A Framework to Automatically Generate Application Frontend Based on OpenAPI Specification

A React-based Implementation for Frontend Generation for Moderately Complex Applications

Group Members
200522F - HPR Randinu
200272L - IP Jayawickrama
200273P - WMID Jayawickrama

Supervisors
Internal - Dr. Buddhika Karunarathne
External - Dr. Srinath Perera

Declaration

We declare that this is our own work and this Thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or Institute of higher learning and to the best of our knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. We retain the right to use this content in whole or part in future works (such as articles or books).

Signature: [illegible] Date: 15/07/2025

Signature: [illegible] Date: 15/07/2025

Signature: [illegible] Date: 15/07/2025

The supervisor should certify the Thesis with the following declaration. The above candidate has carried out research for the BSc. Engineering Honours Thesis under my supervision. I confirm that the declaration made above by the student is true and correct.

Name of Supervisor: Dr. Buddhika Karunarathne

Signature: Date:

Dedication

This project is dedicated to our beloved University of Moratuwa. The continuous support, academic guidance, and inspiring environment provided by the university have played a vital role in making this project a reality. We are deeply grateful for the knowledge, encouragement, and opportunities offered to us throughout our academic journey. It is with sincere appreciation that we acknowledge the university's role in shaping our growth and success.

Acknowledgement

We would like to express our sincere gratitude to everyone who contributed to our project in any way. Without the help and support of the following individuals and organizations, we would not have been able to successfully carry out the project. Firstly, we would like to thank our supervisors, Dr. Buddhika Karunarathne and Dr. Srinath Perera for their guidance, encouragement,and valuable insights into the project. Your support was the backbone for the success of this project. We would also like to thank our batchmates for their support and contribution to our project. Their support greatly helped us to the success of our project. We would also like to thank our university, University of Moratuwa and our department, the Department of Computer Science and Engineering for providing support and resources for the project. The facilities provided by the university and the department greatly helped us to carry out the project successfully. Finally, we would like to thank our family and friends for their immense support and encouragement. With their help and support, we were successfully able to carry out the project. Once again, thank you everyone for your contribution, support and encouragement for the project.

Abstract

The rapid pace of web application development has underscored the need for automation in frontend generation, especially for developers with limited frontend experience. While backend API specifications define the structure and functionality of an application's data, the creation of consistent and functional frontends that interact with these APIs remains a manual and time-consuming process. This research introduces a framework that automates frontend generation by using the OpenAPI Specification for backend API definitions and user-configured page layouts. The framework generates React-based user interfaces, enabling developers to quickly produce responsive and customizable web applications without delving into frontend complexities. The approach reduces manual effort, accelerates development cycles, and offers flexibility in adapting to user requirements. The paper details the framework's methodology, demonstrates the generated frontend through a pet store application, and evaluates the framework using qualitative user feedback and performance metrics. The evaluation indicates the framework's efficacy in simplifying the generation of functional frontends.

Keywords: Frontend Code Generation, OpenAPI Specification, Automated Code Generation, API-Driven Development, Frontend Evaluation, System Usability Scale (SUS)

Contents

List of Figures

List of Tables

List of Appendices

1 Introduction

1.1 Background

The rise of APIs (Application Programming Interfaces) has transformed how modern applications are built. The OpenAPI Specification (OAS), formerly known as Swagger, has emerged as a leading standard for defining APIs. With OAS, developers can create machine-readable descriptions of the behavior, structure, and documentation of APIs. These descriptions can then be used to create a variety of artifacts, including documentation, client libraries, and server stubs. However, despite the widespread adoption of OAS, the process of creating a consistent and functional frontend that interacts with these APIs remains largely manual.

Frontend and backend developers typically collaborate during the development process. The frontend team develops the user interface (UI) that interfaces with the backend, while the backend team designs and implements the API. This often leads to discrepancies, inconsistencies, and additional workload, as frontend developers may need to repeatedly adapt their code to align with API changes. Even with tools like Swagger Codegen and OpenAPI Generator that help produce client-side SDKs, the generation of full UI components and layouts remains out of scope, creating a gap between API specifications and practical, usable interfaces.

At the same time, there is an increasing demand for automating frontend development due to the rising complexity of user interfaces and the pressure to deliver applications more quickly and consistently. Various research efforts have emerged to address this need. These range from template-based generators and model-driven engineering (MDE) approaches to advanced neural and multimodal systems that generate UIs from specifications or mockups. However, many of these solutions focus on idealized or highly controlled inputs and often lack adaptability for real-world development workflows, particularly for backend-focused developers with limited frontend experience.

This project seeks to address these challenges by introducing a framework that leverages the OAS to automatically generate the frontend for web applications. By using the OpenAPI Specification as the backend API specification, along with user-configured page layouts, the framework automates the creation of user interfaces using React as the frontend technology, enabling developers with limited frontend experience to quickly generate functional, responsive, and customizable web applications. This approach reduces manual effort, accelerates development cycles, and ensures flexibility in adapting to specific user requirements.

1.2 Problem Statement

Automating the generation of frontend code from API specifications has the potential to significantly accelerate web development workflows. However, current tools in this space often fall short of delivering truly usable and adaptable solutions. These tools either offer support on producing only boilerplate code or basic scaffolding to speed up the development process or require manual modifications that need to be done by the user in order to make the frontend fully functional. This undermines the core advantage of automation, reintroducing the inefficiencies it seeks to eliminate.

In addition to these shortcomings, most existing tools offer limited flexibility when it comes to customizing the look, feel, and structure of the generated UI. They often impose rigid templates or fixed workflows that do not adapt well to different application needs. Some tools also demand a deep understanding of the underlying technologies—such as templating engines, framework-specific build systems, or model transformation languages—which creates a steep learning curve for developers who are not already experienced in those areas. The setup and configuration processes of some tools can be both time-consuming and unintuitive, often requiring multiple steps, dependencies, or unfamiliar configurations.

These challenges highlight the pressing need for a more robust, customizable, and developer-friendly approach to frontend generation from API specifications.

1.3 Motivation

The motivation for this research stems from the increasing demand for efficient and consistent web application development processes in a rapidly evolving software landscape. Workflow optimization and a reduction in repetitive tasks become critical as businesses strive to produce high-quality apps more quickly.

The motivation of this project is to build a practical solution that bridges the gap between backend API definitions and fully functional frontends, addressing the shortcomings of current tools. With this framework, developers with limited frontend experience will have the ability to create simple and consistent frontends for their moderately complex applications. This aligns with the broader industry trend towards automation, where the goal is to minimize human intervention in repetitive tasks and focus on more complex, value-added activities.

1.4 Project Objectives

1.4.1 Framework Development

The primary objective of this project is to design and develop a framework that can automatically generate fully functional React-based frontend applications directly from OpenAPI specifications.

The specific research objectives are outlined as follows:

- **Automation of Frontend Generation:** Develop a robust and extensible system capable of generating complete, production-ready React-based frontends from OpenAPI-compliant API definitions.
- **Support for CRUD Operations:** Ensure that the generated UI supports the essential Create, Read, Update, and Delete operations corresponding to defined API endpoints.
- **Authentication and Authorization Integration:** Incorporate support for common access control mechanisms to allow secure user interactions with the generated frontend.
- **Customization Flexibility:** Enable developers to configure layout structure, visual styling, and component positioning through a simplified configuration model, thereby maintaining balance between automation and design control.

1.4.2 Evaluation of the Framework and Generating Code

The objective is to assess the effectiveness of the developed framework through a systematic evaluation process that examines both the framework itself and the frontend applications it generates. This includes measuring performance aspects such as responsiveness and application efficiency. Usability assessments are conducted through developer feedback, focusing on factors like overall quality, user experience, ease of use, interface behavior, and code readability. Together, these evaluations aim to validate whether the framework meets its goals of generating functional, maintainable, and user-friendly frontends with minimal manual effort.

2 Literature Review

2.1 Overview

Recently, there has been an increasing demand for automating frontend development due to the complexity of user interfaces and the requirement to develop rapidly and in a consistent manner. One of the most promising directions in this field is to leverage API specifications. In particular, research has been conducted on utilizing OpenAPI specification and other standard API specifications for automated code generation. Beyond leveraging API specifications, several strands of research and practice have emerged as well. These include template-based generators that transform specifications or mockups into code, model-driven engineering (MDE) approaches that use metamodels and transformation rules, and neural or multimodal AI systems that directly interpret visual or semantic prototypes. Together, these methods illustrate a movement toward not just automating code production, but aligning it with maintainability, UI/UX fidelity, and developer collaboration. This review explores the current landscape through two perspectives: code generation and the evaluation of the generated code.

2.2 Code Generation

Code generation in frontend development has evolved from template-based approaches to sophisticated model-driven and AI-enhanced techniques. Luburić et al. [1] presented a simple DSL-based solution that translates JSON definitions into AngularJS applications through Freemarker templates, enabling reusable, modular UI components with pagination and CRUD functionality. Koren and Klamma [2] proposed a two-stage model-driven approach that utilizes OpenAPI-to-IFML transformation, followed by mapping to HTML/JS, thereby facilitating cross-device responsive design through the application of Polymer components. Xiao et al. [3] pushed automation possibilities to new heights with Prototype2Code, an end-to-end pipeline that leverages graph neural networks and multi-head attention mechanisms to take in user interface prototypes and produce semantically meaningful, responsive HTML/CSS. In parallel, Arhandi et al. [4] concentrated on the development of Angular code through the utilization of REST API endpoints, while dynamically establishing smart as well as presentational components through an effective input mechanism.

More recently, researchers have explored multimodal and neural approaches to further enhance generation quality. Si et al. [5] benchmarked LLMs like GPT-4o for UI generation from webpage screenshots, revealing how visual input combined with textual augmentation can improve layout fidelity. Nikiforova et al. [6] utilized Draw.io diagrams and conversion rules for automatically generating AngularJS components with over 93% of visual similarity preserved from mockups. Rohma and Azurat [7] extended IFML transformations to React using Acceleo, supporting modular code output with TailwindCSS and GrapesJS. He et al. [8] advanced Pix2Code with better DSLs and encoder-decoder models for better BLEU and ROUGE scores. Vaziri et al. [9] provided a fresh insight by translating OpenAPI specs into FSM-based chatbots.

Alongside these academic efforts, Swagger Codegen [10] and OpenAPI Generator [11] provide mature tools that are able to translate OpenAPI specifications into type-safe, frontend-friendly client code components like SDKs and API clients for many frameworks. This features provides strong foundations for integration and enables rapid prototyping.

2.3 Generated Code Evaluation

Evaluating the quality of generated frontend code requires both quantitative metrics and qualitative insights. Abrahamsson [12] proposed a multi-criteria model to assess frameworks like React and Angular based on ease of use, community support, performance, and documentation. Dobbala [13] emphasized the role of testing frameworks like Jest, Cypress, and React Testing Library that provide automated verification of layout correctness, user input, and regression, especially in CI/CD pipelines. Knowles et al. [14] adopted a user-centered approach to evaluation through the application of usability testing and pilot simulations, emphasizing the significance of intuitive interaction and task success in validating systems developed. These structured methodologies ensure that auto-generated interfaces meet end-user expectations and support reliable functionality across devices.

Evaluation studies also emphasize practical testing and modular design. Arhandi et al. [4] validated their Angular generator through black-box testing, confirming code correctness while identifying error-handling limitations. Rohma and Azurat [7] evaluated their IFML-to-React tool against SPLE metrics, with high modularity but partial IFML coverage. Fu and Jung [15] analyzed government web platforms through heuristic analysis, demonstrating ongoing usability gaps reflecting the code quality underneath. Fadhlillah et al. [16] applied SUS-based assessments to UI variants generated via delta modeling, finding high usability despite layout control constraints. Broader reviews like those by Raj et al. [17] and Akiki et al. [18] highlight iterative testing, flexibility, and user feedback as crucial in the process of tuning generated UIs so both code functionality and experience design receive full attention in today's auto-generation processes.

3 Methodology

This methodology introduces a framework for automatically generating frontend interfaces by transforming standardized API definitions and user-specified page configurations into responsive, React-based applications. The framework leverages the OpenAPI Specification along with XML-driven layout definitions to automate the generation of frontend components. Additionally, it supports extensive customization, enabling practical adaptation for a wide range of real-world application scenarios.

3.1 Overview of the Framework

The framework generates React-based user interfaces by processing three inputs: the OpenAPI Specification, a page configuration file, and user customization files. This process is illustrated through high-level (Figure 1) and detailed architecture (Figure 2) diagrams, which outline the transformation of inputs into the final output. Further details on the inputs and workflow are provided in the following sections.

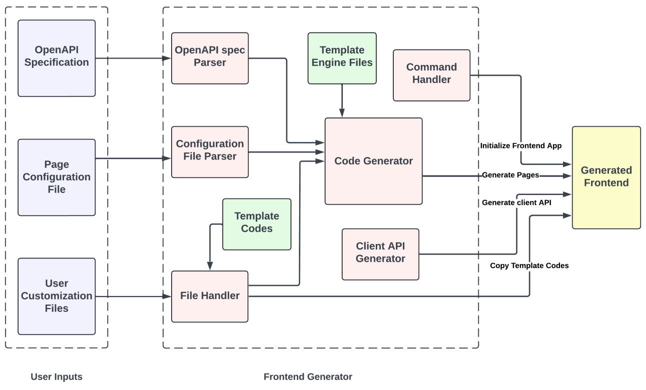


Figure 1: High-level architecture diagram

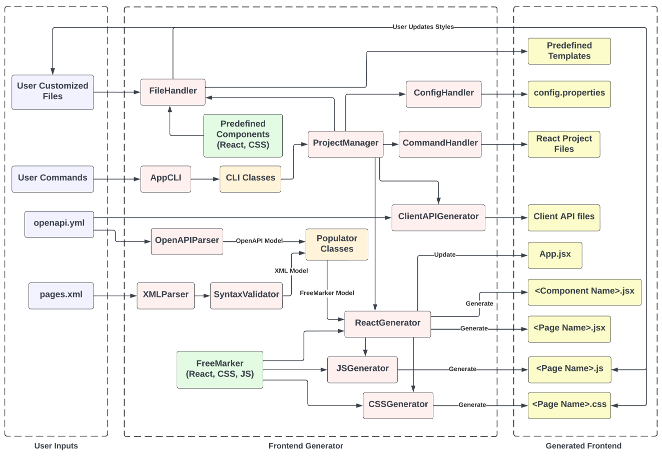


Figure 2: Detailed architecture diagram

Functional Overview of Architecture Modules

1. **AppCLI**
   Serves as the main entry point for the CLI.

2. **ProjectManager**
   Acts as a central manager, coordinating the entire project lifecycle between handlers, parsers, generators, and file operations to streamline the framework's functionality.

3. **CommandHandler**
   Handles execution of external commands, such as creating a React app, running the project, and installing NPM packages.

4. **ConfigHandler**
   Manages the config.properties file, allowing reading, writing, and updating project configuration properties.

5. **FileHandler**
   Provides utilities for file and directory operations, including creating directories, copying files, and handling templates during project setup.

6. **OpenAPIParser**
   Parses and validates the OpenAPI Specification, extracting operations, parameters, schemas, responses, and other API-related data.

7. **XMLParser**
   Parses the pages.xml file to extract page definitions and configurations.

8. **SyntaxValidator**
   Validates pages.xml file syntax.

9. **ClientAPIGenerator**
   Generates client API code based on the OpenAPI Specification using OpenAPI Generator Tools.

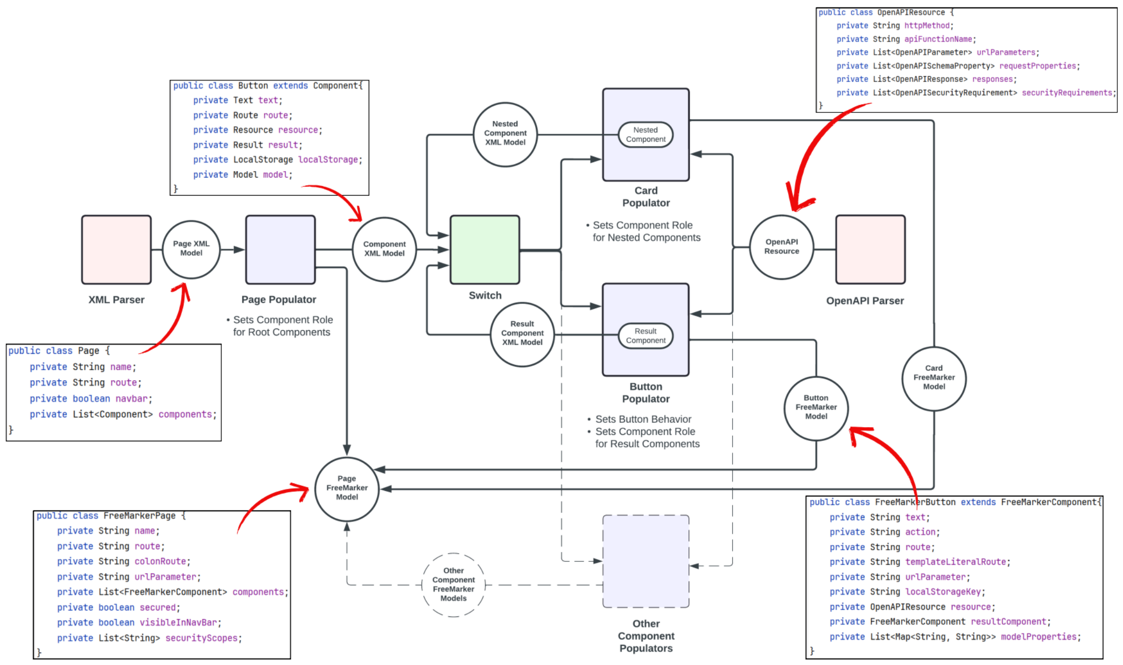
10. **CSSGenerator**
    Creates CSS template files for adding custom styles to React pages and components.

11. **JSGenerator**
    Generates JavaScript styling template files for custom page-level styling.

12. **ReactGenerator**
    Handles the generation of React components and pages by processing input data from OpenAPI and XML configuration files.

13. **Populator classes**
    Creates FreeMarker models using XML models and OpenAPI models to send to the template engine.

Figure 3 illustrates how the Populator classes operate within the framework. Each component type (e.g., Button, Card, Table) has its own dedicated populator, along with a central Page Populator. The XML Page Model defines page-level configurations and includes a list of components. Each component in the XML model is processed through a Switch module, which dynamically routes it to the appropriate populator based on its type. The populator then constructs the FreeMarker model for that component by assigning its role and behavior. If the component interacts with OpenAPI endpoints, the relevant API configurations are also included. The Switch operates recursively to support nested components. After all components are processed, the Page FreeMarker Model is composed with page-level settings and component models, which is finally passed to the FreeMarker engine for code generation.


```
public class Button extends Component{
    private Text text;
    private Route route;
    private Resource resource;
    private Result result;
    private LocalStorage localStorage;
    private Model model;
}
```

```
public class OpenAPIResource {
    private String httpMethod;
    private String apiFunctionName;
    private List<OpenAPIParameter> urlParameters;
    private List<OpenAPISchemaProperty> requestProperties;
    private List<OpenAPIResponse> responses;
    private List<OpenAPISecurityRequirement> securityRequirements;
}
```

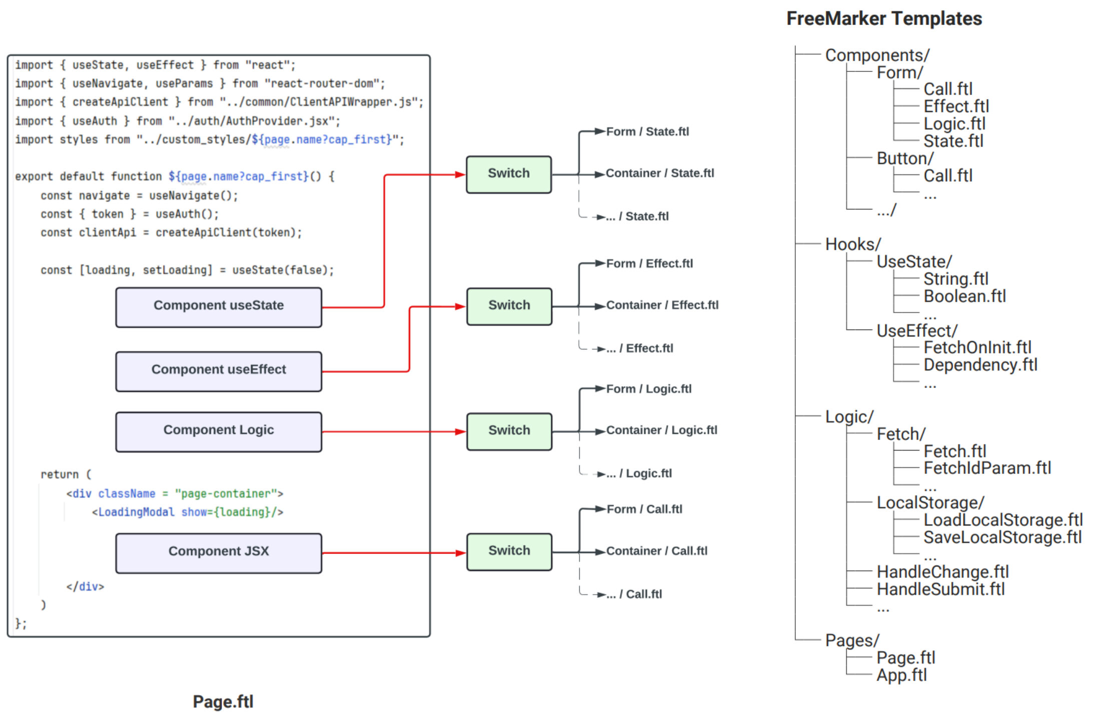
```
public class Page {
    private String name;
    private String route;
    private boolean navbar;
    private List<Component> components;
}
```

```
public class FreeMarkerPage {
    private String name;
    private String route;
    private String colonRoute;
    private String urlParameter;
    private List<FreeMarkerComponent> components;
    private boolean secured;
    private boolean visibleInNavBar;
    private List<String> securityScopes;
}
```

```
public class FreeMarkerButton extends FreeMarkerComponent{
    private String text;
    private String action;
    private String route;
    private String templateLiteralRoute;
    private String urlParameter;
    private String localStorageKey;
    private OpenAPIResource resource;
    private FreeMarkerComponent resultComponent;
    private List<Map<String, String>> modelProperties;
}
```

Figure 3: Populator structure and workflow

Figure 4 explains the inner workings of the FreeMarker template engine used in the framework. The engine follows a modular and hierarchical structure where every component has dedicated template files that define its JSX structure, state hooks (”useState”), effect hooks (”useEffect”), and logic (such as API calls or local storage). In addition to component-specific templates, it includes templates for reusable hooks and logic functions (e.g., Fetch, HandleChange, LocalStorage). The ”Page.ftl” file is the entry template for rendering a React page. It iterates over the list of components in four stages: first generating their state variables, then useEffects, followed by logic, and finally the JSX layout. The appropriate template file is selected during each stage based on the component’s assigned role and behavior.



```
import { useState, useEffect } from "react";
import { useNavigate, useParams } from "react-router-dom";
import { createApiClient } from "../common/ClientAPIWrapper.js";
import { useAuth } from "../auth/AuthProvider.jsx";
import styles from "../custom_styles/${page.name?cap_first}";

export default function ${page.name?cap_first}() {
    const navigate = useNavigate();
    const { token } = useAuth();
    const clientApi = createApiClient(token);

    const [loading, setLoading] = useState(false);
                Component useState

                Component useEffect

                Component Logic

    return (
        <div className = "page-container">
            <LoadingModal show={loading}/>

                Component JSX

        </div>
    )
};
```

```
FreeMarker Templates
├── Components/
│   ├── Form/
│   │   ├── Call.ftl
│   │   ├── Effect.ftl
│   │   ├── Logic.ftl
│   │   └── State.ftl
│   ├── Button/
│   │   ├── Call.ftl
│   │   └── ...
│   └── .../
├── Hooks/
│   ├── UseState/
│   │   ├── String.ftl
│   │   ├── Boolean.ftl
│   │   └── ...
│   └── UseEffect/
│       ├── FetchOnInit.ftl
│       ├── Dependency.ftl
│       └── ...
├── Logic/
│   ├── Fetch/
│   │   ├── Fetch.ftl
│   │   ├── FetchIdParam.ftl
│   │   └── ...
│   ├── LocalStorage/
│   │   ├── LoadLocalStorage.ftl
│   │   ├── SaveLocalStorage.ftl
│   │   └── ...
│   ├── HandleChange.ftl
│   ├── HandleSubmit.ftl
│   └── ...
└── Pages/
    ├── Page.ftl
    └── App.ftl
```

Figure 4: Modular FreeMarker template system

3.2 Inputs

3.2.1 OpenAPI Specification (OAS)

The OpenAPI Specification (OAS) defines backend API endpoints, operations, and data schemas in a structured, machine-readable format, typically written in YAML or JSON.

i. Custom Extensions:
   The framework utilizes the "x-displayName" extension as a custom addition to the standard OpenAPI specification. This extension provides user-friendly names for parameters and properties, enhancing the readability of the frontend UI.

```
parameters:
  - name: id
    in: path
    required: true
    schema:
      type: string
    x-displayName: ID
```

```
properties:
  itemId:
    type: string
    x-displayName: Item ID
```

ii. Input Type Mapping:
The framework intelligently maps OpenAPI schema formats to appropriate HTML input types in the generated frontend UI. By inspecting the “format” property of each schema field, it automatically renders the correct input type. This ensures both semantic accuracy and enhanced user experience during data entry.

```
emailAddress:
  type: string
  format: email
```

```
birthDate:
  type: string
  format: date
```

iii. Authentication and User Roles:
When implementing authentication, user roles (e.g., admin, user) must be defined under scopes in the security schemes section. These roles are then associated with specific API endpoints to enforce Role-Based Access Control (RBAC), allowing the frontend to restrict access to resources or actions based on the user’s role.

```
securitySchemes:
  googleOAuth:
    type: oauth2
    flows:
      authorizationCode:
        authorizationUrl: https://accounts.google.com/o/oauth2/auth
        tokenUrl: https://oauth2.googleapis.com/token
        scopes:
          user: "Registered user access"
          admin: "Administrator access"
```

When defining an endpoint that requires a specific role, include the “security” section to specify which role is needed for access.

```
post:
  summary: Create a new user
  operationId: createUser
  security:
    - googleOAuth: [ admin ]
```

3.2.2 Page Configuration File

The pages.xml file is an XML-based configuration that represents the frontend layout and interaction flow. It defines how pages, components, and user interactions are structured in a declarative manner.

i. Overall Structure:
At its core, the file is structured hierarchically with nested components that define UI layout and behavior. Below is the abstracted structure.

```
<pages>
  <page name="..." route="..." navbar="true|false">
    <!-- Root Component -->
    <component type="..." id="...">
      <!-- Result Component -->
      <result>
        <component type="..." id="..."/>
      </result>
      <!-- Nested Component -->
      <component type="..." id="..."/>
    </component>
  </page>
</pages>
```

ii. Component Types:
The framework defines two major categories of components.

(a) Root Components:
These components are defined directly under a <page> or a <component>. Supported Root Components include Button, Container, Form, HeroSection and SearchBar

(b) Result Components:
These are displayed as a response or result of some action taken by a root component. Supported Result Components include Alert, Card, CardSection and Table.

iii. Component Properties:
Each component may include specific properties or tags that define how it behaves or renders. (Table 1)

Table 1: Supported Component Properties and Their Purposes

Property	**Purpose**
Assign	Maps schema fields to UI elements.
Model	Allows defining a custom data object by assigning keys and values.
Resource	Defines the OpenAPI endpoint used by the component.
Route	Specifies the frontend navigation path.
LocalStorage	Configures save/load/remove actions for browser local storage.
Text	Defines textual content.
Submit	Defines the label of the submit button.
Image	Specifies an image URL to be displayed in components.

iv. Component Reference Table: (Table 2)

Table 2: Supported Properties, Result Components, and Nesting Capabilities of UI Components

Component/ Description	**Supported Properties**	**Supported Result Components**	**Supported Nested Components**
Alert (Displays success/failure messages)	–	–	–
Button (Triggers actions like routing, API calls, or local storage)	Text, Route, Resource, LocalStorage, Model	Alert	–
Card (Visual block with image, title, description, and highlight)	Assign	–	Button
CardSection (Collection of cards showing multiple items)	Assign, Route	–	–
Container (Fetches data on page load using useEffect)	Resource, LocalStorage, Text	Card, CardSection, Table	–
Form (Schema-bound form that submits data to an API)	Resource, Text, Submit	Alert	–
HeroSection (Static UI with image and highlighted text)	Text, Image	–	Button
SearchBar (Search input linked to an API endpoint)	Resource	CardSection	–
Table (Tabular view of data)	Assign	–	Button

3.2.3 User Customization Files

The framework allows users to customize the generated frontend through three types of styling:

i. Component Styling:
This styling method uses CSS files to modify the default appearance of predefined React components, such as Button, Card, and Table, applying the styles globally across all instances of these components within the application.

ii. Page-Specific Styling:
This styling method offers finer control by using JavaScript files for each page, enabling users to customize the styling of page-specific components via React's style prop. These files define styles that are applied to components on a per-page basis.

iii. Layout and Positioning Styling:
This styling focuses on adjusting the positioning and layout of components on the page. Each component is enclosed in a container <div> with CSS files controlling the layout and positioning of these containers, allowing users to customize the page design according to their preferences.

3.3 Workflow of the Framework

3.3.1 Initialization

In this stage, the framework constructs the project structure by creating the necessary directories and files for the frontend. Predefined React components are copied to the build folder, the React application is initialized, and the required NPM dependencies are installed.

3.3.2 Frontend and Client API Generation

In this stage, the framework parses the openapi.yaml file to create OpenAPI models defining backend resources, and the pages.xml file to create XML models describing the frontend layout and components. These models are combined to produce FreeMarker models, which are then processed by FreeMarker templates to generate React components and pages. The templates are organized into sub-templates, each responsible for specific parts of the page, including hooks, logic, and JSX syntax. Additionally, templates for page-specific styling are generated. Finally, the client API is generated using OpenAPI Generator Tools based on the OpenAPI specification.

3.3.3 Apply Custom User Styles

This stage allows users to apply their own styles to modify the appearance of the generated frontend. Users can provide custom CSS and JavaScript styling files, which are then copied into the build folder, giving them the flexibility to personalize the frontend design as per their requirements.

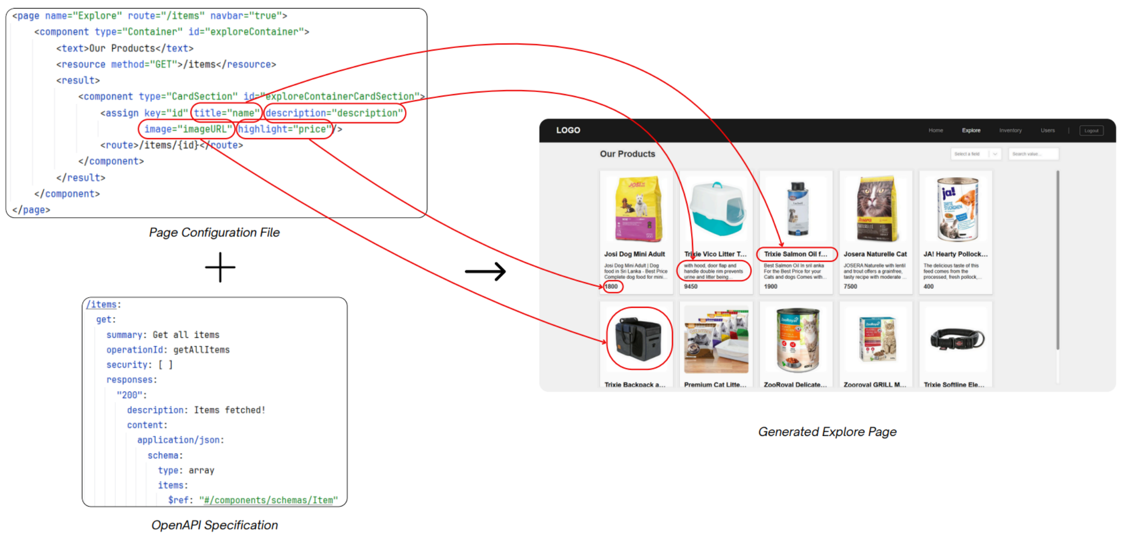
3.3.4 Running the Application

In this final stage, the framework compiles the generated React application, ensuring all components and pages are correctly built and optimized. Once the compilation is complete, a local development server is started, allowing users to preview and interact with the generated frontend in real time. This stage offers an opportunity to test the application, make modifications, and ensure functionality.

4 Results

4.1 Generated Components

Figure 5 illustrates the automatically generated Explore Page. The page configuration file defines a “Container” component associated with the “/items” GET endpoint. A “CardSection” is used as the result component to display multiple product cards. Each card is populated dynamically using key-value assignments (e.g., “title”, “description”, “image”, “highlight”) that are mapped from the OpenAPI response. This configuration enables rapid rendering of product listings in a card-based layout.



```xml
<page name="Explore" route="/items" navbar="true">
    <component type="Container" id="exploreContainer">
        <text>Our Products</text>
        <resource method="GET">/items</resource>
        <result>
            <component type="CardSection" id="exploreContainerCardSection">
                <assign key="id" title="name" description="description"
                        image="imageURL" highlight="price"/>
                <route>/items/{id}</route>
            </component>
        </result>
    </component>
</page>
```

```yaml
/items:
  get:
    summary: Get all items
    operationId: getAllItems
    security: [ ]
    responses:
      "200":
        description: Items fetched!
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: "#/components/schemas/Item"
```

Our Products

Josi Dog Mini Adult
Josi Dog Mini Adult | Dog food in Sri Lanka - Best Price Complete dog food for mini...
1800

Trixie Vico Litter T...
with hood, door flap and handle double rim prevents urine and litter being...
9450

Trixie Salmon Oil f...
Best Salmon Oil In sril anka For the Best Price for your Cats and dogs Comes with...
1900

Josera Naturelle Cat
JOSERA Naturelle with lentil and trout offers a grainfree, tasty recipe with moderate ...
7500

JA! Hearty Pollock...
The delicious taste of this feed comes from the processed, fresh pollock,...
400

Trixie Backpack a...

Premium Cat Litte...

ZooRoyal Delicate...

Zooroyal GRILL M...

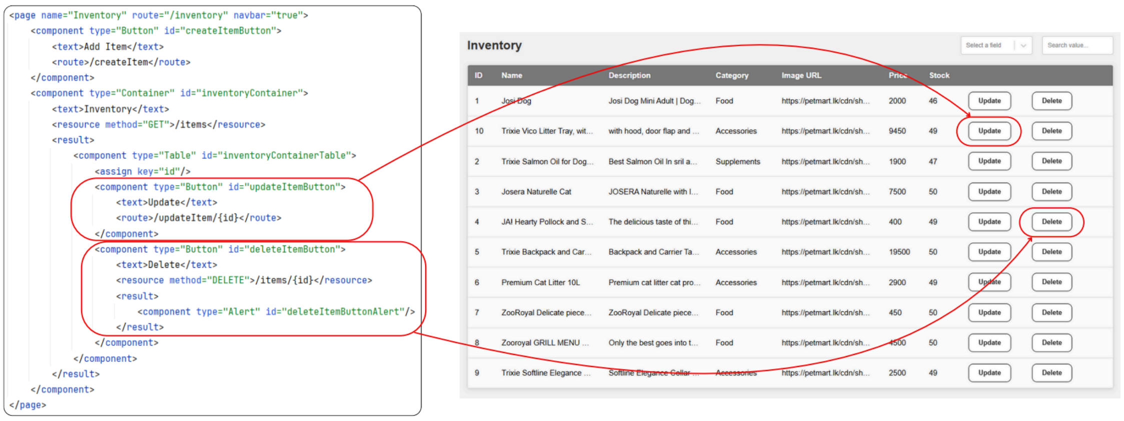
Trixie Softline Ele...

Figure 5: Products CardSection

Figure 6 shows the Inventory Page, which demonstrates a table-based layout generated using the “Table” component. The table rows include nested “Button” components to perform update and delete actions. The delete button is also linked with a “DELETE” request and triggers an “Alert” component upon execution. This demonstrates how nested components can facilitate interactive CRUD operations in a declarative manner.

Figure 7 presents the generated Pet Details Page using the “Card” component. It includes a nested “Button” labeled “Add to Cart” that triggers a POST request to the “/cart” endpoint, with a request payload constructed using the “model” property. On success, an “Alert” component is shown to provide user feedback. This showcases how dynamic interaction and backend integration can be achieved through simple XML configurations.

Figure 8 displays a generated form based on OpenAPI input schemas. Various field formats such as “email”, “password”, “tel”, “color”, and “date-time” are mapped to appropriate HTML input types. These mappings enhance user experience by ensuring proper input types and automatic validation, based entirely on OpenAPI field metadata like “format” and “x-displayName”.



```
<page name="Inventory" route="/inventory" navbar="true">
    <component type="Button" id="createItemButton">
        <text>Add Item</text>
        <route>/createItem</route>
    </component>
    <component type="Container" id="inventoryContainer">
        <text>Inventory</text>
        <resource method="GET">/items</resource>
        <result>
            <component type="Table" id="inventoryContainerTable">
                <assign key="id"/>
                <component type="Button" id="updateItemButton">
                    <text>Update</text>
                    <route>/updateItem/{id}</route>
                </component>
                <component type="Button" id="deleteItemButton">
                    <text>Delete</text>
                    <resource method="DELETE">/items/{id}</resource>
                    <result>
                        <component type="Alert" id="deleteItemButtonAlert"/>
                    </result>
                </component>
            </component>
        </result>
    </component>
</page>
```

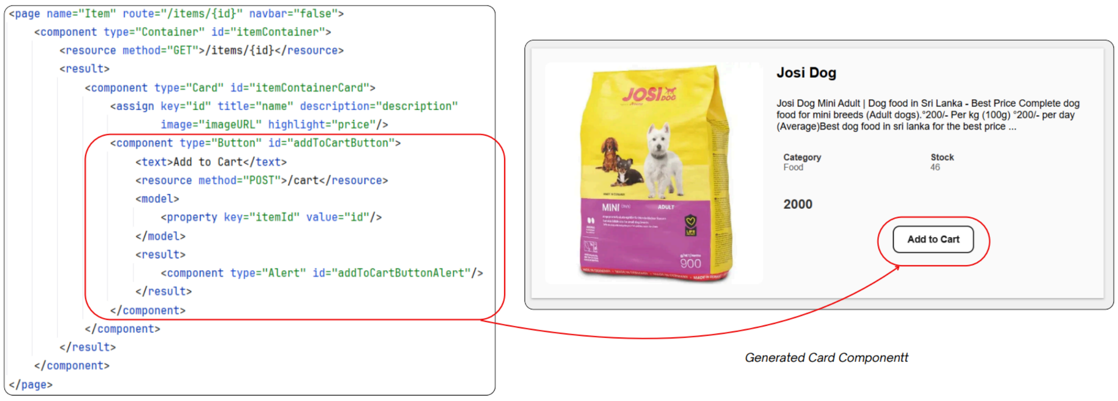
Page Configuration File

Inventory

ID	Name	Description	Category	Image URL	Price	Stock		
1	Josi Dog	Josi Dog Mini Adult | Dog...	Food	https://petmart.lk/cdn/sh...	2000	46	Update	Delete
10	Trixie Vico Litter Tray, wit...	with hood, door flap and ...	Accessories	https://petmart.lk/cdn/sh...	9450	49	Update	Delete
2	Trixie Salmon Oil for Dog...	Best Salmon Oil In sril a...	Supplements	https://petmart.lk/cdn/sh...	1900	47	Update	Delete
3	Josera Naturelle Cat	JOSERA Naturelle with I...	Food	https://petmart.lk/cdn/sh...	7500	50	Update	Delete
4	JA! Hearty Pollock and S...	The delicious taste of thi...	Food	https://petmart.lk/cdn/sh...	400	49	Update	Delete
5	Trixie Backpack and Car...	Backpack and Carrier Ta...	Accessories	https://petmart.lk/cdn/sh...	19500	50	Update	Delete
6	Premium Cat Litter 10L	Premium cat litter cat pro...	Accessories	https://petmart.lk/cdn/sh...	2900	49	Update	Delete
7	ZooRoyal Delicate piece...	ZooRoyal Delicate piece...	Food	https://petmart.lk/cdn/sh...	450	50	Update	Delete
8	Zooroyal GRILL MENU ...	Only the best goes into t...	Food	https://petmart.lk/cdn/sh...	4500	50	Update	Delete
9	Trixie Softline Elegance ...	Softline Elegance Collar ...	Accessories	https://petmart.lk/cdn/sh...	2500	49	Update	Delete

Generated Table with nested Components

Figure 6: Inventory Admin Table



```
<page name="Item" route="/items/{id}" navbar="false">
    <component type="Container" id="itemContainer">
        <resource method="GET">/items/{id}</resource>
        <result>
            <component type="Card" id="itemContainerCard">
                <assign key="id" title="name" description="description"
                        image="imageURL" highlight="price"/>
                <component type="Button" id="addToCartButton">
                    <text>Add to Cart</text>
                    <resource method="POST">/cart</resource>
                    <model>
                        <property key="itemId" value="id"/>
                    </model>
                    <result>
                        <component type="Alert" id="addToCartButtonAlert"/>
                    </result>
                </component>
            </component>
        </result>
    </component>
</page>
```

Page Configuration File

Josi Dog

Josi Dog Mini Adult | Dog food in Sri Lanka - Best Price Complete dog food for mini breeds (Adult dogs).°200/- Per kg (100g) °200/- per day (Average)Best dog food in sri lanka for the best price ...

Category
Food

Stock
46

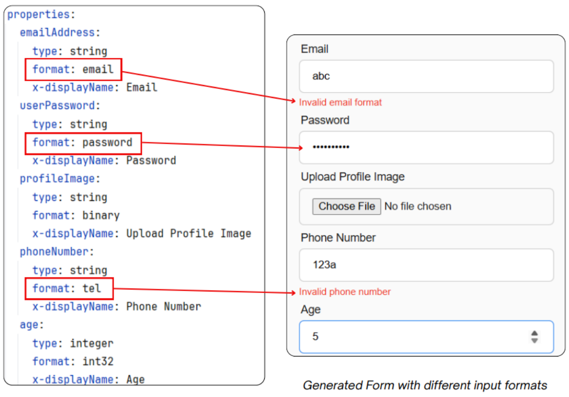
2000

Generated Card Componentt

Figure 7: Product Card

4.2 Generated Frontend

Figure 9 showcases several frontend pages generated for a PetStore application using the proposed framework. The Home Page features a hero section designed to engage users with a welcoming visual and a call-to-action button. The Explore Page presents a card-based layout listing available pet products with images, descriptions, and prices, dynamically populated from the backend. The Product Details Page displays detailed information for a selected item along with an "Add to Cart" button, facilitating user interaction. The Inventory Page provides a table-based interface for managing product listings, supporting update and delete operations through nested interactive buttons. These pages demonstrate the framework's ability to generate functional, data-driven, and visually consistent user interfaces directly from configuration and API definitions.

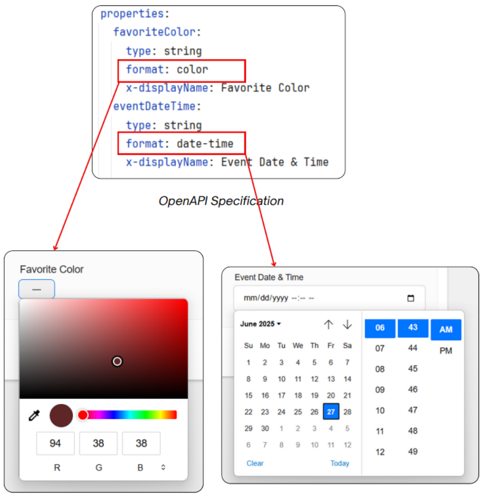


```
properties:
  emailAddress:
    type: string
    format: email
    x-displayName: Email
  userPassword:
    type: string
    format: password
    x-displayName: Password
  profileImage:
    type: string
    format: binary
    x-displayName: Upload Profile Image
  phoneNumber:
    type: string
    format: tel
    x-displayName: Phone Number
  age:
    type: integer
    format: int32
    x-displayName: Age
```

Email: abc
Invalid email format
Password: ••••••••••
Upload Profile Image: Choose File No file chosen
Phone Number: 123a
Invalid phone number
Age: 5

OpenAPI Specification

Generated Form with different input formats



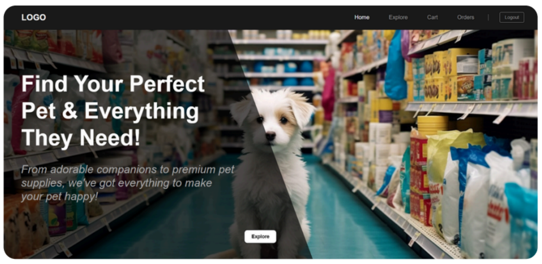
```
properties:
  favoriteColor:
    type: string
    format: color
    x-displayName: Favorite Color
  eventDateTime:
    type: string
    format: date-time
    x-displayName: Event Date & Time
```

Favorite Color: R 94, G 38, B 38
Event Date & Time: mm/dd/yyyy --:-- --

OpenAPI Specification

Generated Form with different input formats

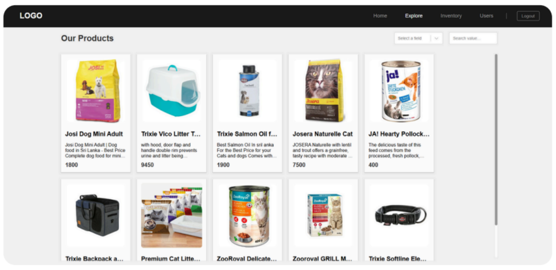
Figure 8: Input Formats



Find Your Perfect Pet & Everything They Need!

From adorable companions to premium pet supplies, we've got everything to make your pet happy!

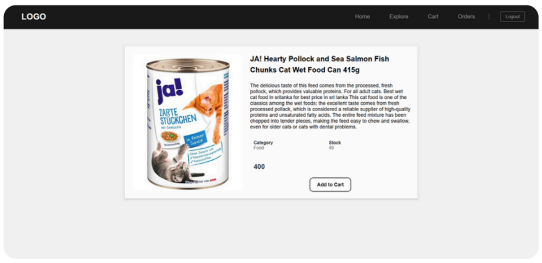
Home Page



Our Products

Josi Dog Mini Adult — 1800
Trixie Vico Litter T... — 9450
Trixie Salmon Oil f... — 1900
Josera Naturelle Cat — 7500
JA! Hearty Pollock... — 400
Trixie Backpack a...
Premium Cat Litte...
ZooRoyal Delicate...
Zooroyal GRILL M...
Trixie Softline Ele...

Explore Page



JA! Hearty Pollock and Sea Salmon Fish Chunks Cat Wet Food Can 415g

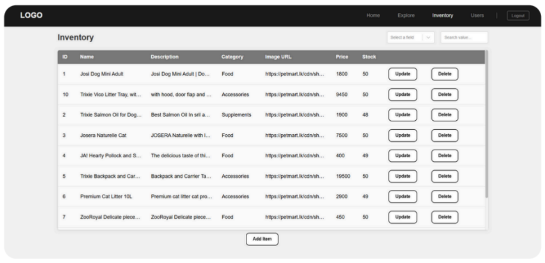
The delicious taste of this feed comes from the processed, fresh pollock, which provides valuable proteins. For all adult cats. Best wet cat food in srilanka for best price in sri lanka This cat food is one of the classics among the wet foods: the excellent taste comes from fresh processed pollack, which is considered a reliable supplier of high-quality proteins and unsaturated fatty acids. The entire feed mixture has been chopped into tender pieces, making the feed easy to chew and swallow, even for older cats or cats with dental problems.

Category: Food
Stock: 49

400

Add to Cart

Product Details Page



Inventory

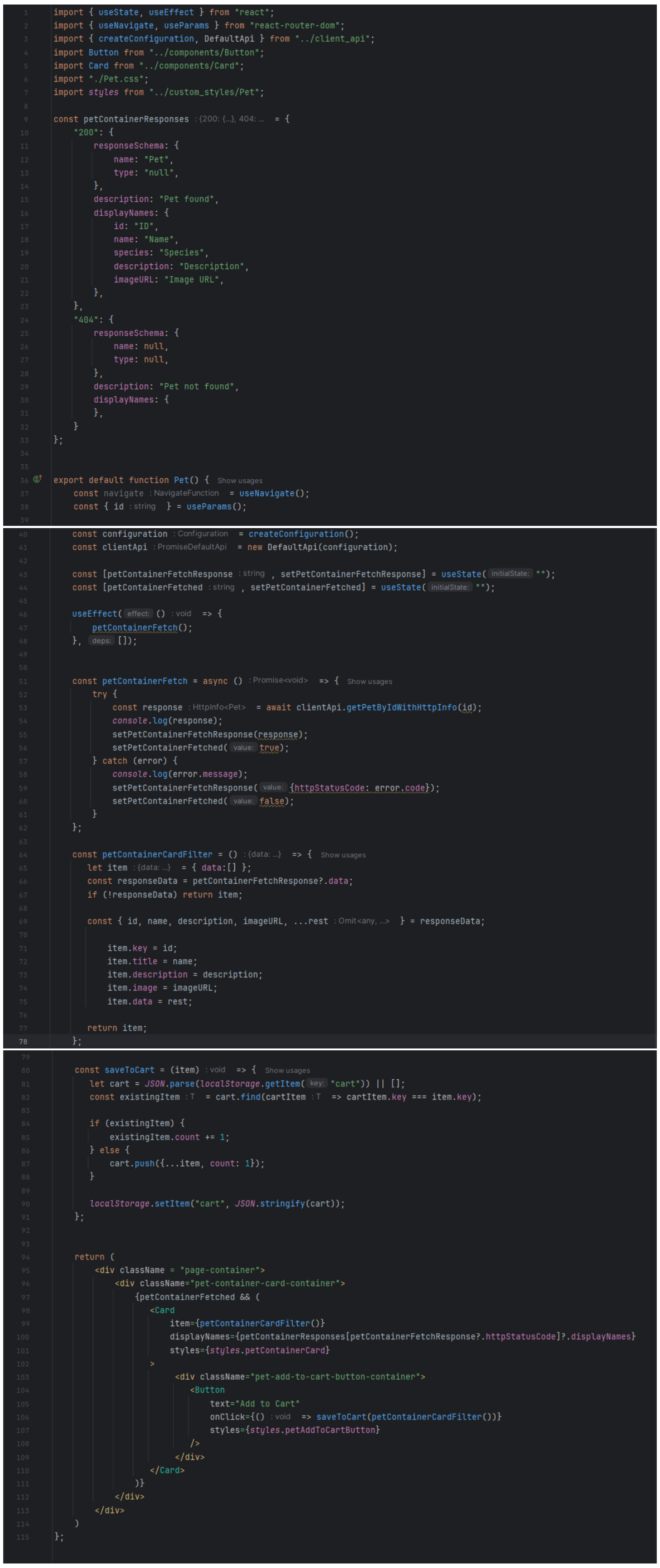
ID	Name	Description	Category	Image URL	Price	Stock		
1	Josi Dog Mini Adult	Josi Dog Mini Adult | Do...	Food	https://petmart.lk/cdn/sh...	1800	50	Update	Delete
10	Trixie Vico Litter Tray, wit...	with hood, door flap and ...	Accessories	https://petmart.lk/cdn/sh...	9450	50	Update	Delete
2	Trixie Salmon Oil for Dog...	Best Salmon Oil In sril a...	Supplements	https://petmart.lk/cdn/sh...	1900	48	Update	Delete
3	Josera Naturelle Cat	JOSERA Naturelle with l...	Food	https://petmart.lk/cdn/sh...	7500	50	Update	Delete
4	JA! Hearty Pollock and S...	The delicious taste of thi...	Food	https://petmart.lk/cdn/sh...	400	49	Update	Delete
5	Trixie Backpack and Car...	Backpack and Carrier Ta...	Accessories	https://petmart.lk/cdn/sh...	19500	50	Update	Delete
6	Premium Cat Litter 10L	Premium cat litter cat pro...	Accessories	https://petmart.lk/cdn/sh...	2900	49	Update	Delete
7	ZooRoyal Delicate piece...	ZooRoyal Delicate piece...	Food	https://petmart.lk/cdn/sh...	450	50	Update	Delete

Inventory Page

Figure 9: PetStore Application

4.3 Generated Code

Figure 10 shows a sample of the React code automatically generated by the framework. The code is clean, human-readable, and well-structured, adhering to modern coding best practices. It follows consistent formatting, modular component usage, and logical separation of concerns, making it easy for developers to understand, customize, and maintain.



```javascript
import { useState, useEffect } from "react";
import { useNavigate, useParams } from "react-router-dom";
import { createConfiguration, DefaultApi } from "../client_api";
import Button from "../components/Button";
import Card from "../components/Card";
import "./Pet.css";
import styles from "../custom_styles/Pet";

const petContainerResponses :{200: {...}, 404: ...  = {
    "200": {
        responseSchema: {
            name: "Pet",
            type: "null",
        },
        description: "Pet found",
        displayNames: {
            id: "ID",
            name: "Name",
            species: "Species",
            description: "Description",
            imageURL: "Image URL",
        },
    },
    "404": {
        responseSchema: {
            name: null,
            type: null,
        },
        description: "Pet not found",
        displayNames: {
        },
    }
};


export default function Pet() {  Show usages
    const navigate :NavigateFunction  = useNavigate();
    const { id :string  } = useParams();

    const configuration :Configuration  = createConfiguration();
    const clientApi :PromiseDefaultApi  = new DefaultApi(configuration);

    const [petContainerFetchResponse :string , setPetContainerFetchResponse] = useState( initialState: "");
    const [petContainerFetched :string , setPetContainerFetched] = useState( initialState: "");

    useEffect( effect: () :void  => {
        petContainerFetch();
    },  deps: []);


    const petContainerFetch = async () :Promise<void>  => {  Show usages
        try {
            const response :HttpInfo<Pet>  = await clientApi.getPetByIdWithHttpInfo(id);
            console.log(response);
            setPetContainerFetchResponse(response);
            setPetContainerFetched( value: true);
        } catch (error) {
            console.log(error.message);
            setPetContainerFetchResponse( value: {httpStatusCode: error.code});
            setPetContainerFetched( value: false);
        }
    };

    const petContainerCardFilter = () :{data: ...}  => {  Show usages
       let item :{data: ...}  = { data:[] };
       const responseData = petContainerFetchResponse?.data;
       if (!responseData) return item;

       const { id, name, description, imageURL, ...rest :Omit<any, ...>  } = responseData;

           item.key = id;
           item.title = name;
           item.description = description;
           item.image = imageURL;
           item.data = rest;

       return item;
    };

    const saveToCart = (item) :void  => {  Show usages
       let cart = JSON.parse(localStorage.getItem( key: "cart")) || [];
       const existingItem :T  = cart.find(cartItem :T  => cartItem.key === item.key);

       if (existingItem) {
           existingItem.count += 1;
       } else {
           cart.push({...item, count: 1});
       }

       localStorage.setItem("cart", JSON.stringify(cart));
    };


    return (
        <div className = "page-container">
            <div className="pet-container-card-container">
                {petContainerFetched && (
                    <Card
                        item={petContainerCardFilter()}
                        displayNames={petContainerResponses[petContainerFetchResponse?.httpStatusCode]?.displayNames}
                        styles={styles.petContainerCard}
                    >
                         <div className="pet-add-to-cart-button-container">
                            <Button
                                text="Add to Cart"
                                onClick={() :void  => saveToCart(petContainerCardFilter())}
                                styles={styles.petAddToCartButton}
                            />
                         </div>
                    </Card>
                )}
            </div>
        </div>
    )
};
```

Figure 10: Generated React Code (Pet Details Page)

5 Evaluation

To assess the effectiveness and practicality of the proposed framework, we carried out an evaluation that combined qualitative user feedback with quantitative performance metrics. This approach allowed us to understand how users interacted with the system, gather insights on the quality and reliability of the generated frontend, and objectively measure the technical performance of the produced applications. As part of this evaluation, we conducted a user study that gathered 15 responses from 15 different users with different levels of frontend developer experience to evaluate both the usability of the framework and the quality of the generated frontend. By combining user feedback with measured data, we aimed to evaluate the overall ease of use, correctness, and performance of the framework in generating frontend applications.

5.1 Framework Evaluation

5.1.1 Performance Evaluation

To evaluate the performance of the proposed framework, we focused on two primary aspects: the efficiency of frontend generation and the memory usage during the generation process. These quantitative benchmarks help validate the practicality of the framework for real-world use, particularly in terms of scalability and responsiveness.

1. Frontend Generation Time

The framework's generation speed was tested by measuring both CPU time and wall time required to generate varying numbers of frontend pages. Results indicate that the time required grows approximately linearly with the number of pages generated. A 10-page UI is generated in approximately 1.5 seconds, while 100 pages take roughly 3 seconds. This demonstrates that the system scales well without exponential time increases, making it suitable for rapid development of larger applications.

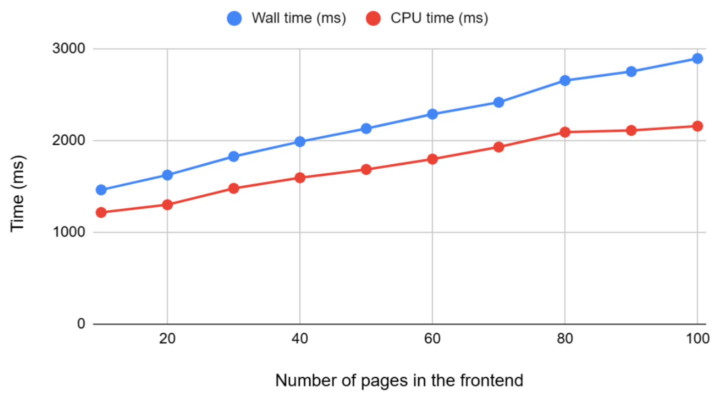


Figure 11: Elapsed time and CPU time (in milliseconds) vs. number of pages generated in the frontend

2. Memory Usage

We also monitored heap memory consumption during the generation process. The system used around 11 MB of heap memory to generate a 10-page UI, while generating 50 pages consumed approximately 16.64 MB (Table 3). This suggests a moderate and near-linear memory growth, which is acceptable and supports the scalability of the tool.

Table 3: Heap memory usage (in MB) for different frontend sizes

Pages	**Heap Memory Delta (MB)**
10	10.97
20	12.55
30	14.47
40	16.26
50	16.64

All performance tests were conducted on the following system configuration:

- **Processor:** Intel® Core™ i5-12500H (12th Gen), 3.10 GHz
- **RAM:** 16 GB
- **Operating System:** Windows 11, 64-bit OS, x64-based processor

These results confirm that the framework is both time-efficient and resource-conscious, making it a viable option for backend developers seeking to generate functional frontends quickly and reliably.

5.1.2 Usability Evaluation

To evaluate the framework's usability, we used the System Usability Scale (SUS), a widely recognized ten-item Likert-scale questionnaire that assesses subjective perceptions of system usability. The questionnaire consists of five positively and five negatively worded statements, each rated on a 5-point scale from 1 (Strongly Disagree) to 5 (Strongly Agree) (Table 4). Participants rated each statement based on their experience using the system. The total SUS score ($TotalSUS$) was derived to assess the usability of the framework.

The total SUS score ($TotalSUS$) is calculated using the Equation 1, where n represents the total number of participants and P_xSUS represents the SUS score of each participant.

$$\text{TotalSUS} = \frac{\sum_{x=1}^{n} P_xSUS}{n} \times 2.5 \tag{1}$$

The individual SUS score (P_xSUS) is calculated using the Equation 2, where VS_i denotes the rating given by participant P_x for statement S_i (Table 4).

$$P_x\text{SUS} = \sum_{1 \leq i \leq 10,\ i\ =\ \text{odd}} (VS_i - 1) + \sum_{1 \leq i \leq 10,\ i\ =\ \text{even}} (5 - VS_i) \quad (2)$$

According to Bangor et al. [19], an interactive system's usability is considered above average if its total SUS score exceeds 70. In our evaluation, the framework achieved an total SUS score of 72.67, reflecting a positive perception of usability among participants. This result indicates that the system is well-integrated, user-friendly, and can be easily adopted by developers with minimal effort.

Table 4: System Usability Scale (SUS) Questionnaire Statements Used for Evaluating Framework Usability

ID	**Questionnaire Statements**
S_1	I think that I would like to use this system frequently
S_2	I found the system unnecessarily complex
S_3	I thought the system was easy to use
S_4	I think that I would need the support of a technical person to be able to use this system
S_5	I found the various functions in this system were well integrated
S_6	I thought there was too much inconsistency in this system
S_7	I would imagine that most people would learn to use this system very quickly
S_8	I found the system very cumbersome to use
S_9	I felt very confident using the system
S_{10}	I needed to learn a lot of things before I could get going with this system

5.2 Generated Frontend Evaluation

5.2.1 Performance Evaluation

To objectively measure the runtime and load performance of the generated application, we utilized established web performance metrics, evaluated using the Lighthouse tool. The following metrics were considered. (Table 5)

The results indicate that the generated applications exhibit fast initial rendering, minimal layout shifts, and efficient resource loading, all contributing to a smooth and responsive user experience. The First Contentful Paint (FCP) ranged from 2.4 s to 2.7 s, and the Largest Contentful Paint (LCP) ranged from 4.3 s to 5.3 s, both within acceptable limits for modern web applications. Additionally, the Cumulative Layout Shift (CLS) remained between 0 and 0.009, ensuring page stability during load. The Total Blocking Time (TBT) was low, ranging from 0 ms to 10 ms, enhancing responsiveness. The Speed Index varied from 2.5 s to 3.9 s, and the Time To Interactive (TTI) ranged from 4.3 s to 5.3 s, both within recommended performance thresholds for responsive web applications [20].

Table 5: Performance Metrics Collected Using Lighthouse

Metric	Description	Min	Max
First Contentful Paint (FCP)	Time taken for the first piece of visible content to appear after navigation.	2.4 s	2.7 s
Largest Contentful Paint (LCP)	Time when the largest visible content element is rendered.	4.3 s	5.3 s
Cumulative Layout Shift (CLS)	Measures unexpected layout shifts during page load.	0	0.009
Total Blocking Time (TBT)	Time when the main thread is blocked, delaying user input.	0 ms	10 ms
Speed Index	Measures how quickly content is visually displayed during page load.	2.5 s	3.9 s
Time To Interactive (TTI)	Time until the page becomes fully interactive and responsive to user input.	4.3 s	5.3 s

5.2.2 Usability Evaluation

The frontend generated for the user study was based on a pet store application, which includes the following key functionalities:

1. **User authentication and authorization**: Users can sign in with different roles (normal user, admin).
2. **Role-based view of frontend components**: Components and pages are conditionally rendered based on the user's role.
3. **Full workflow for normal users**: Includes functionalities such as adding items to the cart, placing orders, and viewing past orders.
4. **User and inventory management for admins**: Admins can add, update, and delete user and inventory entities.
5. **Public access sections**: Certain content, such as the homepage and explore section, is accessible to unauthorized users.

To assess the quality and correctness of the generated frontend, we designed a custom feedback questionnaire that focused on functional accuracy, visual design, responsiveness, code clarity, and overall performance. Participants rated each aspect on a 5-point Likert scale (0 = Strongly Disagree, 4 = Strongly Agree) based on their interaction with the interface (Table 6).

The scores were averaged per question across all participants to assess the system's performance in each area. These average scores, ranging from 3.07 to 3.4 on a 0–4 scale, are presented in Figure 12, providing a clear visual summary of participant feedback.

The highest scores (3.4) were recorded for the generated frontend's expected functionality and correct routing behavior (Q1 and Q2), indicating strong reliability. Moderate scores around 3.2 for UI/UX, responsiveness, and overall performance suggest a generally smooth and user-friendly interface. The lowest score, 3.07, was for form validation and error handling (Q7), highlighting a potential area for improvement. Overall, the results reflect a positive user experience, with minor areas identified for future refinement.

Table 6: Custom Questionnaire Items Used for Evaluating the Generated Frontend

Question No.	**Questionnaire Item**
Q1	Generated frontend worked as expected
Q2	Frontend behaved correctly across all routes
Q3	UI/UX (navigation, layout, clarity)
Q4	Readability and understandability of generated code
Q5	Responsiveness of the generated UI
Q6	Performance of the generated application
Q7	Form validation and error handling

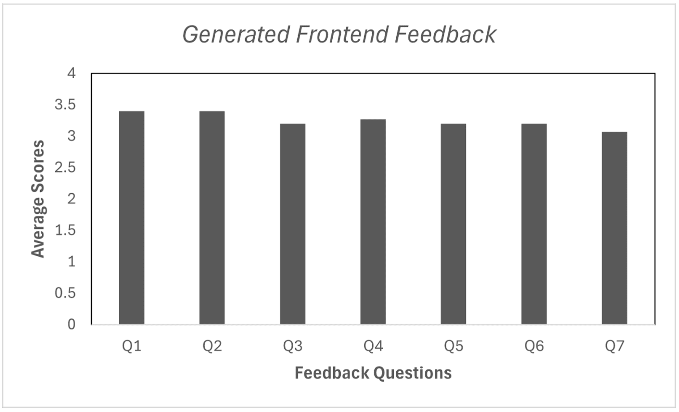


Figure 12: Average Participant Scores for Each Question in Table 6

6 Conclusion

This work presents a framework for the automatic generation of frontend interfaces by integrating OpenAPI Specifications with user-defined layout configurations. The proposed solution focuses on bridging the gap between backend API definitions and functional frontend applications, particularly catering to backend developers who may have limited expertise in frontend technologies. By parsing OpenAPI documents and mapping endpoints to a configurable layout grammar, the system is capable of generating complete, responsive, React-based user interfaces that are ready for integration and deployment.

The framework streamlines the traditional web development workflow by eliminating the repetitive, manual tasks commonly involved in frontend creation. It enables rapid development while ensuring consistency across pages and components. The generated applications support essential CRUD operations, automatically wiring up frontend components to backend endpoints based on the OpenAPI specification. Authentication and authorization mechanisms are also incorporated to facilitate secure user interactions. These features, combined with the flexibility of layout and styling control, make the framework suitable for moderately complex web applications where both development speed and maintainability are critical. This balance between automation and control is a core strength of the framework.

Looking ahead, several avenues for enhancement have been identified. One direction involves extending the XML configuration model to support additional UI components. This would further increase the expressive power of the configuration grammar, enabling developers to generate more diverse and feature-rich applications. Another improvement is the development of an IDE plugin or extension, which would allow developers to configure and preview layouts directly within their development environment, thereby improving usability and reducing context switching. Additionally, incorporating support for popular frontend libraries such as Material-UI would expand the component palette available for generation, allowing for richer visual design and better alignment with industry UI standards.

Overall, the framework represents a meaningful step toward low-code frontend development by reducing the technical barrier for backend-focused developers while maintaining code quality, modularity, and user experience. It lays the groundwork for future research and development in automating full-stack application scaffolding through standardized specifications and configurable design grammars.

References

[1] N. Luburić, G. Savić, G. Milosavljević, M. Segedinac, and J. Slivka, “A code generator for building front-end tier of rest-based rich client web applications,” in *International Conference on Information Society*, Jan. 2016, pp. 28–33.

[2] I. Koren and R. Klamma, “The exploitation of openapi documentation for the generation of web frontends,” in *Companion of the The Web Conference 2018 - WWW ’18*, 2018.

[3] S. Xiao, Y. Chen, J. Li, L. Chen, L. Sun, and T. Zhou, “Prototype2code: End-to-end front-end code generation from ui design prototypes,” *arXiv*, Aug. 2024.

[4] P. P. Arhandi, Y. Pramitarini, and R. Alviandra, “Desain prototype frontend auto generator based on rest api,” in *Seminar Informatika Aplikatif Polinema*, Jan. 2019, pp. 389–393.

[5] C. Si, Y. Zhang, Z. Yang, R. Liu, and D. Yang, “Design2code: How far are we from automating front-end engineering?” *arXiv*, Mar. 2024. [Online]. Available: https://arxiv.org/abs/2403.03163

[6] O. Nikiforova, K. Babris, and F. Mahmoudifar, “Automated generation of web application front-end components from user interface mockups,” in *Proceedings of the 19th International Conference on Software Technologies*, 2024.

[7] I. A. Rohma and N. A. Azurat, “Code generator development to transform ifml (interaction flow modelling language) into a react-based user interface,” *Jurnal Ilmu Komputer dan Informasi*, vol. 17, no. 2, pp. 109–120, Jun. 2024.

[8] X. He, J. Wang, Z. Liu, L. Xie, H. Wang, and X. Le, “Automatic generation of frontend code from design interface,” in *Proceedings of ICIST 2023*, 2023, pp. 94–99.

[9] M. Vaziri, L. Mandel, A. Shinnar, J. Simeon, and M. Hirzel, “Generating chat bots from web api specifications,” in *Onward! 2017*, 2017.

[10] Swagger.io, “Api code & client generator — swagger codegen,” 2020, accessed: 2025-05-09. [Online]. Available: https://swagger.io/tools/swagger-codegen/

[11] OpenAPI Generator, “Openapi generator,” 2025, accessed: 2025-05-09. [Online]. Available: https://openapi-generator.tech/

[12] S. Abrahamsson, “A model to evaluate front-end frameworks for single page applications written in javascript,” DIVA, Tech. Rep., 2023. [Online]. Available: https://liu.diva-portal.org/smash/record.jsf?pid=diva2%3A1758858

[13] M. K. Dobbala, “Validate faster, develop smarter: A review of frontend testing best practices and frameworks,” *Journal of Mathematical & Computer Applications*, pp. 1–4, Jun. 2022.

[14] K. Knowles *et al.*, “Simulation of contraceptive access for adolescents and young adults using a pharmacist-staffed e-platform: Development, usability, and pilot testing study,” *JMIR Pediatrics and Parenting*, vol. 8, Feb. 2025.

[15] X. Fu and E. Jung, "A study on information systems providing local government information: Focusing on website ui/ux evaluation," *Journal of Psychology Research*, vol. 15, no. 2, Feb. 2025.

[16] H. S. Fadhlillah, M. Retno, I. A. Rohma, and E. K. Budiardjo, "Managing customizable user interface for web application product lines using delta modeling," in *Proceedings of the ACM Conference*, Jan. 2024, pp. 61–70.

[17] G. Raj, A. K. Chakraverti, and S. N. Mehta, "User interface and user experience design methodologies and evaluation techniques: A systematic literature review," in *2024 1st International Conference on Advances in Computing, Communication and Networking (ICAC2N)*, Dec. 2024, pp. 398–402.

[18] P. A. Akiki, A. K. Bandara, and Y. Yu, "Adaptive model-driven user interface development systems," *ACM Computing Surveys*, vol. 47, no. 1, pp. 1–33, Jul. 2014.

[19] A. Bangor, P. Kortum, and J. Miller, "Determining what individual sus scores mean: Adding an adjective rating scale," *Journal of Usability Studies*, vol. 4, no. 3, pp. 114–123, May 2009.

[20] Chrome Developers, "Lighthouse performance scoring," 2025, accessed: 2025-05-13. [Online]. Available: https://developer.chrome.com/docs/lighthouse/performance/performance-scoring/?utm_source=lighthouse&utm_medium=devtools

Appendices

APPENDIX A
Publications

Research

Submitted paper to Moratuwa Engineering Research Conference 2025 (MERCon 2025)

- **Title:** A Framework to Automatically Generate Application Frontend Based on OpenAPI Specification for Moderately Complex Applications
- **Date:** 14th and 15th August 2025 at University of Moratuwa

Industry

Submitted to SLASSCOM National Ingenuity Awards – University Category

APPENDIX B
Source Code

Git Repository: https://github.com/ravijar/frontend-generator