# Assessment type (🗹)

|  |  |
| --- | --- |
| Checkbox | Type of Assessment / Notes |
|  | Questioning (Oral/Written) |
|  | Practical Demonstration |
|  | 3rd Party Report |
|  | Other – Project/Portfolio (*please specify below)* |
|  | Portfolio Part 1 covering the ability to create, test and document an API. |

## Version Details

|  |  |  |  |
| --- | --- | --- | --- |
| V | Date | Editor | Summary |
| 1.4 | 2025-09-23 | A Gould | Fixes and clarifications to assessment |
| 1.3 | 2025-03-10 | A Gould | Revision of problem scenario to re-use Joke system |
| 1.2 | 2024-08-22 | A Gould | Clarification of requirements |
| 1.1 | 2024-08-22 | A Gould | Update small errors and clarify details when needed |
| 1 | 2024-07-11 | A Gould | New version using updated template |

Table 1 - Version details

**Note:** On the following page is a table of contents to assist you to navigate this document. You may CTRL+CLICK on an entry to jump to that location.

## Parts of the Document

**Assessment Instructions** These are the instructions that must be followed whilst completing the assessment.

**Assessment Instrument** This is where you may be asked questions, required to supply evidence of your work and other specific information as required.

**Appendices** Further required information that are required guidelines but are better placed external to the work to be performed.

# Table of Contents

[Assessment type (🗹) 1](#_Toc209513972)

[Version Details 1](#_Toc209513973)

[Parts of the Document 1](#_Toc209513974)

[Table of Contents 2](#_Toc209513975)

[Assessment Resources 4](#_Toc209513976)

[College to Provide 4](#_Toc209513977)

[Student to Provide 4](#_Toc209513978)

[Assessment Instructions 5](#_Toc209513979)

[Date Due 5](#_Toc209513980)

[Scenario 5](#_Toc209513981)

[Required Features 6](#_Toc209513982)

[Development, Testing & Documentation 9](#_Toc209513983)

[Information Referencing 10](#_Toc209513984)

[Important: Before Commencing 10](#_Toc209513985)

[Step 1: Project Initialisation 11](#_Toc209513986)

[Step 2: Implementation and Testing of Features 12](#_Toc209513987)

[Step 3: Testing and Code coverage 26](#_Toc209513988)

[Step 4: Review Questions 27](#_Toc209513989)

[Step 5: Deploy 27](#_Toc209513990)

[Step 6: Verify 27](#_Toc209513991)

[Step 7: Submission 28](#_Toc209513992)

[Step 8: Demonstration 29](#_Toc209513993)

[Assessment Instrument 30](#_Toc209513994)

[1 Evidence: Analysis 31](#_Toc209513995)

[2 Answer: API 32](#_Toc209513996)

[3 Evidence: Feature Completion 32](#_Toc209513997)

[4 Answer: APIs and Business 33](#_Toc209513998)

[5 Answer: IDEs and Development 34](#_Toc209513999)

[6 Evidence: Code Review 35](#_Toc209514000)

[7 Deployment 37](#_Toc209514001)

[Appendix A: Assessment Submission 38](#_Toc209514002)

[Appendix B: Code Style and Commenting 39](#_Toc209514003)

[Appendix C: Referencing 42](#_Toc209514004)

[Appendix D: Video Recordings 43](#_Toc209514005)

[Appendix E: Portfolio Project Planning 44](#_Toc209514006)

[Using GitHub to Manage a Project 44](#_Toc209514007)

[GitHub Issues with Tasks 44](#_Toc209514008)

[Appendix F: Git & GitHub 45](#_Toc209514009)

[Version Control Requirements 45](#_Toc209514010)

[Commit Messages 45](#_Toc209514011)

[Further Git/GitHub Resources 46](#_Toc209514012)

# Assessment Resources

## College to Provide

* Web App Development Environment:
  + Web server
  + PHP 8.2+ interpreter plus composer
  + Database server (MySQL, SQLite, MariaDB, or similar)
* Internet Access
* IDE or editor for developing PHP applications  
  (only PhpStorm is supported by the college)
* Access to Office 365 & Microsoft Word

## Student to Provide

N/A

If the student uses their own BYOD, then they are responsible for configuration, maintenance and troubleshooting. Lecturers **may** assist in troubleshooting issues with devices at their discretion, and outside of normal class time.

# Assessment Instructions

* Please read these instructions carefully.
* Follow each step as provided.
* Questions will be asked in a separate section of this document, with space provided for your answers.
* Information in the appendices **MUST** be applied to your assessment submissions.

## Date Due

* 5:30PM on day of Session 11

## Scenario

You are employed as a junior Web Application Developer for RIoT Systems (Robotics & Internet of Things), a Perth based educational and development company who specialise in IoT, Robotics and Web Application systems.

You have been tasked with the implementation of a basic REST API to provide the start of a solution for an opportunity for the company.

You are provided with a set of steps to accomplish this (this document).

At any stage during this assignment item, you may consult the stakeholder(s) or their representative(s).

### Opportunity Outline

The company would like to see a REST API created for the Jokes System created in the previous SaaS: Front-End Development cluster.

They have a set of requirements that need to be met, and this may include changes to the previous understanding of the problem.

### Important

This portfolio **DOES NOT** require a web, mobile, or desktop front end user interface to be created.

You will use Postman or an equivalent application, to interact with your API (acting as the ‘client interface’).

## Required Features

The features shown below are given in more details in the steps within the project, this is for outline and to assist in understanding the context of the opportunity outlined above.

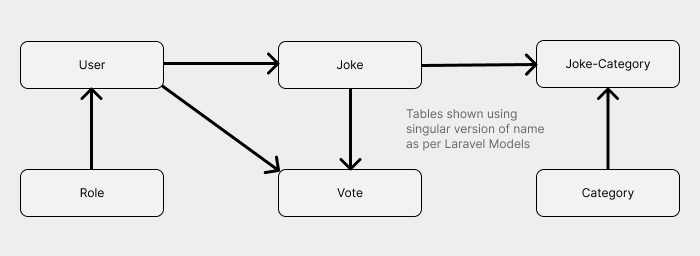
You are to develop API endpoints for the following features:

* Jokes
* Users
* Categories
* Votes
* Authentication

More details will be given in the following sections and may include additional work for the pre-requisite features.

### General: ERD

A possible Database Design is shown below:



Note that a Joke may now have many categories, which encompasses the previous category and tag parts of the application.

### General: Tables

On the following pages are shown a general overview of the tables for the joke database.

No field sizes are given.

| Table | Field | Notes |
| --- | --- | --- |
| User | Id |  |
| Given name | **OPTIONAL**  If you implement the given and family names then the following rules are applied:  At least one of the Family Name or Given Name must be provided. The Family Name will store the name in this case.  The default “name” field is to be used as the nickname/preferred name. |
| Family name |
| Name, Nickname, or Preferred name | You may use Name only, and in this case do not change the defaults.  If you implement a Given and Family name, then this field will represent a nickname or preferred name for the responses from the API.  By default, the nickname is the given name, unless none is provided, then the family name is used. |
| Email |  |
| Password |  |
| Email verified at |  |
| Status   * + This feature may be split into separate fields: Suspended, Active and Banned | **OPTIONAL**  Status is used to identify a user who is ‘Suspended’, ‘Active’ or ‘Banned’.  ‘Suspended’ requires them to change their password and verify their account to become active. When verified, the status changes to ‘Active’.  ‘Banned’ users cannot undo their ban themselves. They must contact an admin. Admin may change banned users to ‘Suspended’.  ‘Active’ is always the status of a user that is not suspended, banned or not verified. |
| Joke | Id |  |
| User Id (Author) |  |
| Joke title | Joke title is a maximum of 64 or 96 characters |
| Joke text | Joke text will contain Markdown. |
| Reference | **OPTIONAL**  Reference is used for adding URI links to the origin of the joke. |
| Category | Id |  |
| Name | Name required |
| Description | Description optional |
| Category-Joke | Id | A joke cannot have the same category more than once, and a category cannot have the same joke more than once. |
| Joke Id | Joke Id and Category Id must be indexed as Unique. |
| Category Id |
| Vote | Id |  |
| User id | User ID and Joke ID must be indexed as unique. |
| Joke id |
| Rating | Users may vote for the joke once. Users may alter their vote (rating).  Rating is a simple thumbs up or down. Thumbs up is equal to +1 points, Thumbs down -1 points.  Average Rating:  (Count of Thumbs up or down – Count thumbs down) / (Count thumbs up or down) \* 100%  E.g. 5 Up, 1 Down 🡪 (6 - 1) / 6 = 5 / 6 = 83.33%  E.g. 7 Up, 0 Down 🡪 (7 + 0) / 7 = 100%  E.g. 0 Up, 3 Down 🡪 (3 – 3) / 3 = 0% |
| Role & Permission | From Spatie Roles & Permissions | This comes from the Spatie Roles and Permissions package and is defined as part of this.  Users may have more than one role.  Users may be given individual additional specific permissions. |

### General: Roles & Permissions

It is possible to define roles and permissions for the features of the API. The table below shows an extract of possible set of permissions that could be used for more defined (granular) access to the features of user administration. Please refer to the User Admin step for precise details.

| User type | Browse (all) | Read (one) | Edit | Add | Delete | Search | Notes |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Client | No | Only Own | Only Own | No | Only Own | No | Soft deletes are used, no access to account recovery |

It is more important to implement the specific actions, roles and permissions as per the following steps of this assessment.

## Development, Testing & Documentation

Development of the Web Application must include:

* PEST (feature) tests must be written for each feature and its components.
* All data must be validated.
* Correct HTTP Responses will be given (200 OK, 201 Created, 404 Not Found, 403 Forbidden, etc).
* A standardised JSON response structure will be used. A common structure is:

{

"success": true|false,

"message": "Some form of message",

"data": {

...

}

}

* The response structure may be amended to include an "errors" field that contains an array of errors.

## Information Referencing

This is COMPULSORY for all assessments and covers resources that include but is not limited to:

* The Internet;
* Books;
* Video;
* Code;
* AI Use; and
* Audio.

More details on referencing requirements may be found in Appendix C: Referencing.

## Important: Before Commencing

**It is very important** that you familiarise yourself with the content of this assessment by reading the whole document at least once before commencing.

As you progress through the steps contained in this assessment document, any questions relating to a step, or required evidence will be added into the Assessment Instrument section.

## Step 1: Project Initialisation

If you have not completed the Jokes Database as a Front-End web application, then before commencing you will need to do the following:

* Create **a new empty private GitHub** (or equivalent) repository:
  + Ensure that the repository does NOT have a ReadMe.md, .gitignore, License or any other files.
* Create a new Laravel application.
* Ensure the application is named the same as your repository name.
* Update the default “.gitignore” as required to make sure that backup files, IDE files and other items are not included in the repository.
* Duplicate the “.env” file and name the copy “.env.dev”, add this to your version control.

If you wish to use your previous work as the basis then make sure you look at organising your code to separate the API from the web front end.

The easiest way to do this is to have the API controllers in a different namespace, App\Http\Controllers\Api\Vn – where Vn is the letter V followed by the version number.

You may use the [https://github.com/AdyGCode/jokes-api-2025-s2](https://github.com/AdyGCode/jokes-api-2025-s2/releases) repository as a starting point for this assessment.

We would suggest that you use the latest code release (<https://github.com/AdyGCode/jokes-api-2025-s2/releases>) from the repository, uncompress and rename the folder. You will then need to initialise version control, and run composer install to install the required framework files.

If you wish to work on a copy of the repository then please fork the code and wok on your own fork.

Pull requests to the original code will be ignored.

## Step 2: Implementation and Testing of Features

You do NOT have to implement these features in the prescribed order.

You may find it easier to implement without authorisation or roles and permissions before adding these to the API.

We will list the features and details associated with each one.

### Feature: Voting (Likes/Dislikes)

The vote feature will allow a client to perform actions on Votes.

The Actions and permissions for this feature are defined as:

| Role | Level | Actions | Notes |
| --- | --- | --- | --- |
| Guest (Unregistered) | 0 | NONE | Unable to vote |
| Client | 100 | May perform SOME of the BREAD actions, which include:   * Add own vote for a joke. * Edit own vote for a joke. * Remove own vote for a joke. | Must be verified before able to perform these actions.  Cannot vote on jokes with an unknown or empty category, or whose category is soft deleted. |
| Staff | 500 | May perform SOME of the BREAD actions, which include:   * Add own vote for a joke. * Edit own vote for a joke. * Remove own vote for a joke.   Plus:   * Clear **all** votes by a user. |  |
| Admin | 750 | All actions of the Staff. |  |
| Superuser | 999 | As above, plus:   * Backup vote data to external location. |  |

Key relationships include, but may not be limited to:

* A joke has zero or more votes
* A vote has zero or one joke
* A vote belongs to one user
* A user has zero or more votes

### Feature: Roles & Permissions

#### Permissions

* No API
* Migration
* Seeder

Permissions are programmatically fixed. We do NOT allow permissions to be added, edited, nor deleted.

#### Roles

* BREADS (CRUD)
* Migration
* Seeder
* Data validation

Roles may be added, edited and deleted.

The Super-Admin role MUST NOT be allowed to be Deleted. Super-Admin always has full access to all features.

#### Permissions & Roles Table

| User type | Browse (all) | Read (one) | Edit | Add | Delete | Search | Notes |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Unregistered,  Client,  Staff | No | No | No | No | No | No | - |
| Administrator | All | Any | Any | Yes | No | Yes | - |
| Super-User | All | Any | Any | Yes | Any  Cannot delete super-admin role | All | - |

### Feature: Authentication

#### Authentication

* Login
* Logout
* Password reset

#### Authentication Pest Tests

Make sure you provide the minimum tests:

* Log in correct user data
* Login with incorrect user data
* Logout
* Force Logout of all users in a role

#### Permissions & Roles Table

| User type | Login | Logout | Register | Reset Password | Notes |
| --- | --- | --- | --- | --- | --- |
| Guest (Unregistered) | No | No | Yes | No | Must register to work on account |
| Client | Yes | Self | No | Yes |  |
| Staff | Yes | All  May logout other individual Client users | No | Yes  Send reset for users who are clients |  |
| Administrator | Yes | All  May logout ALL users with role Staff, Client | No | Yes  Send reset for users who are Staff or Client |  |
| Super-User | Yes | All  May logout all other users including Clients, Staff, and Administrators. | No | Yes  Send reset for any user from any role |  |

### Feature: User (Profile)

The user profile is NOT THE SAME as the User Admin.

Each user has their own profile (name, email. Password etc).

They are allowed to edit their own data.

#### User Profile

* Edit

#### Profile Pest tests

Make sure you provide the minimum tests:

* Update own data
* Unable to access different user
* Able to delete own account

#### Permissions & Roles Table

| User type | Read | Edit | Notes |
| --- | --- | --- | --- |
| Guest (Unregistered) | No | No | Must register to work on account |
| Client | Own | Own |  |
| Staff | Own | Own |  |
| Administrator | Own | Own |  |
| Super-User | Own | Own |  |

### Feature: Categories

The category feature will allow a user to perform actions on Categories. Depending on the user’s role and permissions the actions they may perform include the ability to:

* Browse, read, edit, add and delete,
* Search

For this step, you must:

* Implement the Category API if it is not completed already,
* Create Tests to verify the Category API,
* Run tests on your code,
* Fix any issues when testing is executed,
* Ensure the feature is complete for all roles and permissions.

Whilst performing the above you must:

* Answer any questions in the 2 Answer: API section of this document,
* Add evidence in the 3 Evidence: section within this document.

#### Pest Tests

Make sure you provide the minimum tests:

* Logged in user can retrieve categories
* Unable to access different user
* Able to delete categories created by themselves

#### Categories Roles/Permissions

Permissions for this set of actions are defined as:

| Role | Level | Actions | Notes |
| --- | --- | --- | --- |
| Guest | 0 | None | Unable to interact with categories |
| User | 100 | Search  Browse  Show | Must be verified before able to perform these actions.  When show is used, the user will be presented with five random jokes as part of their data.  Cannot see other people’s jokes with an unknown or empty category, or whose category is soft deleted. |
| Staff | 500 | Perform all BREAD/CRUD actions and search.  Categories:   * Are SOFT DELETED, and * May be RESTORED by this role or higher-level roles. | A category, when deleted, it is moved to the trash can (soft delete).  “Soft deleted” categories are not available to users below the Staff level. |
| Admin | 750 | All actions of the Staff, plus:   * Permanently remove categories (from Trash). * Restore soft deleted categories. | Removal of a category will remove the category from all jokes.  Any joke without a category will then be assigned “unknown”. |
| Superuser | 999 | As above, plus:   * Backup category data to external location. |  |

Key relationships include, but may not be limited to:

* A joke has zero or more categories
* A category has zero or more jokes

### Feature: Jokes

The joke feature will allow a client to perform actions on Jokes. This includes the ability to:

* Browse, read, edit, add and delete,
* Search

Remember that the Joke BREAD has restrictions. These may include

- User/Staff/Admin/Super-Admin - Browse, Read any

- User - Add, Edit, Delete own jokes

- Staff/Admin/Super-Admin - BREAD on all Jokes

#### Overview

For this step, you must:

* Implement the User API,
* Create Tests to verify the User API,
* Run tests on your code,
* Fix any issues when testing is executed,
* Ensure the feature is complete for all roles and permissions.

Whilst performing the above you must:

* Answer any questions in the 2 Answer: API section of this document,
* Add evidence in the 3 Evidence: section within this document.

Continued overleaf…

#### Permissions/Roles Summary

Permissions for this set of actions are defined as:

| Role | Level | Actions | Notes |
| --- | --- | --- | --- |
| Guest | 0 | Register | Can only retrieve ONE random joke.  No unknown category jokes are available. |
| Client | 100 | Search  Browse  Show | Must be verified before able to perform these actions.  When show is used, the user will be presented the data from the requested joke.  Cannot retrieve jokes with an unknown or empty category, or whose category is soft deleted. |
| Staff | 500 | Perform all BREAD/CRUD actions and search.  Categories are SOFT DELETED and may be RESTORED by higher level roles. | Note that a category, when deleted, it is moved to the trash can (soft delete).  “Soft deleted” categories are not available to users below the Staff level. |
| Admin | 750 | All actions of the Staff, plus:   * Permanently remove categories. * Undo soft deleted categories. |  |
| Superuser | 999 | As above, plus:   * Backup category data to external location. |  |

Key relationships include, but may not be limited to:

* A joke has zero or more categories
* A category has zero or more jokes

Continued overleaf…

#### Pest Tests

Make sure you provide the minimum tests:

* A guest is able to generate a rand joke
* Logged in user may perform operations on Jokes based on Permissions table
  + Browse,
  + Read,
  + Edit,
  + Add and
  + Delete.

#### Permissions & Roles Table

The Categories permissions table in detail.

| User type | Browse (all) | Read (one) | Edit | Add | Delete | Search | Notes |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Unregistered | No | No | No | No | No | No | Must register to work on account |
| Client | No | Only Own | Only Own | No | Only Own | No | Soft deletes are used, no access to account recovery |
| Staff | All | All | Only own  Clients  Applicants | Yes Limited  Clients  Applicants | Clients, Applicants  Cannot delete themselves, Administrators or Super-users | All | Soft deletes are used, no access to account recovery. |
| Administrator | All | All | All | All | Clients, Applicants, Staff  Cannot delete themselves, Administrators or Super-users | All | Soft deletes are used  No access to account recovery. |
| Super-User | All | All | All | All | Any  Cannot delete themselves | All | Soft deletes are used  No access to account recovery. |

### Feature: Like/Dislike aka Voting

For this step, you must:

* Implement the Voting API,
* Create Tests to verify the API,
* Run tests on your code,
* Fix any issues when testing is executed,
* Ensure the feature is complete for all roles and permissions.

#### Overview

The Voting API is for adding Likes/Dislikes and modifying these requests.

The Voting (Like/Dislike) may be tackled in several ways. For example:

* Option 1
  + Use Only POST to send value for like on a joke, code determines if Create, Edit or Delete.
* Option 2
  + Use the full BREAD - and have each endpoint actioned as needed (more complex).

Remember that user must be authenticated to like/dislike.

#### Pest Tests

Make sure you provide the minimum tests:

* A guest is unable to vote in any way
* Logged in user may perform operations on Jokes based on Permissions table
  + Browse,
  + Read,
  + Edit,
  + Add and
  + Delete.

#### Permissions/Roles Summary

Permissions for this set of actions are defined as:

| Role | Level | Actions | Notes |
| --- | --- | --- | --- |
| Guest | 0 | NONE | Cannot Like/Dislike jokes |
| Client | 100 | Registered users may perform the following actions on any joke:   * Add a Like or Dislike * Change to Dislike or Like * Remove their vote completely   They may only change/remove their own votes. | Must be verified before able to perform these actions.  May only perform these actions on their own votes.  May perform actions on any accessible joke. |
| Staff | 500 | Basic actions are as per a client user. | As per Client user. |
| Admin | 750 | All actions as per Staff.  Admin may also:   * Remove ALL votes made by a Staff or Client user. |  |
| Superuser | 999 | As above, and they may also:   * Reset the votes for ALL users. * Reset the votes made by any individual user. |  |

Key relationships include, but may not be limited to:

* A joke has zero or more categories
* A category has zero or more jokes

### Feature: User Admin

For this step, you must:

* Implement the User API,
* Create Tests to verify the User API,
* Run tests on your code,
* Fix any issues when testing is executed,
* Ensure the feature is complete for all roles and permissions.

Whilst performing the above you must:

* Answer any questions in the 2 Answer: API section of this document,
* Add evidence in the 3 Evidence: section within this document.

#### Overview

The users feature will allow a user to perform actions on Users. This includes the ability to:

* Browse, read, edit, add, delete (BREAD/CRUD)
* Search
* Add or change user role
* Register
* Login
* Verify email
* Change password

Not all roles will be able to perform all the above actions.

Continued overleaf…

#### Roles/Permissions Summary

Permissions for this User Feature actions are defined as:

| Role | Level | Actions | Notes |
| --- | --- | --- | --- |
| Guest | 0 | Register |  |
| Client | 100 | Verify email | Must be verified before able to perform other actions. |
|  | Edit own user data  Edit own profile  Delete own profile | Ordinary users are **unable** to browse, add, edit, delete, search other users in the system.  They are only able to access their own information.  Deleting own profile removes all jokes they added and all votes they made. |
| Staff | 500 | Edit own data  Edit profile data for users of lower role  Mark user as banned or suspended  May revert suspended user to active |  |
| Admin | 750 | Create users  Edit users  Delete users  Edit roles for those with lower role  Ban users, suspend users, revert with of these statuses. | An admin may revert a banned user to the suspended role. This will enable the user to change their password and verify their email account before being able to use the system.  Deleting a user removes all jokes they added and all votes they made. |
| Superuser | 999 | Full access, including creating new users, assigning roles etc.  That is, all possible actions the lower roles have.  One Superuser only  Backup user data to external location. |  |

Key relationships include, but may not be limited to:

* A user has **one** Role
* A user has zero or more Jokes
* A user has zero or more Votes

#### User Admin Tests

| User type | Browse (all) | Read (one) | Edit | Add | Delete | Search | Notes |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Unregistered | No | No | No | No | No | No | Must register to work on account |
| Client | No | Only Own | Only Own | No | Only Own | No | Soft deletes are used, no access to account recovery |
| Staff | All | All | Only own  Clients  Applicants | Yes Limited  Clients  Applicants | Clients, Applicants  Cannot delete themselves, Administrators or Super-users | All | Soft deletes are used, no access to account recovery. |
| Administrator | All | All | All | All | Clients, Applicants, Staff  Cannot delete themselves, Administrators or Super-users | All | Soft deletes are used  No access to account recovery. |
| Super-User | All | All | All | All | Any  Cannot delete themselves | All | Soft deletes are used  No access to account recovery. |

## Step 3: Testing and Code coverage

Ensure you have implemented Pest tests for the API end point. The test should cover at least:

* Authentication
* Jokes
* The application of Roles & Permissions
* Voting

Complete a review of your code to make sure all the following are completed, and working as expected:

* User feature completed and tested
* Vote feature completed and tested

Perform a Code Coverage using xDebug or a similar system. This code coverage report should show:

* + An overview of the coverage,
  + A more detailed report of for the Users feature,
  + A more detailed report of for the Jokes feature,
  + A more detailed report of for the Votes feature.

Any feature you implement should have at least 50% code coverage.

Place your supporting evidence in the section 6 Evidence: Code Review.

If, during this review, you notice a very large gap in coverage of your tests, it is a very good idea to consider how to improve this.

## Step 4: Review Questions

Either during or after completion of the API, the tests and the documentation of the API, complete the following:

* Answer any questions in the 4 Answer: APIs and Business section of this document,
* Answer any questions in the 5 Answer: IDEs and Development section of this document,

## Step 5: Deploy

Deploy your application to your ScreenCraft hosting using a suitable automated method.’

Identify the details of how you did this deployment.

Provide a Markdown document as part of the submitted code outlining the steps to implement your deployment method.

* Answer any questions in the **7 Deployment** section of this document,

## Step 6: Verify

Before submission and the required demonstration, you should:

* Check that you have implemented the features as required
* Documented your code
* Run and passed your feature tests
* Documented your API
* Created a Postman project with the required collections for the features
* Tested the API via Postman

## Step 7: Submission

For this assessment we require:

* This document with:
  + all questions answered,
  + all required screenshots, and
  + any code that has been requested to be inserted into the document.
* A compressed copy of the Project Code **WITHOUT** the node\_modules and vendor folders.
* A copy of any video evidence recording, **if** it is requested.

|  |
| --- |
| **DO NOT COMPRESS** any of the following when submitting:   * MS Office Documents (including Word, Excel and other files) * PDF Documents * Images (if less than three) * Video Recordings |
| **COMPRESS** the following:   * Project Code (exclude vendor and node\_modules folders) * Images if more than 3 |
| Submissions must be completed on or before the date specified at the beginning of the assessment, unless otherwise indicated. |

## Step 8: Demonstration

You will require approximately 30 minutes with the company representatives (the lecturer) to go over your code and implementation. During this meeting you will:

* Demonstrate features of the API
* Demonstrate the API documentation
* Answer questions about your implementation

A separate checklist will be used for this.

The demonstration will be added as a separate assessment item for easier tracking and feedback.

You may be asked:

* to demonstrate the formulating and development of additional code within the context of the assessment.
* Online students will book for a meeting via video where they will share their screen and demonstrate using the same checklist.

# Assessment Instrument

When a step includes a question, you must attempt to answer it.

There is a minimum and maximum number of words to use for each answer.

If a step has more than one question, these maxima and minima are a total for all the questions in that specific step.

All answers must be in complete sentences unless indicated.

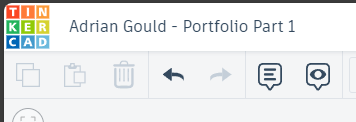
Unless otherwise directed, make sure to add any code you’ve written in a separate file to your submission. Also, unless otherwise directed, DO NOT put code in a Word document.

The following pages contain the areas for evidence and answers to be included.

## 1 Evidence: Analysis

### Screenshot: GitHub Repo

Provide a screenshot of your empty GitHub repository (replace the image below).



## 2 Answer: API

### Questions:

Using your code as a reference point, answer the following questions:

* What are APIs in the context of Software as a Service?
* Why are APIs so important and useful?
* What is REST and how is it important in the context of your API implementation?

Within your answers to the above, provide examples of how your code implemented the requirements of a REST based API.

### Answer:

#### What are APIs in the context of Software as a Service?

#### Why are APIs so important and useful?

#### What is REST and how is it important in the context of your API implementation?

## 3 Evidence: Feature Completion

**No evidence required in this document.**

Code must have been tracked, committed and pushed to your repository.

## 4 Answer: APIs and Business

### Question:

During the development of this portfolio, you were asked to use a particular framework. With this as the context, answer the following questions:

* If you were to select a programming framework to assist you in developing an API, would it be the one used for the assessment, or would you use another framework.

Provide your supporting reasoning for either case, referring to your implementation code within your answer.

### Answer:

#### If you were to select a programming framework to assist you in developing an API, would it be the one used for the assessment, or would you use another framework.

## 5 Answer: IDEs and Development

### Question:

During the development of this portfolio, you were asked to use a given IDE. With this as the context, answer the following question:

* If you were to select an IDE to develop code, would it be the one used for the assessment, or would you use another framework?
* Provide your supporting reasoning for either case. This reasoning should include items such as cost, extensibility, flexibility, support for frameworks and languages, and ease of use.

### Answer:

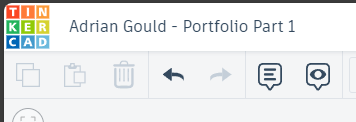
#### If you were to select an IDE to develop code, would it be the one used for the assessment, or would you use another framework?

#### Provide your supporting reasoning for either case. This reasoning should include items such as cost, extensibility, flexibility, support for frameworks and languages, and ease of use.

## 6 Evidence: Code Review

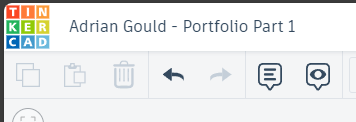
### Screenshot: User Feature Tests

Provide a screenshot of your User feature test results (replace the image below).



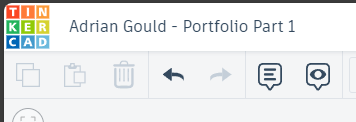
### Screenshot: Voting Feature Tests

Provide a screenshot of your Voting feature test results (replace the image below).



### Screenshot: Code Coverage Reports

Provide screenshots of your code coverage results (replace the image below).



## 7 Deployment

### Question:

During the development of this portfolio, you were asked to deploy the project to an external web service provider.

With this as the context, answer the following question:

* Give a broad overview of the deployment method used.
* How did you employ continuous deployment techniques?
* If you did not employ continuous deployment techniques, how would you go about implementing this?

### Answer:

#### Give a broad overview of the deployment method used.

#### How did you employ continuous deployment techniques?

#### If you did not employ continuous deployment techniques, how would you go about implementing this?

# Appendix A: Assessment Submission

These assessment submission guidelines are common for all submissions in this cluster.

**DO NOT COMPRESS** any of the following when submitting:

* MS Office Documents (including Word, Excel and other files)
* PDF Documents
* Images (if less than three)
* Video Recordings

**COMPRESS** the following:

* Project Code (exclude vendor and node\_modules folders)
* Images if more than 3

Any single submission must contain all required components unless stated.

Submissions must be completed BEFORE 5:30PM on the date specified at the beginning of the assessment, unless otherwise indicated in this document.

# Appendix B: Code Style and Commenting

### Code File Headers

At the start of EVERY file written in a C-style language (C, C++, C#, PHP, JavaScript, et al), the following block of comment is required, and must be completed with the appropriate information:

/\*\*

\* Assessment Title: Portfolio Part X

\* Cluster: Intermediate RIoT

\* Qualification: ICT50220 Diploma of Information Technology (Advanced Programming)

\* Name: YOUR NAME

\* Student ID: xxxxxxxxx

\* Year/Semester: 2024/S2

\*

\* YOUR SUMMARY OF PORTFOLIO ACTIVITY

\* GOES HERE

\*/

### Code Style (Naming Conventions)

For code written in a C-style language (C, C++, C#, PHP, JavaScript, et al), the following will be required…

#### Case (Upper/Lower/Mixed)

|  |  |  |
| --- | --- | --- |
| Case | Use For… | Example |
| Camel Case | Variables  Methods  Functions | ledState  toggleSwitch()  toggleLed() |
| Pascal Case | Class names | class Led() { …} |
| Snake case | n/a | bonus\_value |
| Shouty/Angry Snake Case | Constants | LED\_1 |

#### Length

|  |  |  |
| --- | --- | --- |
| Use | Requirements | Example |
| Variables, Constants, Methods, Functions, Class Names… | Minimum one word  NO abbreviations | ledState  LED\_1\_PIN  taxRate |

### Code Style (Formatting)

Code must be formatted consistently to facilitate ease of reading, debugging and collaboration. The following will be used as basic requirements:

|  |  |  |
| --- | --- | --- |
| Rule | Requirements | Example |
| Indenting | Multiples of 2 or 4 spaces  Do not mix indent sizes | if (switchState == LOW) {  setLedOn(LED\_1);  } |

### Code Documentation (All Comments)

Commenting of code will depend on the requirements of the assessment or project. The following are good guidelines for good code documentation for use in your work:

| Rule | Requirements | Example |
| --- | --- | --- |
| Value | Comments must add value to the code | // Calculate the power using Ohm’s law |
| Length | Lines should be less than 96 characters including prefixing symbols | // Determine taxation rate  /\*\*  \* Determine taxation rate  \*/ |

### Code Documentation (Doc Blocks)

Doc Blocks are used for commenting of methods, function, and classes. They are required to explain what the purpose is and how to use the item being described.

| Rule | Requirements | Example |
| --- | --- | --- |
| Doc Block | Provide summary details  Used for functions, methods and classes  Defines inputs and types  Defines output and types  Start with /\*\*  Each line starts with: \*  Last line: \*/  General Structure:   * First line after /\*\* is a one sentence short description * One blank line * Optional longer explanation with example usages * One blank line * Inputs * One blank line * Outputs   Inputs and Outputs are optional, so if the function/method does not contain these then the detail may be omitted. | /\*\*  \* LED On  \*  \* @input int ledPin  \*/ |

# Appendix C: Referencing

You will be expected to use **MyBib** (<https://mybib.com>) to collate and create your references.

We **DO NOT** expect a university style references with in-text citations.

We **DO** expect to see any references to use APA 6 or APA 7 style

We **DO** expect to see references added after answers to questions.

For example:

Imagine that you're working on a project locally and bump into an exception. You try to figure out the problem, but you're unable to find a solution. In that case, you might want to ask a colleague for help.  
  
Introducing Laravel Error Share - Blog. (2024, June 6). Flare. https://flareapp.io/blog/introducing-laravel-error-share

# Appendix D: Video Recordings

Video recording may be a required part of many of your submissions.

You may be required to record whilst demonstrating components of the assessment.

The following list is a set of basic requirements for video recordings:

* Recording MUST be done in LANDSCAPE mode only (image is WIDE not tall).
* The video MUST be recorded in a SINGLE take.
* No editing permitted.
* At the start you must:
  + Show your face
  + Verbally state the Cluster name
  + Verbally state the Assessment title
  + Verbally stating your name
  + Verbally state your student number
* When demonstrating you are expected to explain what you are showing.
* At the end of demonstrating you are expected to state your name once more.

# Appendix E: Portfolio Project Planning

Please refer to the following as a guide on how to plan the portfolio:

## Using GitHub to Manage a Project

Getting started with project planning on GitHub - The GitHub Blog

* <https://github.blog/developer-skills/github/getting-started-with-project-planning-on-github/>

GitHub Projects and Issues to manage a software project:

* <https://youtu.be/oPQgFxHcjAw?si=eUpDK8HlLNjFfHxs>

## GitHub Issues with Tasks

A good idea is to break down the big issues into a checklist of sub-tasks.

The best bit about this is that you may then convert them into individual GitHub issues. The following article describes how to do this.

Create Sub-Issues in GitHub Issues

* <https://dev.to/keracudmore/create-sub-issues-in-github-issues-409m>

# Appendix F: Git & GitHub

## Version Control Requirements

All code must be version controlled and placed into a **PRIVATE** repository on GitHub or a similar remote system. Access to this repository **must** be given to the assessors and lecturers via their **TAFE eMail** address(es).

You must create a new empty **private** repository on GitHub (or equivalent) for this work. The repository will use the naming structure:

XXX-SaaS-BED-Portfolio

Replace XXX with your initials.

If you are repeating the cluster, then you must use the following format for your repository name:

XXX-SaaS-BED-Portfolio-YY-SN

Where YY is the two digit year and N is the semester of study.

## Commit Messages

Commit messages should use the conventional commit style. This is outlined below:

|  |  |  |  |
| --- | --- | --- | --- |
| Type of commit | Prefix | Example | Notes |
| Start of project | init | init: Start of Project |  |
| Feature work | feat | feat: Add User create method |  |
| Feature with identifier | feat(...) | feat(user): Add create method | Preferred |
| Bug fix | fix(...) | fix(user): Fix issue #1234 |  |
| Documentation | docs(...) | docs(api): Update Scribe documentation | Preferred |

Other conventional commit message types are available, and you are directed to Conventional Commits (<https://www.conventionalcommits.org/en/v1.0.0/>) for more guidance and examples.

Note that conventional commits allow for multiple line comments. For example:

feat(user): Update browse API

- Add pagination to user API

- Add example use to API docs

close #1234

You may also link commits to your issues, and automatically close them by using the following keywords and syntax:

* fix #xxx
* close #xxx
* resolve #xxx

We suggest using close when completing a new (sub-)feature and resolve when completing a bug-fix.

Further useful details see below.

## Further Git/GitHub Resources

These resources will assist you in creating good commit messages, pull requests and provides other useful version control resources.

### Commit Message Related

Conventional Commits provide useful commit messages for anyone involved in a project.

* Conventional Commits Cheat sheet (github.com)
  + <https://gist.github.com/qoomon/5dfcdf8eec66a051ecd85625518cfd13>

Using commit messages to close issues is a great time saver.

* Closing Issues via Commit Messages - The GitHub Blog
  + <https://github.blog/news-insights/product-news/closing-issues-via-commit-messages/>

### Requests and Issues

Linking a pull request to an issue - GitHub Docs

* <https://docs.github.com/en/issues/tracking-your-work-with-issues/linking-a-pull-request-to-an-issue>

### General GitHub Issues

Quickstart for GitHub Issues - GitHub Docs

* <https://docs.github.com/en/issues/tracking-your-work-with-issues/quickstart>

GitHub Issue Templates

* <https://youtu.be/UPsCXqxxJUA?si=xj7Dom12MymZxn_S>