

National University of Singapore  
School of Computing  
Computing for Voluntary Welfare Organisations (CVWO) AY2022/23

## Assignment: **Gossip with Go/Rails**

Issue date: 5th December 2022

Due date: **30th December 2022 (mid-assignment submission)**  
**25th January 2023 (final submission)**

### 1 Overview

You will be making a **simple web forum** in this assignment. Through the assignment, you will learn about the programming skills required for CVWO's summer projects and go through a website development cycle. The focus of this assignment is to maximise learning of proper code structure, coding techniques and familiarity with various development tools. You are expected to build up a good web development foundation through this assignment, and prepare yourself for a development cycle during your stint over the summer.

To achieve the above, this assignment is split into several levels. You will incrementally gain an understanding of different web development principles. At the end, you will produce a simple web application that allows several people to discuss topics and collaborate online.

<b>Reminder:</b> Please read the entire assignment before starting.
---

## 2 Assignment breakdown

This assignment has 5 levels. Level 1 and 3 are compulsory while Level 4 is highly recommended. The other levels are provided as stepping stones or to increase your learning.

### 2.1 Level 1: Make a plan

*“If You Fail to Plan, You Are Planning to Fail.”*

—Benjamin Franklin

This level starts your development journey by introducing some ways to plan for a software project. In this case, you will be planning for a **web forum**. Remember to look through the entire assignment before diving into the level!

#### 2.1.1 User requirements

Before making any software, we need to be clear about the purpose of the software. Who is it for? What can people do with it? This is when you gather software requirements. As a user, think about the features of a forum that is useful to you. You may realise that your ideas are all over the place. Hence, there are different methods that developers use to organise requirements. For example, **user stories** and **use cases**.

#### 2.1.2 Overview of tools

A software developer also needs to plan the technical aspects of the app. In this assignment, you will create a web application. Your frontend code must be written in **React.js** and your backend code can be written in either **Ruby on Rails** or **Go**. If these terms are completely new to you, do not worry about the programming aspects yet. Instead, get an overview about different parts of a web application by looking up some of these terms:

- Frontend
- Backend
- MVC
- **RESTful APIs**

#### 2.1.3 Data

Next, what kind of data will your app need? How can you keep track of the data? Use these resources to help you:

- [What is a relational database and schema?](#)

### **Deliverable**

Submit a writeup of your execution plan by 30th December 2022 through [this google form](#). This is the mid-assignment submission. It should include:

1. Use cases for your application.
2. Whether you plan to implement a Rails or Go backend.
3. Anything else you wish to include in your plan.

This write-up must be no longer than 3 A4 sides, with a 12pt. Georgia or equivalent font-face. Please submit in pdf format and use the following naming convention: `YourFullName_MatriculationNumber` e.g. `JohnTeo_A9876543U`

## **2.2 Level 2: Getting started (Optional)**

This level is a stepping stone to help you familiarise with different tools before you make an entire web application. It also includes snippets about good coding practices which will smoothen your software development journey in the long run. It may seem daunting to pick up many frameworks, but independent learning and being resourceful is key to staying ahead in CVWO and beyond.

### **2.2.1 Tutorials**

If you have never used any of these tools before, get some practice with these short tutorials. They are not compulsory so do what works best for your learning. Apply coding principles that you learned in introductory CS modules to make small functionalities work first.

- [Git](#)
- [React](#)
- [Typescript](#)
- [Ruby on Rails](#)
- [Go](#)

### **2.2.2 Best practices**

Learn about some [good coding practices](#) and apply them as much as you can during this assignment. Benefits of good coding practices include writing bug-free code and making a project more extensible.

### 2.2.3 Practice ground

This **sample React application** is provided to help you experiment and practice what you learnt from the tutorials. Try this for a start:

1. Fork the git repo containing the sample React app.
2. Implement a functionality in the sample React app to allow users to add their own comments to an existing thread.
3. Improve on the code.

#### Deliverable

Nothing. If you worked on the sample React application, you can choose to continue building on it for your full application and submit it in Level 3.

## 2.3 Level 3: Implement your app

In this level, you will implement your web forum. Your app should fulfill the following technical requirements:

1. **Frontend:** You are required to use **React.js** as your main framework to render your web pages.
2. **Backend:** The backend must be written in either **Ruby on Rails**, or **Go**.  
For Ruby on Rails, there are gems (libraries) that allow you to integrate React components within Rails. You may refer to [this guide](#) to get started.  
Implementing a Go backend is significantly harder than doing so Rails. However, we have provided [a skeleton project in Go to help you get started](#). You may also refer to the [resources](#) for a better understanding.
3. **Database:** Any Relational Database is allowed, such as **MySQL**, **PostgreSQL** or **SQLite**. Implementations using non-relational databases such as *MongoDB* will be disqualified.

We highly recommend the usage of **RESTful APIs** to integrate your frontend with your backend. If you use RESTful APIs, you can incorporate work from Level 2 into your backend in this level.

### 2.3.1 Basic Requirements

Your forum should have the following:

1. A simple authentication system where users are authenticated *solely* by their user name (a system that is similar to [when2meet.com](#)).

2. The ability to perform basic CRUD operations for forum threads and comments. CRUD is an acronym for Create, Read, Update and Delete.
3. A category/tagging system to categorise threads so that they are easier to search for.
4. Any other features to improve overall user experience (Recall Level 1 when you mapped out user requirements)

An example of a minimal viable product that we are expecting is something similar to **Telegram's Bugs and Suggestions platform**.

**Note:** Account-based authentication is an optional requirement that is defined in **Level 5**.

#### **Deliverable**

1. A `README` file containing your name and matriculation number.
2. Source code of your application.
3. A short write-up on what you feel about your accomplishments in this assignment as well as a short user manual. This write-up must be no longer than 3 A4 sides, with a 12pt. Georgia or equivalent font-face. Please submit in pdf format and use the following naming convention: `YourFullName_MatriculationNumber_FinalWriteup` e.g. `JohnTeo_A9876543U_FinalWriteup`

Submit a link to the git repo containing all the above by 25th January 2023 through [this google form](#).

## **2.4 Level 4: Deploy your app (Optional, but highly recommended)**

During the development phase, you will likely be running and hosting your application on your local machine, and therefore only you will be able to access it. Deployment means making the application available for use to the public. To do so, you need a server to build and run your application and to accept client requests. In addition, a domain name is necessary so that people can access your application through a URL.

## Render

For this assignment, one option for deployment is **Render**. Render is a platform as a service (**PaaS**) which provides an easy-to-use platform to deploy a web application for free. [This tutorial](#) teaches you how to set up a Rails project on Render. Meanwhile, [this tutorial](#) is for setting up a Go project (with Gin framework) on Render.

## Heroku

Another option for deployment is **Heroku**. Heroku is also a **PaaS**, and there are tutorials to help you get started setting up [a Rails project](#) and [a Go project](#) on Heroku. After a proper setup, you can deploy your application simply by `git push heroku master`.

### Note:

Heroku has ended their free-tier recently. Nevertheless, there is still an option to sign up for the [GitHub Student Developer Pack](#) with your NUS email address to obtain credits for Heroku (and get other benefits!).

## Netlify

If you followed a RESTful API architecture, you can also deploy the React application on **Netlify**. Netlify is similarly a PaaS that helps to deploy frontend web applications quickly. You can use your GitHub account to register for a Netlify account. [This guide](#) runs through how you can deploy a React project on Netlify.

Alternatively, you can use other cloud services such as [DigitalOcean](#) or [AWS Elastic Beanstalk](#) if you wish. You may also claim credits for them from the GitHub Student Developer Pack.

### Deliverable

1. Link to your deployed app.

Submit the link by 25th January 2023 through [this google form](#). Note that this is the same form as the one in Level 3. You only need to submit the form once as your final submission.

## 2.5 Level 5: Further your learning (Optional)

This level introduces additional tools that can be incorporated in your project. Maximising learning across various tools will be useful in your web development journey. Extra credit might be awarded for attempting parts of this level, depending on the completeness of your implementation.

### 2.5.1 Account-Based Authentication

**Difficulty: 3/5** — *Highly Recommended*

Account-based authentication may be tricky to do in an application that has separated frontend and backend instances. Typically after logging in, the frontend will have to cache an authorisation token that is provided by the backend, to be used for authenticating and authorising any CRUD actions that the user performs.

To get started, we recommend using JSON Web Tokens (JWT) in your application for this feature. There are libraries available for you to implement JWT, such as [ruby-jwt](#) and [golang-jwt](#). Ideally, you should cache the authentication token as a cookie on your frontend after logging in, so that users do not have to log in again on subsequent visits to the forum.

### 2.5.2 Cron

**Difficulty: 2/5**

Cron is a time-based job scheduler in Unix-like computer operating systems. It enables users to schedule jobs (commands or shell scripts) to run automatically at a certain time or date. It is commonly used to automate system maintenance or administration, though its general-purpose nature means that it can be used for other purposes, such as connecting to the Internet and downloading email.

**Set a Cron Job under Linux.** This example is to create a cron job to automatically backup a MySQL database.

Inside terminal:

```
> sudo crontab -e

// START THE TOOL TO SET THE CRON JOB
// RUN THE BACKUP SCRIPT EVERY DAY

0 0 * * * bash backup.sh
```

Inside `backup.sh`

```
cd /root/autobackups

mysqldump -u root -<DatabasePassword> <DatabaseName> > <FileName>
```

### 2.5.3 TypeScript

**Difficulty: 3/5** — *Highly Recommended*

**TypeScript** is a superset of JavaScript that adds optional static typing.

Static typing greatly reduces the frequency of type-related errors. Its verbosity also improves the maintainability and scalability, especially for large projects.

We encourage adding **TypeScript** to your project. It may be easier to start with a TypeScript project than to introduce it later. **This guide** provides a good starting point for using TypeScript in your project.

**Note:**

We expect you to declare appropriate types in TypeScript if you integrate it in your project. Usage of `any` everywhere in your code defeats the purpose of TypeScript.

### 2.5.4 Redux

**Difficulty: 5/5**

Redux is an open-source JavaScript library for managing application state and is commonly used with React.

Redux uses what is known as **Flux Architecture** to provide a "single source of truth" for the state of an application. This is especially useful for large projects with many components, as the state of an application can quickly become decentralised and disorganised.

While it is likely not necessary given the scale of this assignment, it is a good exercise to improve your understanding of state management in React.

If you want to give it a go, **this guide** is a good place to get started.

### 2.5.5 Docker

**Difficulty: 5/5**

Docker is a tool designed to package software into standardised **Containers**.

Containers are similar to virtual machines, but unlike a virtual machine, they do not create a whole virtual operating system. Rather, they allow applications to use the same Linux kernel as the system that they're running on but in an isolated environment.

Containers are a solution to the problem of how to get software to run reliably when moved from one computing environment to another. This could be from a developer's laptop to a test environment, from a staging environment into production, and perhaps from a physical machine in a data centre to a virtual machine in a private or public cloud.

As Docker requires some understanding of operating systems and networks, it can be challenging to grasp the concept behind it. Nevertheless, it is a useful skill to pick up if you want to become a more competent programmer. If you are up for the challenge, do check out **this guide** to get started.



### 2.5.6 Amazon Web Services

#### Difficulty: 5/5

**Amazon Web Services (AWS)** is a platform offering many cloud computing services. Its benefits include being scalable and secure.

You may want to use AWS to deploy your app. For example, you can deploy your frontend using an **AWS S3 bucket**. S3 is a cloud object storage service. You could also host your backend on an **Amazon EC2** instance with a reverse proxy server.

AWS requires understanding of operating systems, networks and perhaps even security. It is challenging to understand the different services and choose the right ones for your application in a short timeframe. However, it will give you exposure to cloud services if you choose to attempt it. Beware of costs that can accumulate when using AWS services. To begin, you can sign up for a free **AWS Educate** account and get access to limited services. You can also use the **AWS Free Tier** account. However, remember to check the billing details for each service before you use them.

#### Deliverable

Anything attempted in this level should be submitted as part of the deliverables in Level 3 or Level 4.

## 2.6 Grading Scheme

We grade your assignments based on how much effort you put into learning about web development and the assignment. Below is the brief description of each aspects we look out for in your assignment.

1. **Code Quality:** How well your code is organised, and appropriateness of the data structures (i.e. models declared in your backend).
2. **Logic:** Presence of input validations and bugs, consideration of edge cases for inputs.
3. **UI/UX:** User friendliness of your application.
4. **Effort:** Deployment, documentation and implementation of any optional requirements, together with any other additional features that are not defined in the assignment.

The key point of this assignment is for you to demonstrate what you have learned in this process. Do take note that this list is *non exhaustive*, and the grading is flexible. This grade will be considered during the application process for CVWO.

We will also be running **plagiarism checks** on your submissions. We hope that this assignment will serve as a learning experience for everyone, so please do not copy code either from your peers or from past submissions. 1:1 replicas of online tutorials will also cause you to be disqualified, so be creative and come up with your own design and features!

## 3 Resources

The following section has resources to aid you in your assignment. They include modern programming languages and technologies that are widely used in the industry.

### 3.1 RESTful APIs

Representational State Transfer, REST in short, is an architecture for networked applications. Being one of the simplest architectures to deploy, it is a popular choice for many web services. It is a client-server architecture where a client initiates a request to the server to be processed, and receives a response with updated data. The Application Programming Interface or API defines the specifications of this communication. In the context of this assignment, the server will be our Ruby on Rails or Go application, while the client will be our React application.

Requests and responses can take many forms - JSON is the most common format. JavaScript Object Notation or JSON is a text-based data-interchange format. It is already supported by all modern browsers and server-side scripting languages so you do not need to implement it yourself, but if you want to know more, you can visit [this website](#). You may also want to read up on [HTTP Messages](#), which will be how the JSON data are exchanged.

Note that this is just one of the many ways that a frontend can interface with a backend. If you wish to try this out, do check out [this guide](#) for Ruby on Rails and [this guide](#) for React.

### 3.2 Relational Database

Relational databases are databases that store data in tables, each containing a number of columns and rows. It is called "relational" because tables are linked to other tables through foreign keys. For example, the Singapore Government's address database may use the NRIC/FIN of individuals to link addresses in the `addresses` table to names in the `names` table. In this case, the column containing NRIC/FIN in the `addresses` table is called the foreign key as it references the primary key in the `names` table.

In this assignment, we will be going through fundamental database concepts. Usually, an application will use one database, with several applications sharing the same database server. For instance, if you and a friend each had a blog, each blog needs one database, but the same server could have two databases defined, one for each blog.

At the highest level, a database contains a schema. A schema is a blueprint of your tables, containing their structures and relationships.

Tables can contain one or more columns, each defining an attribute that requires storage. (For example, a `students` table containing students' personal data might contain five columns: `matric_no`, `name`, `address`, `phone`, `date_of_birth`. In

MySQL, columns have types: in this case, name is of type text (`varchar(255)`) and `date_of_birth` is of type `datetime`).

Actual information is stored as rows in a table. Each row must be uniquely identifiable; if two rows cannot be distinguished from each other, then there is no way to change the content or delete one of the rows. Therefore, it is common practice to dedicate one column that is guaranteed to be unique for each row. We call such a column a primary key. In this instance, the `matric_no` is probably a good choice to be the primary key as it uniquely identifies a row, since no two students can share a matric number. You will be prevented from inserting a new row when another row with the same primary key already exists in the table.

Relations indicate relationships between two tables. For example, the `home_faculties` table may contain two columns: `matric_no` and `faculty`, indicating a mapping from a student to her faculty. We can link this to the `students` table by using `matric_no`. As discussed earlier, `matric_no` in the `home_faculties` table is called the foreign key. The foreign key must map into the primary key column set of another table. Note also that students may become members of two faculties when doing double degrees.

Other important concepts include *indexed columns* (where searching within the column is fast, at the expense of increased time required for modification), *unique keys* (which enforce uniqueness among values for non-primary key columns), and *relation cascading* (where deleting a row from a table can automatically update/delete all related entries in tables which reference this key).

After this section, you should be ready to produce a schema for your application. Consider how efficient your schema will be: How complex will it be to access the most commonly accessed data? Think about the number of queries and the number of tables accessed to complete a single user query. Your schema should be graphical, with the table names, column names/types, primary keys, and relationships clearly indicated.

Keywords to look up: SQL, primary key, foreign key, Entity-Relationship Diagram. Try to get the overview before diving into details.

What happens if you need to relate two tables together to retrieve an attribute? For instance, you might have a student in the double-degree programme and you need to retrieve his CAP for each of his faculties.

This is called an Entity-Relationship Diagram.

### 3.3 Git

It might be your first time developing any piece of code which is non-trivial to write and maintain. As such, you might like to store copies of your work over time. This is called versioning. An additional function software engineering teams require is the ability to share code as it is being written. One such software used by software engineers (and CVWO) to solve both problems is [Git](#). Git is however far more able than this and more information can be found [here](#).

It will save you time and effort to use a versioning software since you will most likely run into bugs as you change (supposedly unrelated) pieces of code in your application.

### 3.4 HTML & CSS

HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) are two of the core technologies for building Web pages. HTML provides the structure of the page, while CSS provides the (visual and aural) layout. Along with graphics and scripting, HTML and CSS are the basis of building Web pages and Web applications.

Below is a list of HTML tags you will frequently be using.

```
<a>, <body>, <br>, <button>, <div>, <em>, <form>,
<head>, <h1> - <h6>, <hr>, <html>, <img>,
<input>, <label>, <legend>, <li>, <option>, <p>, <script>,
<strong>, <style>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>,
<thead>, <title>, <tr>, <ul>.
```

Please go through the tutorial from [HTML Dog](#). You can experiment with various HTML tags on [JSBin](#). We will not formally introduce CSS as it is better to learn from practice. Later sections will cover CSS briefly as the need arises.

### 3.5 JavaScript

JavaScript is a scripting language used to create and control dynamic website content, i.e. anything that moves, refreshes, or otherwise changes HTML or CSS elements on your screen without requiring you to manually reload a web page.

If you did not take CS1101S: Programming Methodology, [Eloquent JavaScript](#) is a good starting point. You are not expected to master JavaScript; however, you must demonstrate basic understanding. If you already know basic JavaScript syntax, you might like to start from Chapter 11 - Web programming: A crash course. Alternatively, you can refer to existing JavaScript examples on the web, and use the book to clarify any doubts.

Since JavaScript is used in React and React is an integral part of this project as well as CVWO projects, well-written JavaScript is very important for CVWO.

**Suggestion:**

Modern browsers all come with a debugging inspector for HTML, CSS and JavaScript. You should decide on which browser you will be primarily debugging and testing with. However, note that there are compatibility differences between browsers, and you **must** test your application using different browsers. For CVWO, we usually recommend VWOs to use Firefox or Chrome, so do conduct smoke tests on both of the browsers before your submission.

### 3.6 React

**React** is a JavaScript library for building user interfaces. It is the view layer for web applications.

At the heart of all React applications are components. A component is a self-contained module that renders some output. We can write interface elements like a button or an input field as a React component. Components are composable. A component might include one or more other components in its output.

Broadly speaking, to write React apps, we write React components that correspond to various interface elements. We then organise these components inside higher-level components which define the structure of our application.

For example, take a form. A form might consist of many interface elements, like input fields, labels, or buttons. Each element inside the form can be written as a React component. We would then write a higher-level component, the form component itself. The form component would specify the structure of the form and include each of these interface elements inside of it.

Importantly, each component in a React app abides by strict data management principles. Complex, interactive user interfaces often involve complex data and application state.

You may refer to this [React tutorial](#) to help you understand how to use React.

### 3.7 Ruby on Rails

*“Ruby on Rails® is an open-source web framework that’s optimized for programmer happiness and sustainable productivity. It lets you write beautiful code by favoring convention over configuration.”*

—From the homepage of Rails

Ruby on Rails is a free and open-source web framework. As the name suggests, this framework is written in Ruby. Notice that we have not explicitly asked you to learn the language. This is because the language is relatively easy to pick up, and you should be able to get a sufficient grip on the language by yourself.

Rails abstracts and simplifies common repetitive tasks by eliminating the need to write boilerplate code. There are also many auxiliary libraries that we can use to speed up the development process. Furthermore, it is also relatively easy to deploy as many hosting service providers support it.

Rails usually ships with a default web server (Puma) and database adapter (SQLite), which should be sufficient for this assignment. We do not impose any restrictions on the choice of either for the purpose of this assignment.

Although having a more in-depth lesson on Rails would be appropriate here, we shall stay true to one of Rails’ major philosophies - *DRY*, and leave this task to a more established resource from the community, the Rails Guide <http://guides.rubyonrails.org/>.

### 3.8 Go

Go, also known as Golang, is an open source programming language. It is designed by a team of Google engineers. The syntax in Go is similar to C. As it is a programming language and not a framework, it does not come with default implementations like Rails that abstracts part of the web development process. This requires a finer understanding of how different parts of a web application are connected. For example, learning how to make connections between the database and backend, writing SQL queries and writing boilerplate code to handle every part of data conversion.

However as Go is statically-typed, it reduces type-related bugs which makes the application more secure. As it is compiled, its execution is faster than Ruby which is interpreted during runtime.

You may try [A Tour of Go](#) to get learn the basics of the language and refer to the [tutorial](#) to help you get started with Go.

To get a quick overview of Go, you can also read the talks provided by the Go team on [Go for Pythonistas](#), [Go for Java Programmers](#) or [Go for C++ developers](#).

If you are ready to start building applications in Go, do read through the following resources to have a better understanding of how you can develop and structure applications in Go.

- **Best Practices & Conventions**

- <https://go.dev/talks/2013/bestpractices.slide#1>

- <https://go.dev/talks/2014/names.slide#1>

- <https://go.dev/blog/package-names>

- **Organising Go Code**

- <https://go.dev/talks/2014/organizeio.slide#1>

- **Using a Web Framework**

- <https://github.com/go-chi/chi>

- <https://github.com/gin-gonic/gin>

### 3.9 Best Practices

*"If you give a hacker a new toy, the first thing he'll do is take it apart to figure out how it works."*

—Jamie Zawinski

#### 3.9.1 Abstraction and Modularisation

People who are new to web development usually end up with an unreadable mess. As such, as a project grows in size and scope, you may find it increasingly difficult to maintain and add new features. This is undesirable, especially for large projects.

The key insight here is to abstract common patterns away and reuse them where necessary.

For instance, you may decide to use the same headers and footers for every page in your app. These blocks can be abstracted into separate files and included where required instead of copying and pasting them everywhere. A logical structuring of your files helps increase its maintainability and encourages future code reuse. You may want to take a look at some websites and consider how you could incorporate abstraction.

Moreover, one important aspect of abstraction and modularisation is the organisation of your code architecture. Writing all your code in a single file is a no-go. Just like organising code into functions, your code base needs to be aptly organised into appropriate files and folders. How can we do so? You may realise that code organisation reflects the design and architecture of your code, and it is insensible to attempt to organise your code simply by arbitrarily putting them into folders and more folders. Some common patterns that you can adopt to design your code is the **Model-View-Controller framework**.

### 3.9.2 Documenting Your Work

Programming is all about communicating computational processes. This means that your code should be clear and easily understood. Remember that code is read many more times than it is written!

Help others comprehend your code by including clear comments. However, having good comments does not excuse writing bad code. High quality of maintainable code is expected of you so that subsequent batches of students are able to work on your project. Some tips:

- Give sensible names to variables and functions so that their contents and goals are easily understood.
- Use standard algorithms where available. Mark modified algorithms as such.
- Use the standard library of the language you are using. JavaScript and Rails come with a large standard library that accomplishes many things. If the functionality you require is not in the standard library, find a well-maintained third-party library that accomplishes what you need. Write your own implementation only as a last resort.
- Don't over clutter your code with unnecessary comments. That means you don't have to comment every single line. Comments are excessive when it's blatantly obvious what the code does. A comment on top of each function describing what it does (often) makes your code more comprehensible than multiple in-line comments.

### 3.10 Tools

Various tools can be used for web application development. In all cases, you should always find one that suits your working style. As such, there is no one answer to "which is the best IDE to use". Nonetheless, strictly speaking, any text editor will suffice. However, the following software is used by some of our CVWO members:

- **Visual Studio Code**
- **WebStorm**: Students are eligible for a free copy of the app. [See here.](#)
- **RubyMine**: Students are eligible for a free copy of the app. [See here.](#)
- **GoLand**: Students are eligible for a free copy of the app. [See here.](#)
- **Sublime Text**
- **Vim**
- **AWS Cloud9 IDE**

<b>The end:</b> Good luck and have fun!
---