**Smart Nation Scholarship Technical Assessment**

**Krystal Lim Yun Shi**

**31 May 2022**

**ABOUT THE PROJECT/OBJECTIVES**

**PART I: SYSTEM SPECIFICATIONS**

**PART II: GETTING STARTED**

**PART III: APIs**

**PART IV: FEATURES/TECHNICAL REQUIREMENTS**

**PART V: DESIGN RATIONALE**

**PART VI: USAGE/DOCUMENTATION**

**CONCLUSION**

**ABOUT THE PROJECT/OBJECTIVES**

This prototyped web application, named, Krystal Lim Yun Shi (“KrystalLYS”), allows anyone to Create, Read, Update or Delete Ramen Data and/or Ramen Reviews. It was built on the framework stack consisting of Python 3, Flask, and SQLite Database. This web application allows any authenticated user to view a wide array of ramen types, flavors, and brands. It also allows one to view all reviews of specific ramen or give a review for a particular ramen type.

As this marks only the first prototype, I believe the objectives of this web application must be set out clearly for future iterations and sprints to be of high value and meaning. For instance, is this web application intended to be a universal “one-stop-shop” for ramen enthusiasts? If yes, then other features such as adding to the cart, the ability to purchase, credit card API integrations and analytical predictors should also be added. This will not only boost a user’s seamless experience but increase the utility of the website, eventually increasing profits if the intention was for ramen sales.

**PART I: SYSTEM SPECIFICATIONS**

* Python 3 (Version 3.10.4)
* Flask (Version 2.1.2)
* SQLite Database
* Virtual Environment in Python
* Bootstrap

**PART II: GETTING STARTED**

*Prerequisites:*

* *Requirements.txt*

|  |
| --- |
| pip install -r requirements.txt |

* Run “db-populate.py” to create the tables on your localhost if the KLDB file is not captured.

|  |
| --- |
| py db-populate.py |

* Run “app.py” to activate APIs and Views

|  |
| --- |
| py app.py |

**PART III: APIs**

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **API Endpoint** | **Description** |
| CRUD Ramen Data into Ramen Table | | |
| GET | /krystalapi | Getting all Ramen from Ramen\_Table |
| POST | /krystalapi | Creating a Ramen |
| GET | /krystalapi/<int:id> | Getting a Specific Ramen |
| PUT | /krystalapi/<int:id> | Updating a Specific Ramen based on the following fields:   * id * country * brand * type * package * rating * image |
| DELETE | /krystalapi/<int:id> | Deleting a Specific Ramen |
| Search Queries | | |
| GET | /krystalapi/search/<type> | Partial Search by Type/Flavour of Ramen |
| GET | /krystalapi/search/brand/<brand> | Partial Search by Brand of Ramen |
| GET | /krystalapi/country/<country> | Ramen filtered by Country |
| CRUD Reviews Data into Reviews Table | | |
| GET | /krystalapi/reviews | Getting all Review from Review\_Table |
| POST | /krystalapi/reviews | Creating a Review |
| GET | /krystalapi/reviews/<int:review\_id> | Getting a Specific Review |
| PUT | /krystalapi/reviews/<int:review\_id> | Updating a Specific Review based on the following fields:   * review\_id * brand * type * rating * note * ramen\_id * user\_id |
| DELETE | /krystalapi/reviews/<int:review\_id> | Deleting a Specific Review |
| GET | /krystalapi/reviews-byramen/<ramen\_id> | Get all reviews by ramen\_id |
| GET | /krystalapi/reviews-byuser/<user\_id> | Get all reviews by user\_id |
| CRUD User Data into User Table | | |
| POST | /krystalapi/login | Verifying, validating and logging an existing user in. |
| GET | /krystalapi/login | Getting all users |
| POST | /krystalapi/register | Creating a User |
| GET | /krystalapi/user/<user\_id> | Getting a Specific User |
| PUT | /krystalapi/user/<user\_id> | Updating a Specific User based on the following fields:   * user\_id * firstname * lastname * domicile * username * password |
| DELETE | /krystalapi/user/<user\_id> | Deleting a Specific User |
| Others | | |
| GET | /krystalapi/userdata | Get ramen data from the domicile of user's and ramen country |

\*Note: The base URL (“http://127.0.0.1:5000/”) for the endpoints has been omitted for brevity.

**PART IV:** **FEATURES/TECHNICAL REQUIREMENTS**

* All the columns provided in the sample dataset (“ramen-ratings.csv”) have been included in the database.
* All queries are supported by REST API endpoints and all endpoints can be accessed by software such as curl or Postman (refer to Part III - APIs).
* A front-end interface has also been created (refer to Part VI – Usage/Documentation).
* CRUD for all 3 Tables (Ramen\_Table, Review\_Table, User\_Table) in Krystal Lim Database (“KLDB”)

Diagram

Description automatically generated

* + User\_Table 🡪 Reviews\_Table (One to Many)
  + Ramen\_Table 🡪 Reviews\_Table (One to Many)
* Flash
* Toast Notifier

**PART V:** **DESIGN RATIONALE**

I chose to adopt a RESTful API format as they allow various clients including browser apps to communicate with my database and server. I have taken into account [security](https://stackoverflow.blog/2021/10/06/best-practices-for-authentication-and-authorization-for-rest-apis/), performance, consistency, and ease of use for API consumers. I have designed my API based on commonly accepted conventions as it is the universally understood and best practice method for any user to easily understand and adapt for use. A well-designed API should enable:

* Improved Developer Experience
* Intuitive, Hard to Misuse
* High Adoption
* Complete and Concise

I have utilized routes in this instance, with the CRUD methods (POST, GET, PUT, and DELETE) for all three tables I have created – Ramen\_Table, Reviews\_Table, and User\_Table in Krystal Lim Database (‘KLDB’).

Referring to Part III – APIs, in each instance, I have created the APIs in a uniform manner. I have also adopted local nesting on the endpoints. It is recognised as best practice to design endpoints that are structured in this manner where associated information is contained in the primary object.

An example is appended below.

|  |
| --- |
| POST /krystalapi/reviews  GET /krystalapi/reviews  PUT /krystalapi/reviews/<review\_id>  DELETE /krystalapi/reviews/<review\_id> |

I have also included error handling in the APIs to allow better comprehension of requests and queries between the client-server frameworks.

At the crux of it all, APIs allow for scalability with security. It is crucial that APIs are designed in a manner that is standardized and easily understood.

In addition, I have also looked into imbuing a Model-View-Controller (MVC) Architecture into the application as it would improve the efficacy of relating the user interface (i.e., views) to underlying data models and organizing the data flow. At the crux of it all, it adopts the concept of Separation of Concerns (“SOC”) which establishes a well-organized system where each segment fulfils a meaningful and robust role while maximizing its ability to adapt to change. However, for the current scale of this web application, it is not a necessity yet.

**PART VI: USAGE/DOCUMENTATION**

*User Authentication; Login and Authentication Fields powered by User\_Table*

Graphical user interface, application, Teams

Description automatically generated

*Test Login Data to Utilize:*

[

        2065635,

        "Krystal",

        "Lim",

        "SGP",

        "krystallimyunshi83",

        "krystallim83@"

    ]

*Index Page with All Ramen Selection (Card View):*

A picture containing text, food, dish

Description automatically generated

*Index Page with All Ramen Selection (List View):*

Graphical user interface, text, application, email

Description automatically generated

*Simple Form to Add a Ramen:*

Graphical user interface, text, application, email

Description automatically generated

*We can then see that the Ramen has been Added (“Brand ABSCC”):*

Graphical user interface, application, website

Description automatically generated

*View All Reviews:*

Graphical user interface, text, application

Description automatically generated

*Simple Form to Add a Review:*

Graphical user interface, text, application, email

Description automatically generated

*We can then see that the Review has been Added (“Brand ABSCC”):*

Graphical user interface, text, application

Description automatically generated

*Partial Search (by Type/Flavour):*

A picture containing text, food, dish

Description automatically generated

A picture containing food, dish

Description automatically generated

*Profile Page:*

Graphical user interface, text, application

Description automatically generated

**CONCLUSION**

All in all, it is important to note that the Front-End Documentation is briefly illustrated and visualized in Part VI – Usage/Documentations for a more comprehensive prototype (inclusive of front-end and back-end). The front-end can certainly be improved and effectuated given a slightly longer working duration. It also can be more robust if other frameworks or database of greater capacity is used. However, all routes and APIs have been architected and established at full scale, drawing reference from fulfilling all user stories and more. It is recommended for tests of this web application to be conducted via endpoints on software such as Postman. The extensive list of APIs has been listed in Part III – APIs.

Thank you.