profiles and fields and finally control access to privileged operations (e.g. users cannot edit/perform GET requests

other users' profiles). This service interacts with a users database in Firebase, and JWT tokens are dispatched for authentication and stored in the Session Cache.

2) *Favors services* - This service coordinates the organisation and access of information to favours registered in the application by controlling HTTP requests on favours and access to privileged information and operations, determining how favours are organised and delivering the functionality for requests and interactions between users' favours (e.g. requests to complete favours for others, setting completion status of other users for a favour). It interacts with the favours database in Firebase.

3) *Chats service* - This service delivers on inter-user communication functionalities in the form of messages. It coordinates critical communication functionality, allowing users to start chats, the storage of prior messages and user interactions, and the ability to send messages. The messages service also ensures security and confidentiality in user interactions, by implementing route blocks, authentication and authorisation protocols to ensure access to privileged information and operations are controlled. This service interacts with the messages database on Firebase.

4) *Leaderboard service* - This service is derived from special sets of operations on both the users and favours databases, consisting of operations related to accumulation and sorting. It organises favours by category or by time period and their associated users and derives the top four users based on the selected category or time period.

The adoption of a microservice architecture retains the advantage of maintaining relatively low dependencies of one service to other services. This means each service is easily modifiable and extensible, while maintaining the core functionality of the application.

5.2.2 Frontend

The frontend of *Favours4Uni* utilizes a Angular 12, which while is embedded with original Model-View Controller (MVC) is more of a Model-View-ViewModel (MVVM) architectural setup. In this MVVM mode, the framework supports two-way binding of data between the View and the ViewModel. In this design, the ViewModel will use the Observer model to propagate changes from the ViewModel to the Model. As such, the frontend application just requires division into components and the MVVM implementation will handle the rest.

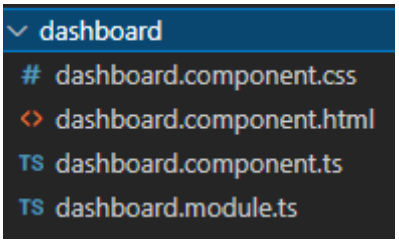


Figure: Component Breakdown

The structure of the Angular Framework is as follows:

View (HTML & CSS Files): Visual Layer of the application, contains HTML, CSS and Angular elements. Connects to ViewModel via data-binding.

ViewModel (TypeScript File): Abstract part of the View which manages the logic related to the models and properties which are shown on the User Interface (UI).

Model (App Services): Provides the link to the backend through API calls.

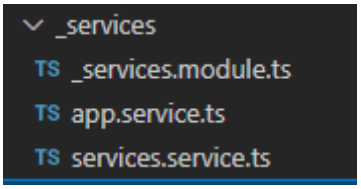


Figure: App Services

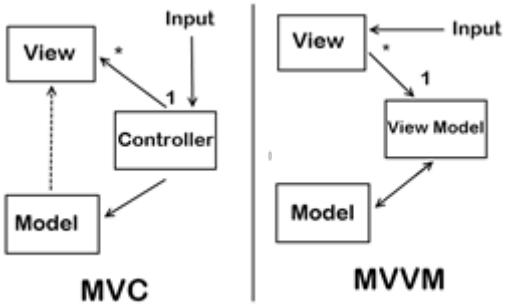
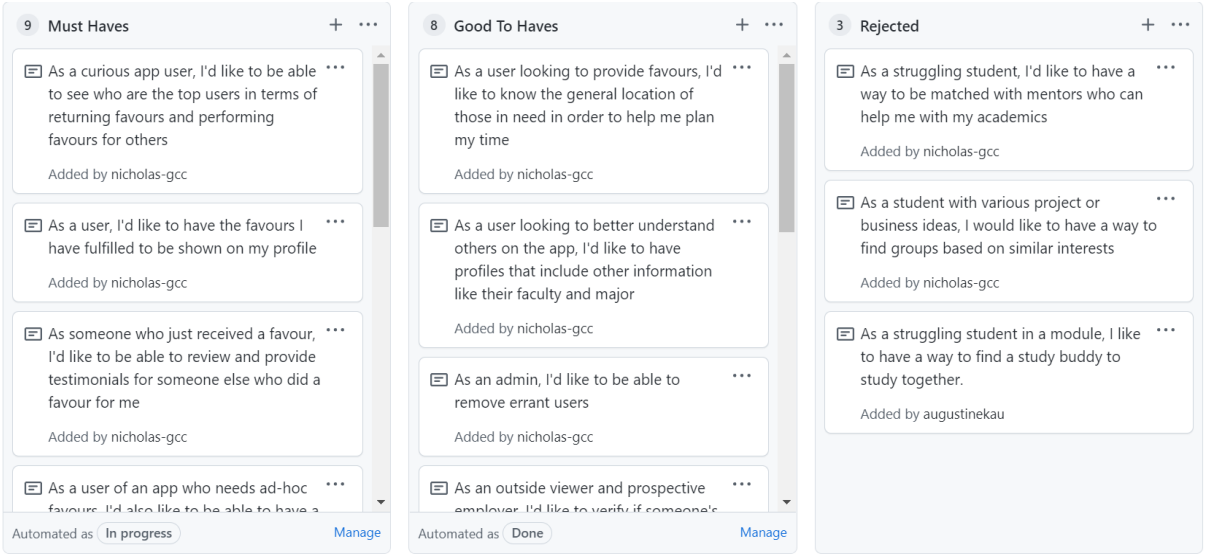


Figure: MVC/MVVM

5.3 Development Process

5.3.1 Ideation

The initial process of deciding of features utilised an empathy-driven approach hinged on User Story philosophies, aided by the use of Automated Kanban Boards. Through this process, we were able to decide on features to implement for the application, and organise them according to priority as shown below:



5.3.2 Tech Stack

While the team initially decided on using the MEAN (MongoDB, Express, Angular and Node.js) stack to develop *Favours4Uni*, the initial idea to use MongoDB was scrapped in favour of Firebase for the main reason that the developers were more familiar with Firebase. More specific considerations for selection of a non-relational database are covered in section 5.4.3. However between both of these options, the choice of tech stacks had the same reasons that:

1) Since all the frameworks are written in JavaScript or TypeScript for client and server-side code, the use of similar languages across both tiers made context-switching much simpler and thus easier to integrate

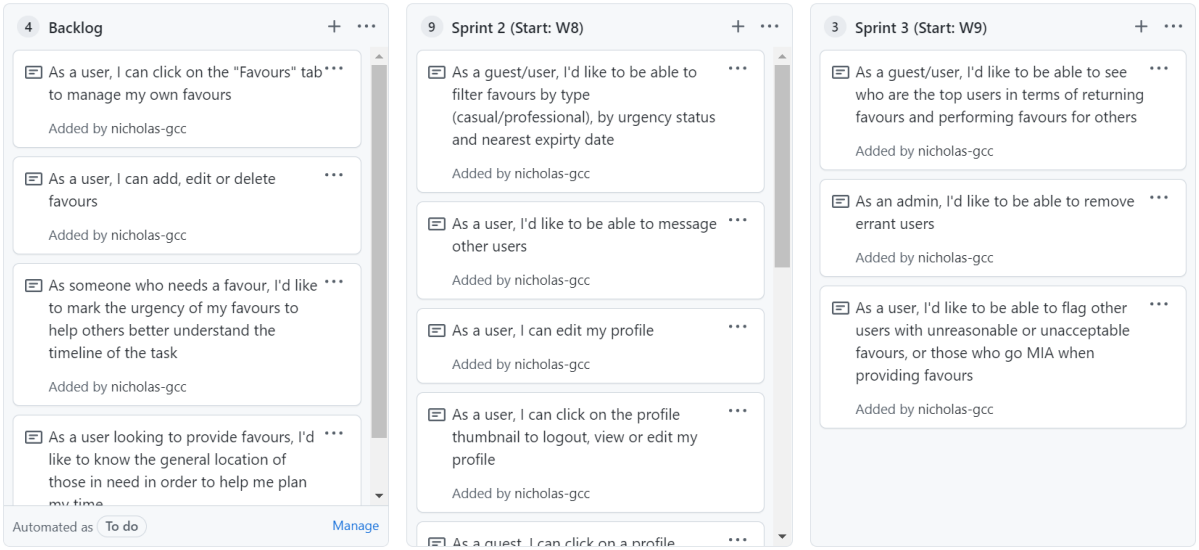
2) These frameworks have extensive libraries and methods to support the REST API and the middleware between frontend and backend, thus facilitating the development of a full-stack web application easily accessible across many devices

5.3.3 High-Fidelity Prototypes (Frontend)

In the early stages of frontend development, the team developed high fidelity prototypes through Canva to draft the client-side design of the application. This included the drafting and ideation of pages, components and the rough design of the application's elements.

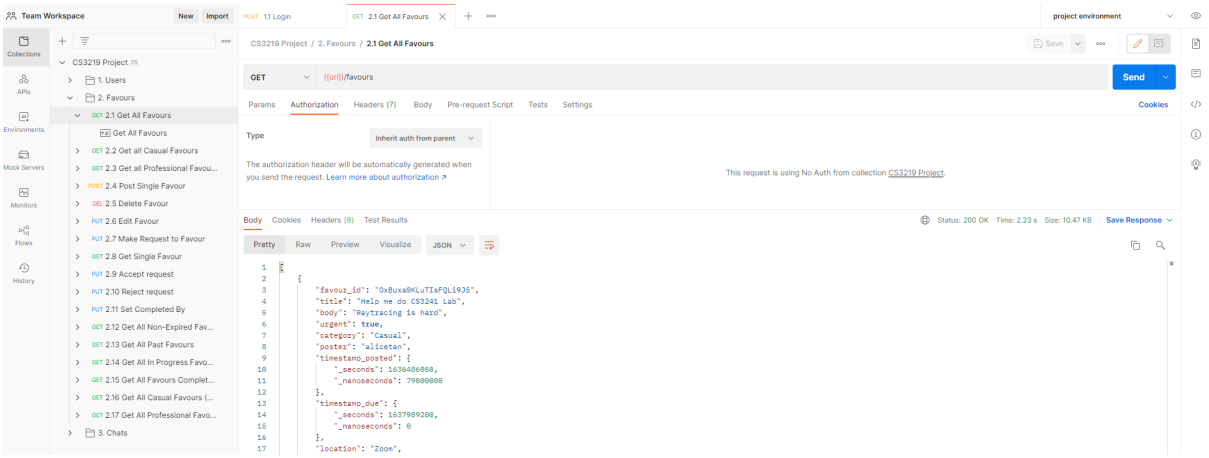
5.3.4 Sprints and Codebase Development

In the development of the codebase, the team adapted the Agile methodology for this project. We conducted weekly sprints with frequent stand-up meetings when necessary to update on individual progress and kept a record of prioritised product backlogs on an Automated Kanban Board as shown below:



5.3.5 Postman

We set up a team workspace on Postman with our project environment to facilitate easier testing of the APIs. The backend team updated the workspace whenever a new API was created, and added examples to show successful requests and edge cases. This allowed the frontend team to easily access the APIs and also give feedback for what to include in the API response and how it should be structured.



5.3.6 App Deployment

Our application is integrated into a CI/CD pipeline, and is automatically deployed via Github Actions. The deployment is done via two workflows, one for frontend and one for backend. It is hosted on Google Cloud (Firebase) Functions, as well as Google Cloud Hosting.

```
      - name: Cache node modules
        id: cache-nodemodules
        uses: actions/cache@v2
        env:
          cache-name: cache-node-modules
        with:
          # caching node_modules
          path: node_modules
          key: ${{ runner.os }}-build-${{ env.cache-name }}-${{
hashFiles('**/package-lock.json') }}
          restore-keys: |
            ${{ runner.os }}-build-${{ env.cache-name }}-
            ${{ runner.os }}-build-
            ${{ runner.os }}-
      - name: Install Dependencies
```

```
      if: steps.cache-nodemodules.outputs.cache-hit != 'true'
      run: |
        cd Frontend
        npm ci
    - name: Build
      run: |
        cd Frontend
        npm run build
    - name: Deploy
      run: |
        cd Frontend
        npm install -g firebase-tools
        firebase deploy --token ${{ secrets.FIREBASE_TOKEN }}
```

Figure: Backend Deployment yml

Node Module caching is used on the frontend deployment due to the sheer number of node modules that are used, and this allows for the deployment to run faster and more efficiently.

```
    - name: Install Backend Dependencies
      run: |
        cd Backend/functions
        npm ci

    - name: Deploy Firebase
      run: |
        cd Backend
        npm install -g firebase-tools
        firebase deploy --token ${{ secrets.FIREBASE_TOKEN }}
```

Figure: Backend Deployment yml

For both deployments, it uses the FIREBASE_TOKEN stored in Github Secrets for CI authentication for deployment. This is due to the fact that we are using Github Actions in our CI/CD workflow, and Secrets functions as hidden environment variables.

Our hosted web application can be viewed at:
https://favours4uni.web.app/dashboard

Our deployed backend API can be requested at:
https://us-central1-favours4uni.cloudfunctions.net/api

5.4 Design Decisions and Implementation

5.4.1 Code Organisation and File Structure - Backend

Client-side code and server-side infrastructure were kept in separate directories developed in parallel to each other, with consistent code integrations in each sprint iteration to minimise failures and ensure sufficient upkeep of the application's functionality.

The code organisation of the server-side code sought to mimic microservices architecture, keeping an *index.js* file to act as the middleware API gateway between services and client-side code. This is a snippet of the API gateway from *index.js*:

```
const functions = require("firebase-functions");
const app = require('express')();
const auth = require('./util/auth');
const cors = require("cors")({
    origin: "*"
});
app.use("*", cors);

const {
    getAllFavours,
    getAllFavoursSortedByDue,
    /* (other favour APIs listed here)
            ...
    */
} = require('./APIs/favours')

const {
    loginUser,
    signUpUser,
    /* (other user APIs listed here)
            ...
    */

} = require('./APIs/users')

const {
    postSingleChat,
    /* (other chat APIs listed here)
```

```
            ...
    */

} = require('./APIs/chats')

// API url endpoints for Favours
app.get('/favours', getAllFavours);
app.get('/favours/sort/due', getAllFavoursSortedByDue);
/* (other favour APIs listed here)
            ...
*/

// API url endpoints for Users
app.post('/login', loginUser);
app.post('/signup', signUpUser);
/* (other user APIs listed here)
            ...
*/

// Chats
app.post('/chats', auth, postSingleChat);
/* (other chat APIs listed here)
            ...
*/
```

Figure: Backend Deployment yml

Do note that because the Leaderboard service directly performs operations on the Users database to accumulate the top few users on an application level, the Leaderboard service APIs are grouped together with User APIs. This is the file structure for the backend:

```
+-- firebase.json
+-- functions
|   +-- API
|   +-- +-- favours.js
|   +-- +-- users.js
|   +-- +-- chats.js
|   +-- util
|   +-- +-- admin.js
|   +-- +-- auth.js
```

```
|   +-- +-- validators.js
|   +-- +-- config.js
|   +-- index.js
|   +-- node_modules
|   +-- package-lock.json
|   +-- package.json
|   +-- .gitignore
```

5.4.2 Code Organisation and File Structure - Frontend

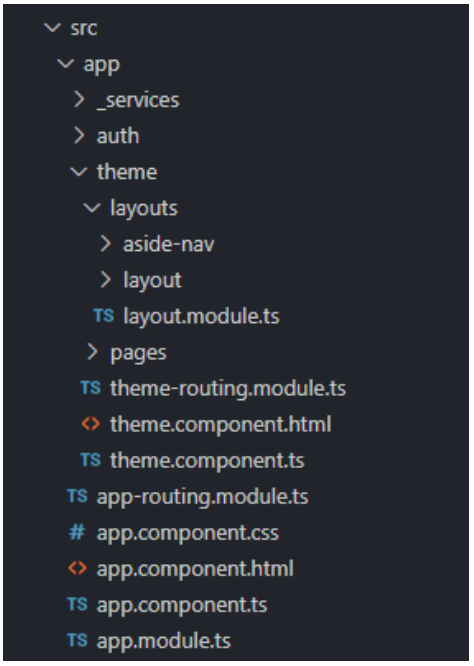


Figure: Frontend Source Directory Breakdown

The frontend (FE) code is organised into its respective responsibility.

1. *_services*: app.service.ts contains all functions that are used across the app and not specific to any pages. For example, functions for http call, setting and getting of tokens exist here. All API endpoints are in the file services.service.ts which is an interface that extends to app.service.ts via Angular injectable.
2. *auth*: files that are authentication related are extracted here, including login and register page. *accessibility.ts* is subpages available on the side nav which can be extended to control which user (with different roles) has access to which side-nav buttons/pages.
3. *theme*: responsibility for all the pages of the website. *layout* determines the layout of the website (side-nav bar with main content); *aside-nav* contains the logic and styling of the side nav bar; *pages* consist of all the pages (dashboard, favours, leaderboard, message, profile).

Modules are separated and splitted into its respective responsibility instead of having all modules in *app.module.ts*. This modularization prevents a monolithic app module which increases coupling and dependencies within pages as rules of encapsulation will be enforced. This also allows for a cleaner code and easier scalability - for example, new pages can be added with ease.

5.4.3 Database

The application interacts with a Firebase database. The main advantage of using a NoSQL framework like Firebase as the database lies in the flexibility offered by the document-collection model of Firebase, its non-relational nature and lenient schemas. This meant that for an application that was quickly evolving in scope, functionality and data usage, it was relatively simple to make changes to the schema and perform successful HTTP REST operations on our application, while also ascertaining that the application could correctly interact with the database. The final database schema for the three collections, *users, favours* and *chats* are as follows:

1) Users

```
bio -> String containing biographic information regarding a user

createdAt -> timestamp object to store creation time and date of user

email -> String containing information for user email, used to log in

favoursCompleted -> Array of objects containing information on favours
completed by user
      {date_completion -> timestamp object to store completion date
       favour_id -> string containing id of the favour completed
       type ->  string containing the category of the favour}

firstName -> string containing first name of the user

lastName -> string containing last name of the user
```

```
phoneNumber -> string containing phone number of user

telegram -> string containing telegram handle of user

userId -> string containing auto generated identifier for user

username -> string containing username of the user. Also set as the
document id
```

2) Favours

```
favour_id -> Automatically generated document ID by Firebase

title -> string containing title of favour

body -> string containing description of favour

urgent -> boolean indicating urgency of favour

location -> string describing location of favour

timestamp_posted -> timestamp of when favour was posted

timestamp_due -> timestamp of when favour is due to be finished

poster -> string containing username of the user who posted favour

category -> string containing the type of favour posted (Professional or
Casual)

completedBy -> string containing user who completed that favour.
Default: null

requestors -> Array of objects containing information on requests made
to favour
```

```
    {isAccepted -> integer containing status of request
     message -> string containing request message to the favour
     username -> string containing username of requestor}
```

3) Chats

```
messages -> array of objects containing message information
    {message -> string containing message to be delivered
     sender -> string containing username of sender
     time -> timestamp describing time of message send}

parties -> string array of size 2 indicating username of sender and
receiver
```

5.4.4 User Authentication and Security

The application makes use of Firebase authentication APIs, which upon successful authentication (via login or registration), releases a JWT token that is stored in the session cache for authentication. Authentication is configured and handled in our *functions/utils/auth.js* file where authorisation tokens are dispatched in successful authentications

```
const { admin, db } = require('./admin');

module.exports = (request, response, next) => {
    let idToken;
    if (request.headers.authorization &&
request.headers.authorization.startsWith('Bearer ')) {
        idToken = request.headers.authorization.split('Bearer ')[1];
    } else {
        console.error('No token found');
        return response.status(403).json({ error: 'Unauthorized' });
    }
    admin
        .auth()
        .verifyIdToken(idToken)
```

```
        .then((decodedToken) => {
            request.user = decodedToken;
            return db.collection('users').where('userId', '==',
request.user.uid).limit(1).get();
        })
        .then((data) => {
            request.user.username = data.docs[0].data().username;
            request.user.imageUrl = data.docs[0].data().imageUrl;
            return next();
        })
        .catch((err) => {
            console.error('Error while verifying token', err);
            return response.status(403).json(err);
        });
};
```

This authentication is used as a route guard in our API gateway, as well as constraining privileged HTTP requests on each microservice.

For instance, authentication is used to guard API routes involving POST, PUT or DELETE requests on favours for unauthenticated users or favours of other users, while allowing GET requests for unauthenticated users to allow viewing of favours. These are the restrictions put in place in our API gateway file *index.js* where a field *auth* is required

```
/*
Other code here
...
*/
const auth = require('./util/auth');

// Favours
app.get('/favours', getAllFavours);
app.get('/favours/sort/due', getAllFavoursSortedByDue);
app.get('/favours/get/:favourId', auth, getSingleFavour);
app.post('/favour', auth, postSingleFavour);
app.delete('/favour/:favour_id', auth, deleteFavour);
app.put('/favour/:favour_id', auth, editFavour);
/*
More favours APIs below
```

```
...
*/
```

In the microservice files such as *favours.js*, we've also added authentication-based constraints to prevent already-authenticated users from performing operations that they do not have the permission or authority to perform. Such an instance is shown in our API for deleting favours as follows:

```
exports.deleteFavour = (request, response) => {
    /*
       Other code here
       ...
    */
    let document=db.collection('favours').doc(`${request.params.favour_id}`);
    // Get favour poster and store in a variable called user

    document.get()
    .then((doc) => {
        if (!doc.exists) {
            return response.status(404).json({ error: 'Favour not found' });
        }
        if (doc.data().poster != request.user.username) {
            return response.status(401).json({ error: 'Not authorised to delete
favour' })
        }
    });
      /*
       Other code here
       ...
     */
   };
```

Such a pattern of authentication-based safeguards in our API gateway and individual microservices are distributed to all of the other services to ensure two layers of security and protection.

5.4.5 User Constraints and Validation

As an extension to security protocols delineated in section 5.4.3, the application also contains a User Validation layer (in Backend/function/util/validators.js and Backend/function/APIs/users.js) to apply constraints during user registration as an additional

safeguard to prevent rogue HTTP requests, or susceptibility to hacking. These include enforcing uniqueness of usernames to prevent access to user-specific operations, making use of Firebase Authentication API's password strictness requirements, regular expressions to enforce proper email formatting and disallowing empty user fields to maintain user profile integrity.

```
const isEmpty = (string) => {
    if (string.trim() === '') return true;
    else return false;
};

exports.validateLoginData = (data) => {
   let errors = {};
   if (isEmpty(data.email)) errors.email = 'Must not be empty';
   if (isEmpty(data.password)) errors.password = 'Must not be  empty';
   return {
       errors,
       valid: Object.keys(errors).length === 0 ? true : false
    };
};

const isEmail = (email) => {
    const emailRegEx =
/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1
,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
    if (email.match(emailRegEx)) return true;
    else return false;
};
```

Code snippets in validators.js to show enforcement of email regex and ensuring fields are non-empty

```
    let token, userId;
    db
        .doc(`/users/${newUser.username}`)
        .get()
        .then((doc) => {
            if (doc.exists) {
                return response.status(400).json({ username: 'this username is already
taken' });
            } else {
                return firebase
                        .auth()
```

```
                    .createUserWithEmailAndPassword(
                        newUser.email,
                        newUser.password
                );
        }
    })
```

Code snippets in users.js to show enforcement of username uniqueness

6. Improvements and Enhancements

With more users in the future, we foresee that some would post irrelevant or inappropriate favours. As such, we plan to allow other users to flag such favours. Furthermore, we plan to add an admin role to manage users. Admins will be able to delete favours deemed inappropriate or irrelevant and will be involved in managing users, including removing users who repeatedly post inappropriate/irrelevant favours, and promoting a fellow user to an admin.

As mentioned in our project requirements, *Favours4Uni* allows users to post *professional* favours pertaining to educational or work matters, such as needing help to build a website, or aiding in a research study. As such, we value the quality of work done when a user takes up a favour. To better ensure this, we plan to allow users to write testimonials and reviews for users who fulfilled their favour. This will hopefully encourage users to do quality work, which is especially important for professional favours as they pertain to educational or work matters.

Currently, *Favours4Uni* supports users posting favour requests. For future releases, we plan to include professional service providing. Users can post professional services for others to take up in addition to favour requests. Some examples include (not limited to) providing manicure services, graphic designing and module consultations. By incorporating this new feature, it aligns with our team's vision of promoting kindness and greater support among the NUS community. Students are also able to hone their skills by providing professional favours.

Lastly, we plan to adapt the UI for mobile support as it currently only supports web browsers on laptops.

7. Reflections

The process and journey of the project was a meaningful and memorable one. For most of us, it was the first time that we were working on this particular tech stack and it was a difficult and fulfilling experience developing this application. As we embarked on the greenfield project option, the first challenge we had was to ideate and come up with an application idea that we felt was appropriate, and that would be useful to a target audience. Here, I believe our experience from CS2103 helped and we managed to narrow down our target group efficiently. In addition, we decided to come up with an idea that we ourselves as university students would use, and this allowed us to learn how both the clients and developers think in the development phase of the application.

In terms of project management, the team was split into half frontend and half backend teams, with 2 on each side. There were many challenges such as time management, meeting deadlines as well as adapting and responding to differing requests from both sides. This was a good learning experience as we learnt how to communicate across teams, and work within the constraints of the implementation. For example, there were times where API calls had to be added to accommodate different frontend features, and to allow a better abstraction of the logic to the server. In addition, with many team members still new to the frameworks used, it was a good learning experience as we debugged and learnt the characteristics of each component.

Being a team project, communication also became an important theme in the development process. We were probably too ambitious when it came down to planning our sprints, with us underestimating the time needed to complete the tasks as well as juggle the workload from other modules. Developing the application was also difficult when team members were working in different trajectories, when it could have been streamlined to provide a better end result. It was definitely important that we learnt to communicate with one another when we couldn't meet the deadlines or weren’t familiar with a certain process, and this helped when pushing toward the final sprint. Communication then helped during the crunch time where we played to each other's strengths, and worked around clashing schedules. This gave us a very realistic experience of being in a high speed work environment, and taught us many things about working in a development team.

In the area of project specifications, we learnt how to be adaptable and change things quickly. For instance, once communicated across teams, requirements can be changed in an

hour to reflect a more logical flow of implementation. The project also gave us genuine insight into how an Agile Scrum structure/workflow is, and also taught us how to better plan our tasks for future projects.

Overall we found the project to be something we would definitely remember, and one where we learnt many lessons and gained much experience from. The technical skills such as the frameworks that we learnt are definitely applicable to any software engineering job in the future, and would be useful if required. It also reinforced the value of communication and taught us the importance of teamwork and professionalism in doing our tasks. We are definitely confident that we are now better equipped to take on greater projects in the future.

8. References:

K. Maaoui, “Why angular is your best choice for you next projects ?,” *Medium*, 15-Oct-2019. [Online]. Available: https://medium.com/@maaouikimo/why-angular-is-your-best-choice-for-you-next-projects-9d754fb18f91. [Accessed: 10-Nov-2021].

L. Marx, “Is angular 2+ MVVM?,” *malcoded*, 02-Dec-2016. [Online]. Available: https://malcoded.com/posts/angular-2-components-and-mvvm/. [Accessed: 10-Nov-2021].