



Compsci Technology solutions

Internship Report on CERTIFICATE GENERATOR

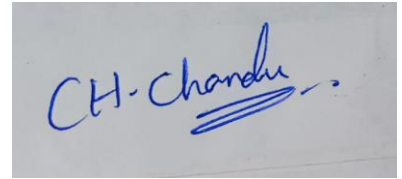
**Prepared by
Chandu Chitteti**

Intern ID :240402

**Under the mentorship of
Mr. Naveen Kumar
&
Dr. Kumar Raja**

DECLARATION

I, Chandu Chitteti, hereby declare that this project work titled “Certificate generator” is a genuine work carried out by me and has not been submitted to any other course or University for the award of any degree by me.

A handwritten signature in blue ink on a light-colored background. The signature reads "CH-Chandu" with a stylized flourish underneath the name.

Signature

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my institute, NBKR institute of Science And Technology Vidhyanagar, and its Placement Cell for providing me with the opportunity to undertake this internship.

I am also extremely grateful to Mr. Naveen Kumar and Dr. Kumar Raja from Compsci Technology solutions for offering me this internship and providing continuous guidance and support throughout the duration. Their insights and mentorship have been instrumental in making this internship a truly enriching experience.

Additionally, I would like to acknowledge my co-interns for their support and collaboration throughout this internship. Their assistance and camaraderie have made this experience both enjoyable and educational.

Thank you to everyone who has contributed to my learning and growth during this period.

Abstract

During my internship at CompSci Technologies, I worked on a project focused on generating certificates. My main task was to build the user interface (UI) using React, a popular library for creating web applications. I designed the UI to handle both single and bulk certificate submissions, making it easy for users to generate certificates as needed. This frontend was connected to a backend built with Django, which managed all the server-side logic and data.

I collaborated closely with the backend team to ensure that the React frontend and Django backend worked well together. This included designing various components, managing the application's state, and integrating API calls to handle data retrieval and submission. I also spent time optimizing the UI for better performance and responsiveness.

This project significantly improved my skills as a full-stack developer. I gained practical experience in combining front-end and back-end technologies to create a seamless and efficient application. My contributions to this certification generation project have given me valuable insights and skills that will be useful in my future career in web development.

TABLE OF CONTENTS :

1. Introduction	1
2. Problem Statement	1
3. Objectives.....	1
4. Scope	2
5. Applications.....	2
6. Limitations.....	2
7. Software Requirements	3
8. Literature survey.....	13
9. Physical Model.....	14
10.Network Model.....	15
11. Software Development Lifecycle	16
12.Implementation.....	17
13.programme listing / Code.....	22
14.Testing Approach	78
15.Testing	79
16.TestPlan.....	82
17.TestCases.....	83
18.BugList.....	87
19.SoftwareDevelopment Lifecycle	88
20.Enhancements and Futurework.....	91
21.Release Notes	93
22.List Of Abbreviations/ Nomenclature.....	94
23.List Of Figures and Screen Shots.....	96
24.Screenschotes.....	97
25.Conclution And Futurework	107
26.UserMannual.....	108
27.References	111

Table of contents

1. Introduction

In today's digital age, generating certificates is an essential but often tedious task across various sectors like education, professional certifications, and event management. Traditionally, this process has been manual, involving significant administrative effort and time. With the shift towards digital solutions, there is a growing need for an automated system that can streamline certificate generation while ensuring accuracy and efficiency.

During my internship at CompSci Technologies, I contributed to a project aimed at developing an automated certificate generation system. The project focused on creating a seamless and efficient method for generating certificates using a combination of React for the user interface (UI) and Django for the backend. The goal was to facilitate both single and bulk certificate submissions, optimize performance, and enhance user experience.

2. Statement of the Problem

The manual process of certificate generation is fraught with inefficiencies, errors, and delays, especially when dealing with large volumes. This traditional approach is not only time-consuming but also prone to inconsistencies, which can undermine the credibility of the issuing organization. There is a pressing need for an automated system that can handle certificate generation more efficiently, reducing administrative burden and minimizing errors.

3.Objectives

The primary objective of the certificate generation project is to develop a comprehensive solution that addresses the inefficiencies of manual certificate generation processes. Key objectives include:

- Designing and implementing a user-friendly interface using React, a leading JavaScript library for building interactive user interfaces.
- Integrating the frontend seamlessly with a robust backend system powered by Django, a high-level Python web framework, to manage server-side logic and data operations.
- Facilitating both single and bulk certificate generation capabilities to accommodate varying operational needs.
- Ensuring smooth communication and data flow between the frontend and backend components through well-defined API integrations.
- Enhancing the overall user experience by optimizing the frontend for performance, responsiveness, and ease of use.

4. Scope

The scope of the project encompasses the development and deployment of a scalable certificate generation system. Key aspects of the scope include:

- Creation of a React-based frontend application tailored for intuitive certificate generation workflows.
- Implementation of backend functionalities within Django to handle data storage, retrieval, and processing related to certificate generation.
- Provision of features supporting the generation of individual certificates as well as batch processing for multiple certificates simultaneously.
- Integration of secure API endpoints to facilitate seamless interaction between the frontend application and backend services.
- Optimization efforts focused on ensuring the system's responsiveness, reliability, and scalability to meet anticipated user demands.

5. Applications

The developed certification generation system offers versatile applications across various domains, including but not limited to:

- Educational institutions seeking efficient methods to issue academic certificates and diplomas.
- Professional organizations involved in certifying skills, qualifications, and achievements of their members or employees.
- Event organizers aiming to swiftly distribute participant certificates for conferences, workshops, and other events.
- Corporate entities requiring streamlined processes for internal certifications, compliance training, and employee recognition programs.

6. Limitations

While the certification generation system promises substantial benefits, it also faces certain limitations:

- Initial setup and configuration may necessitate technical expertise in web development and server administration.
- Dependence on backend infrastructure reliability, including server uptime and database performance, to maintain system functionality.
- Customization options for certificate templates may be constrained within the scope of predefined design and formatting parameters.
- Resource-intensive operations during bulk certificate generation processes may require adequate server resources and capacity planning.

Software Requirements Specification For Certificate Generation Project

1.Introduction

1.1 Problem Statement

The traditional method of generating certificates manually is fraught with several challenges. This labor-intensive process requires significant time and effort from staff, which can lead to delays and inefficiencies, especially when dealing with large volumes of certificates. Moreover, manual certificate generation is prone to human errors, such as incorrect details, formatting issues, and inconsistencies, which can compromise the professionalism and reliability of the certificates issued. As organizations and institutions grow and the demand for certificates increases, scaling up this process becomes increasingly difficult and inefficient, further exacerbating these issues.

To address these challenges, there is a clear need for an automated certificate generation system that minimizes human intervention, reduces errors, and improves efficiency. Such a system would streamline the certificate creation process, allowing for quick and accurate generation of certificates even in large quantities. By automating this task, organizations can ensure consistency and accuracy in their certificate issuance, thereby saving time and resources.

1.2 purpose

The primary purpose of the Certificate Generator is to provide an efficient and user-friendly platform for generating certificates. By automating the certificate creation process, this application eliminates the need for manual interference, tat saves time and resources.

1.3 Intended Audience

The Certificate Generator is designed for various user groups, including Educational Institutions ,Companies and Event Organizers.

Educational institutions can use the Certificate Generator to create certificates for academic achievements, extracurricular accomplishments, and special recognitions for students.

Companies can use the Certificate Generator to create certificates for employee of the month, outstanding performance, or completion of internal training programs.

Event organizers can streamline the process of creating certificates for participants, speakers, and volunteers involved in conferences, workshops, or seminars.

1.4 Project Scope

The Certificate Generator project aims to provide a user-friendly web application for generating certificates. It has the following Features :

- * Selection of predefined certificate templates or custom templates provided by users.
- * Generation of single or bulk certificates based on user specifications.
- * Ability to send bulk certificates via email to specified recipients.
- * Download option for single certificates directly from the web interface.

1.5 References

Official documentation for Django web framework.

Official documentation for Python programming language.

Documentation for ReportLab library for PDF generation.

Documentation for Django Mailer library for email sending in Django applications.

2. Overall Description

2.1 Product Perspective

The Certificate Generator is a web application designed to operate on all other systems. It provides users with a convenient platform for generating certificates without relying on external services. However, it may integrate with existing email services for sending bulk certificates.

2.2 Product Features

Key Features:

Users can choose predefined templates or upload custom ones.

Supports single and bulk certificate generation.

Allows users to send bulk certificates via email.

Provides the option to download single certificates directly from the web interface.

2.3 User Classes and Characteristics

User Classes:

Administrators - Manage templates, users, and system settings.

Template Designers - Create and upload custom certificate templates.

End Users - Generate and download certificates.

User Characteristics:

Administrators - Have full control over the system and require administrative privileges.

Template Designers - Posts designs and understand template creation processes.

End Users - Include students, employees, or event participants who need certificates.

2.4 Operating Environment

The Certificate Generator operates within a web environment and is accessible through modern web browsers. It is built using Django, a Python web framework, and requires a compatible web server. That could be Apache, etc.

2.5 Design and Implementation Constraints

Compatibility : Must be compatible with a wide range of web browsers and devices.

Security: Implement proper authentication and authorization mechanisms to ensure data security.

Scalability: Design the system to handle increasing user loads and data volumes.

Performance: Optimizing code and database queries for efficient and smooth performance.

2.6 User Documentation

User documentation will include instructions on template selection, certificate generation, email distribution, and other key functionalities.

2.7 Assumptions and Dependencies

Assumptions:

Users have basic knowledge of operating web applications and navigating web interfaces.

Users have access to a stable internet connection for accessing the Certificate Generator.

The system will be hosted on a reliable web server with sufficient resources.

Dependencies:

Relies on Django for web application development.

Requires Python programming language for server-side logic.

Depends on a database system (e.g., PostgreSQL, MySQL) for data storage.

A database connection will be established to store and retrieve data.

3. System Features

3.1 System Feature 1. Template Selection

Description: Users can select from a variety of predefined certificate templates or upload their own custom templates.

Functionality:

Predefined Templates: Display a list of available predefined templates.

Custom Templates: Provide an option to upload custom templates.

User Interaction: Users can browse through the available templates and select the one that best suits their requirements.

3.2 System Feature 2: Certificate Generation

Description: The system supports both single and bulk certificate generation based on user specifications.

Functionality:

Single Certificate - Allow users to generate a single certificate by filling in relevant details (e.g., recipient name, event name, date).

Bulk Certificate - Enables users to generate multiple certificates simultaneously by uploading a CSV file containing recipient information.

Friendly User Interaction - Users provide necessary information through a web form or CSV file upload, triggering the generation process.

3.3 System Feature 3: Email Distribution

Description: Users can send bulk certificates via email to specified recipients.

Functionality:

- * Configure email settings such as SMTP server details and sender email address.
- * Specify email addresses or upload a CSV file with recipient email information.
- * Trigger the sending of certificates via email to the selected recipients.
- * Users input recipient email addresses and initiate the email sending process from the system interface.

3.4 System Feature 4: Instant Download

Description: Users have the option to download single certificates directly from the web interface.

Functionality:

- * Ensure certificates are available for download in PDF format.
- * Users click on the download button to save the certificate to their local device.

4. External Interface Requirements

4.1 User Interfaces

The user interface should be intuitive, user-friendly, and accessible across different devices and screen sizes.

Requirements:

Template Selection Interface: Provide a visually appealing interface for selecting predefined or custom templates.

Certificate Generation Form: Design a form for users to input details and generate certificates.

Email Configuration Interface: Allow administrators to configure email settings (SMTP server details, sender email address).

Bulk Certificate Email Selection: Enable users to specify recipient email addresses or upload a CSV file for bulk certificate email distribution.

Download Button: Include a prominent download button for users to download generated certificates.

4.2 Hardware Interfaces

The Certificate Generator is a web-based application and does not have direct hardware dependencies. It should be compatible with standard web browsers and devices.

4.3 Software Interfaces

The Certificate Generator relies on the following software interfaces:

* Utilizes Django for web application development.

- * Uses Python programming language for server-side logic.
- * Interfaces with a database system (e.g., SQL, MySQL) for data storage and retrieval.
- * Integrates with email services (e.g., SMTP) for bulk certificate email distribution.

4.4 Communications Interfaces

The Certificate Generator communicates with external systems via HTTP requests for email distribution and other external services.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

The system should respond to user requests and handle concurrent user interactions efficiently.

Requirements:

Response Time: Ensure that the system responds to user actions within a reasonable time frame (e.g., under 2 seconds).

Scalability: Design the system to scale horizontally to accommodate increasing user loads.

5.2 Safety Requirements

The Certificate Generator may not have specific safety requirements as it does not deal with critical or hazardous operations. But basic

authentication for users is done to make sure their certifications related things are private.

5.3 Security Requirements

Ensure the security of user data, authentication mechanisms, and protection against common web security threats such as SQL injection.

Requirements:

Secure Authentication: Implement secure authentication mechanisms (e.g., password hashing, session management).

Data Encryption: Encrypt sensitive data such as user credentials and certificate content.

5.4 Software Quality Attributes

The Certificate Generator should exhibit high software quality attributes, including reliability, maintainability, and usability.

Reliability: Ensuring the system operates reliably without unexpected failures or errors.

Maintainability: Designing the system with clean, modular code to facilitate future maintenance and enhancements.

Usability: Prioritize user experience and interface design to make the system friendly and easy to use.

8.Literature Survey

This is a review of established knowledge (literature overview), which is relevant to the main topic of certificate generation systems. It may include a historical review, a critical account of more recent work, and an exposition of the theory and techniques used for the practical work.

Overview of Certificate Generation Systems

Historically, certificate generation has been a manual process, often leading to inefficiencies and errors. The transition to automated systems has been driven by the need for accuracy, efficiency, and scalability in various sectors such as education, professional certifications, and event management.

Technologies Used in Automated Systems

Advancements in web technologies have significantly impacted the automation of administrative tasks. The use of React for frontend development and Django for backend operations provides robust frameworks for creating responsive user interfaces and scalable backend systems. These technologies are essential for handling the complex data operations required in certificate generation applications.

Integration of APIs and Data Management

The integration of APIs is crucial for seamless communication between frontend and backend components. APIs facilitate efficient data retrieval, processing, and storage, which enhances the overall performance and responsiveness of automated systems. This integration ensures that data flows smoothly and securely across the application.

User Experience and Interface Design

Research has shown that user interface design greatly influences user experience and system usability. Intuitive design principles and responsive interfaces are critical for optimizing user interactions in certificate generation processes. Studies emphasize the importance of creating user-friendly interfaces that enhance usability and satisfaction.

Challenges and Future Directions

Despite significant progress, challenges remain in customizing certificate templates, ensuring system scalability, and maintaining backend infrastructure reliability. Future research may focus on improving system robustness, expanding customization options, and exploring emerging technologies to enhance automation and efficiency. Addressing these challenges will be vital for the continued advancement of certificate generation systems.

9. Physical Model / Analysis

Overview

In the context of certificate generation systems, a physical model analysis involves understanding and designing the system's hardware and infrastructure components. This analysis ensures that the system can handle the required processing, storage, and networking tasks efficiently and reliably.

System Architecture

The physical architecture of the certificate generation system includes the following key components:

Servers: The system uses web servers to host the frontend application built with React and application servers to run the Django backend. Database servers store and manage the certificate data.

Networking: Robust networking infrastructure is necessary to ensure seamless communication between the frontend and backend components. This includes local area networks (LAN) within the hosting environment and wide area networks (WAN) for external user access.

Storage: Adequate storage solutions are essential for managing certificate templates, user data, and generated certificates. This may involve using relational databases like PostgreSQL or MySQL, along with file storage solutions for handling large files.

Load Balancing and Scalability

To ensure the system can handle high volumes of certificate generation requests, load balancing techniques are implemented. Load balancers distribute incoming traffic across multiple servers, preventing any single server from becoming a bottleneck. This approach enhances the system's scalability, allowing it to accommodate growing user demands.

Security Measures

Security is a critical aspect of the physical model. Measures include:

Firewalls: Protect the network and servers from unauthorized access and attacks.

- **Encryption:** Secure data transmission between the frontend and backend components, ensuring the privacy and integrity of sensitive information.
- **Regular Backups:** Implementing regular backup routines to prevent data loss and ensure system recovery in case of hardware failure or cyber incidents.

Performance Optimization:

Performance optimization involves monitoring and adjusting the hardware and network configurations to ensure the system runs efficiently. This includes:

- **Server Resource Management:** Monitoring CPU, memory, and disk usage to prevent resource exhaustion and optimize performance.
- **Network Latency:** Minimizing network latency to improve response times and user experience.
- **Database Tuning:** Regularly optimizing database queries and indexes to enhance data retrieval speeds.

Environmental Considerations

Physical model analysis also considers the environmental impact and energy efficiency of the system. This includes:

- **Energy-efficient Hardware:** Using energy-efficient servers and networking equipment to reduce power consumption.
- **Data Center Location:** Choosing data centers in regions with favorable environmental conditions and energy policies to minimize the carbon footprint.

10.Network Model/mathematical model/Design

Architecture Design

- **Client-Server Architecture:** Defines how clients interact with the server.
- **Microservices Architecture:** Explains modular service handling.
- **API Integration:** Details RESTful API usage for frontend-backend communication.

Design

Scalability and Security

- **Scalability:** Discusses horizontal scaling and load balancing.
- **Security:** Covers HTTPS, data encryption, and access control.

Fault Tolerance and User Experience

- **Fault Tolerance:** Addresses redundancy and failover systems.
- **User Experience:** Focuses on usability and responsive design principles.

11. Agile Process Overview

In the certificate generation project, I utilized Agile methodology to ensure a structured and collaborative development process. Here's a summary of the work I've done as per the Agile process:

1. Sprint Planning: I started by defining project goals and user stories related to certificate generation. I prioritized tasks based on their importance and dependencies, creating a backlog for the sprint.

2. Daily Stand-ups: I participated in daily meetings to discuss my progress, roadblocks, and next steps. This helped maintain clear communication within the team and ensured we stayed aligned on our tasks.

3. Development Iterations: I worked in iterative cycles, focusing on specific features, including:

- **Frontend Development:** I implemented the React UI for certificate generation and submission forms.
- **Backend Integration:** I connected the React frontend to the Django backend to handle API requests and data storage.

4. Continuous Feedback: I regularly sought feedback from team members and stakeholders on my progress and the functionality of the application, allowing for necessary adjustments and improvements.

5. Testing and Review: I conducted unit testing and integration testing during each sprint to ensure the quality of the features I implemented. I also participated in code reviews, providing constructive feedback and learning from my peers.

6. Sprint Review and Retrospective: At the end of each sprint, I demonstrated the completed features to stakeholders for approval and feedback. I engaged in retrospective meetings to reflect on what worked well and what could be improved in future sprints, fostering a culture of continuous improvement.

Contributions

- **UI Development:** I designed and implemented user-friendly interfaces for single and bulk certificate submissions.
- **API Development:** I assisted in setting up and integrating Django REST APIs to manage certificate data effectively.
- **Session Management:** I implemented session management features to enhance user authentication and data persistence.
- **Collaboration:** I worked closely with team members, actively contributing to discussions and decisions that shaped the project.

Conclusion

By following the Agile methodology, I contributed to a flexible and adaptive development process that prioritized collaboration, customer feedback, and continuous improvement, resulting in a successful certificate generation project.

12.Implementation/simulation

Django :

Create a virtual environment for our project :

```
Python -m venv env
```

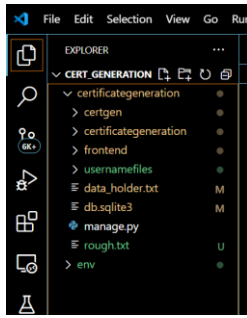
Where env is the name of our environment

Activate it : `./env/scripts/activate` ----- in powershell

`env/scripts/activate` ----- in command prompt

Install Django: Ensure you have Django installed in your environment. You can install Django using pip:

```
pip install django
```



Create a Django Project: To create a new Django project, run the following command:

```
django-admin startproject certificategeneration
```

Navigate to the Project Directory: Change into your project directory:

```
cd projectname
```

In our case its `cd certificategeneration`

Create a Django App: To create an app within your project, run:

```
python manage.py startapp appname
```

In our case ,... certgen is the name of our app

Our project strucucture looks like :

```
projectname/
  manage.py
projectname/
  __init__.py
  settings.py
  urls.py
  wsgi.py
appname/
  __init__.py
  admin.py
  apps.py
```

```
models.py
tests.py
views.py
migrations/
__init__.py
```

Creating Initial Migrations: Run the following command to create initial migrations for our app:

```
python manage.py makemigrations
```

Apply Migrations: Apply the migrations to your database:

```
python manage.py migrate
```

Run the Development Server: Finally, you can start the development server to see our project in action: `python manage.py runserver`

If we Visit `http://127.0.0.1:8000/` in our web browser, we should see the Django welcome page.

Later, We need to create `Serializer.py` file in our project structure for serializing data. Our project files are described as following :

- **Admin.py file :**

`django.contrib.admin:` This module provides the admin interface.

`certgen.models:` This imports the `Certificate` and `BulkCertificates` models from our `certgen` Django application.

`admin.site.register(Certificate):` This line registers the `Certificate` model with the Django admin interface. Once registered, you can perform CRUD (Create, Read, Update, Delete) operations on `Certificate` objects via the admin interface.

`admin.site.register(BulkCertificates):` Similarly, this line registers the `BulkCertificates` model, allowing admin interface access to manage `BulkCertificates` objects.

models.py file :

The models.py file for our Django project, defines two models: Certificate and BulkCertificates. These models are used to create database tables that store information related to individual certificates and bulk certificate submissions respectively. The Certificate model includes fields like heading, certificate_about, certificant_name, issue_date, company_name, and certificate_provider_name. The BulkCertificates model includes additional fields such as usernames, and number_of_certificates for handling bulk submissions.

Serializer.py File

We've defined serializers for your Django models Certificate and BulkCertificates in the serializers.py file using Django REST Framework. These serializers convert complex data types like querysets and model instances into native Python datatypes, which can then be easily rendered into JSON or XML formats.that's what serializers module does.

The CertificateSerializer and BulkCertificateSerializer classes use the ModelSerializer provided by Django REST Framework. This allows you to automatically generate serializers based on your model definitions. By specifying fields = '__all__', you include all fields of the respective models (Certificate and BulkCertificates) in the serialized output. You can also include then specifically.

Views.py File

Template Rendering:

We Ensure that each view (index, certgen, formresults) renders the appropriate template based on its intended functionality. Currently, they all render through index.html.

ViewSet: we've defined two viewsets:

CertificateView: Inherits from ModelViewSet provided by Django REST Framework. It handles CRUD operations (Create, Retrieve, Update, Delete) for the Certificate model.

- queryset = Certificate.objects.all(): Retrieves all instances of the Certificate model.
- serializer_class = CertificateSerializer: Specifies the serializer class (CertificateSerializer) to be used for serializing/deserializing Certificate instances.

BulkCertificateView: Similarly, this viewset manages CRUD operations for the BulkCertificates model.

- `queryset = BulkCertificates.objects.all()`: Retrieves all instances of the `BulkCertificates` model.
- `serializer_class = BulkCertificateSerializer`: Specifies the serializer class (`BulkCertificateSerializer`) for `BulkCertificates`.

urls.py File:

Imports: We've imported necessary modules and components from Django (include, path, admin) and from our certgen app (views, CertificateView, BulkCertificateView).

Router Setup: We've configured a `DefaultRouter` from Django REST Framework (`routers.DefaultRouter()`).

URL Patterns:

1. `/admin/` : Directs requests to Django's admin interface.
2. `/` : Routes requests to the index view in `views.py`.
3. `/certgen/` : Routes requests to the `certgen` view in `views.py`.
4. `/formresults/` : Routes requests to the `formresults` view in `views.py`.
5. `/api/` : Includes URLs configured by the `DefaultRouter` (`route.urls`) for handling API endpoints related to certificates and bulk certificates.

Settings.py file

General Settings

- **BASE_DIR:** Defines the base directory of the Django project.
- **SECRET_KEY:** Used for cryptographic signing and should be kept secret in production.
- **DEBUG:** Enabled for development; should be False in production for security reasons.
- **ALLOWED_HOSTS:** List of strings representing the host/domain names that this Django site can serve.

Installed Apps

- Includes standard Django apps (admin, auth, contenttypes, etc.).
- Additional apps like `frontend`, `certgen`, `rest_framework`, and `corsheaders` for extended functionality.

Middleware

- Various middleware components configured to handle security, sessions, CSRF protection, authentication, messages, and CORS headers.

REST Framework Configuration

- Configures the Django REST Framework with default permission settings (AllowAny) and CORS settings (CORS_ORIGIN_ALLOW_ALL).

Templates

- Configured to load templates from both the standard app directories (APP_DIRS=True) and a custom directory (frontend/build).

Database

- Uses SQLite as the default database engine ('django.db.backends.sqlite3').
- Database file ('db.sqlite3') located in the project's base directory (BASE_DIR).

Static Files

- Configured to serve static files (CSS, JavaScript, Images) from frontend/build/static directory.

Security Settings

- Includes password validation settings (AUTH_PASSWORD_VALIDATORS).
- CSRF trusted origins and CORS whitelist settings for security and cross-origin requests.

Reactjs:

Setting Up our React Project

Install Node.js and npm: Ensure you have Node.js and npm (Node Package Manager) installed. You can download and install them from the [official Node.js website](https://nodejs.org/en/).

Create a New React App: Use Create React App, a tool that sets up a new React project with a sensible default configuration. Open our terminal and run:

```
npx create-react-app myapp
```

In our case its frontend as we are using react fro frontend.

This command creates a new directory with the same name as your project containing the basic React project structure.

Navigate to the Project Directory: Change into your project directory:

```
cd myapp
```

Start the Development Server: Start the React development server:

```
npm start
```

This command will start the development server and open our new React application in our default web browser at `http://localhost:3000/`.

Basic Structure

After running the above commands, our directory structure should look something like this:

```
myapp/  
  node_modules/  
  public/  
    index.html  
    ...  
  src/  
    App.css  
    App.js  
    App.test.js  
    index.css  
    index.js  
    logo.svg  
    reportWebVitals.js  
    setupTests.js  
  .gitignore  
  package.json  
  README.md  
  ...
```

Later on the following describes the page content of our react app:

App.js file :

BrowserRouter: Provides the routing context for our application, enabling client-side navigation using the HTML5 history API.

Routes: The `<Routes>` component contains individual `<Route>` components that define mappings between URLs (paths) and React components (element prop).

Route Components:

- `<Route exact path="/" element={<Templatelist/>} />`: This route specifies that when the path is exactly '/', the `Templatelist` component will be rendered.

- `<Route path="/certgen" element={<CertGen />} />`: Maps the '/certgen' path to render the CertGen component.
- `<Route path="/formresults" element={<FormResults />} />`: Associates the '/formresults' path with the FormResults component.

Components: Each element prop points to a specific React component (Templatelist, CertGen, FormResults). These components are imported from their respective files ("./pages" for pages and "./components" for components).

Styling: Imports App.css for styling purposes, though the specific styling details aren't included here.

Templatelist.js File

State Management (useState): Uses the useState hook to manage the selectedOption state, which stores whether the user has selected to upload a file or choose a template.

File Upload Handling (handleFileChange): Updates selectedOption with the uploaded file when the user selects an image file.

Template Selection Handling (handleTemplateChange): Updates selectedOption with the selected template ID when the user chooses a template.

Generate Certificate (generateCertificate):

- Constructs a URL (url) based on the selectedOption.
- Redirects the user to the generated URL to proceed with certificate generation.
- Displays an alert if no option (file or template) is selected when attempting to generate.

Template Options: Iterates over an array of template IDs ([1, 2, 3, 4, 5]) to display radio buttons for each template option. Each radio button allows the user to select a template.

Button and SVG Decoration: Renders a "Generate Now" button that triggers the generateCertificate function. Includes a decorative SVG for visual appeal within the button.

Styling: Imports templatelist.css for styling, although the specific CSS details are not included here.

CertGen.js File

State Management (useState): Manages form fields and error messages using useState hooks.

handleCertificateTypeChange: Updates certificateType state based on radio button selection for single or bulk certificates.

submitFormData: Handles form submission;

- Checks if certificateType is selected; displays an error if not.
- Uses axios to post form data (FormData) to appropriate endpoints (/api/certificates/ for single, /api/bulkcertificates/ for bulk).
- Stores the submitted ID and form type (single or bulk) in sessionStorage.
- Redirects to /formresults after successful submission.

Conditional Rendering: Displays different form fields based on certificateType selection (single or bulk).

Formresults.js file

State Management (useState): Manages submittedId (for storing submitted ID) and formType (for storing form type - single or bulk).

Hook (useEffect) : Retrieves submittedId and formType from sessionStorage on component mount and sets state accordingly.

Conditional Rendering:

- Displays form submission results (submittedId and formType).
- Displays appropriate message if no results are found.

Performance Evaluation

Performance evaluation is crucial to ensure the certificate generation system operates efficiently under different conditions. This section focuses on assessing key performance metrics and optimizing system performance.

Key Metrics

- **Response Time:** Measure the time taken to process certificate generation requests and return responses to users.
- **Throughput:** Evaluate the system's capacity to handle concurrent certificate generation requests over a specific period.
- **Scalability:** Assess how well the system scales with increasing user load and certificate generation demands.
- **Resource Utilization:** Monitor CPU, memory, and disk usage to optimize resource allocation and avoid bottlenecks.
- **Error Rate:** Track the occurrence of errors during certificate generation processes and identify their causes.

Methodology

Testing Environment: Describe the setup used for performance testing, including hardware specifications, network configuration, and software versions.

Load Testing: Conduct load tests to simulate realistic user scenarios and assess system behavior under varying loads. Use tools like Apache JMeter or Gatling for load generation.

Stress Testing: Apply stress tests to push the system beyond its normal operational capacity to identify breaking points and performance degradation thresholds.

Scalability Testing: Evaluate how well the system scales by increasing the number of concurrent users or certificate generation requests.

Monitoring: Implement monitoring tools to collect real-time performance data during testing, focusing on response times, throughput, and resource utilization.

Results and Analysis

Response Time Analysis: Present response time data under different load conditions and analyze trends or patterns observed.

Throughput Analysis: Discuss the system's throughput capacity and any observed limitations during peak loads.

Scalability Analysis: Evaluate scalability based on performance metrics and identify areas for improvement in handling increased workload.

Resource Utilization: Analyze CPU, memory, and disk usage patterns to optimize resource allocation and improve overall system efficiency.

Optimization Strategies

Code Optimization: Review and optimize frontend and backend code to enhance performance and reduce response times.

Database Optimization: Optimize database queries, indexes, and transactions to improve data retrieval and storage efficiency.

Caching: Implement caching mechanisms to store frequently accessed data and reduce load on backend services.

Infrastructure Scaling: Consider horizontal scaling options and load balancing strategies to distribute workload effectively across multiple servers.

Appendix :

13.Programme Listing/Code :

Admin.py

```
from django.contrib import admin

from certgen.models import Certificate, BulkCertificates


# Register your models here.

admin.site.register(Certificate)

admin.site.register(BulkCertificates)
```

Models.py

```
from django.db import models

import uuid

class Certificate(models.Model):

    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)

    heading = models.CharField(max_length=100, null=False, blank=False)

    certificate_about = models.TextField(max_length=150, blank=False, null=False )

    certificant_name = models.CharField(max_length=100, blank=False, null=False )

    issue_date = models.DateField(default='2024-01-01')

    company_name = models.CharField(max_length=100, blank=False, null=False )

    certificate_provider_name = models.CharField(max_length=100,
blank=False, null=False)

    def __str__(self):

        return str(self.id) # uid generated is actually a alphanumerical stuff. we are
treating it to string.

class BulkCertificates(models.Model):
```



```

id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)

heading = models.CharField(max_length=100, blank=False, null=False)

certificate_about = models.TextField(max_length=150, blank=False, null=False)

usernames = models.FileField(upload_to='usernamefiles/', blank=False, null=False)

issue_date = models.DateField(default='2024-01-01')

company_name = models.CharField(max_length=100, blank=False, null=False)

certificate_provider_name = models.CharField(max_length=100, blank=False,
null=False)

email = models.EmailField(blank=False, null=False)

number_of_certificates = models.PositiveIntegerField(blank=False, null=False)

def __str__(self):

    return str(self.id)

```

Serializer.py

```

from rest_framework import serializers

from certgen.models import Certificate, BulkCertificates

class CertificateSerializer(serializers.ModelSerializer):

    class Meta:

        model = Certificate

        fields = '__all__'

class BulkCertificateSerializer(serializers.ModelSerializer):

    class Meta:

```

```
model = BulkCertificates
```

```
fields = '__all__'
```

Views.py

```
from django.shortcuts import render
```

```
from . models import *
```

```
from .serializer import CertificateSerializer, BulkCertificateSerializer
```

```
from rest_framework import viewsets
```

```
def index(request):
```

```
    return render(request, 'index.html')
```

```
def certgen(request):
```

```
    return render(request, 'index.html')
```

```
def formresults(request):
```

```
    return render(request, 'index.html')
```

```
class CertificateView(viewsets.ModelViewSet):
```

```
    queryset = Certificate.objects.all()
```

```
    serializer_class = CertificateSerializer
```

```
class BulkCertificateView(viewsets.ModelViewSet):
```

```
    queryset = BulkCertificates.objects.all()
```

```
    serializer_class = BulkCertificateSerializer
```

Urls.py

"""

URL configuration for certificategeneration project.

The `urlpatterns` list routes URLs to views. For more information please see:

<https://docs.djangoproject.com/en/5.0/topics/http/urls/>

Examples:

Function views

1. Add an import: `from my_app import views`
2. Add a URL to `urlpatterns`: `path("", views.home, name='home')`

Class-based views

1. Add an import: `from other_app.views import Home`
2. Add a URL to `urlpatterns`: `path("", Home.as_view(), name='home')`

Including another URLconf

1. Import the `include()` function: `from django.urls import include, path`
2. Add a URL to `urlpatterns`: `path('blog/', include('blog.urls'))`

"""

`from django.contrib import admin`

`from django.urls import include, path`

`from certgen import views`

`from rest_framework import routers`

`from certgen.views import *`

`route = routers.DefaultRouter()`

`route.register(r'certificates', CertificateView)`

`route.register(r'bulkcertificates', BulkCertificateView)`

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", views.index),
    path('certgen/', views.certgen),
    path('formresults/', views.formresults),
    path('api/', include(route.urls)),
]
```

Settings.py

```
"""
```

Django settings for certificategeneration project.

Generated by 'django-admin startproject' using Django 5.0.2.

For more information on this file, see

<https://docs.djangoproject.com/en/5.0/topics/settings/>

For the full list of settings and their values, see

<https://docs.djangoproject.com/en/5.0/ref/settings/>

```
"""
```

```
from pathlib import Path
```

```
import os
```

```
# Build paths inside the project like this: BASE_DIR / 'subdir'.
```

```
BASE_DIR = Path(__file__).resolve().parent.parent
```

```
# Quick-start development settings - unsuitable for production
```

```
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/
```

```
# SECURITY WARNING: keep the secret key used in production secret!
```

```
SECRET_KEY = 'django-insecure-$le0mo7p5^@7&)bw^0f=e!23%yqn$kc2@tzh-8n(mhs)ua@hn%'
```

```
# SECURITY WARNING: don't run with debug turned on in production!
```

```
DEBUG = True
```

```
ALLOWED_HOSTS = []
```

```
# Application definition
```

```
INSTALLED_APPS = [
```

```
    'django.contrib.admin',
```

```
    'django.contrib.auth',
```

```
    'django.contrib.contenttypes',
```

```
    'django.contrib.sessions',
```

```
    'django.contrib.messages',
```

```
    'django.contrib.staticfiles',
```

```
    'frontend',
```

```
    'certgen',
```

```
    'rest_framework',
```

```
    'corsheaders',
```

```
]
```

```
MIDDLEWARE = [
```

```
    'django.middleware.security.SecurityMiddleware',
```

```
    'django.contrib.sessions.middleware.SessionMiddleware',
```

```
    'django.middleware.common.CommonMiddleware',
```

```
    'django.middleware.csrf.CsrfViewMiddleware',
```

```
    'django.contrib.auth.middleware.AuthenticationMiddleware',
```

```
    'django.contrib.messages.middleware.MessageMiddleware',
```

```

'django.middleware.clickjacking.XFrameOptionsMiddleware',
'corsheaders.middleware.CorsMiddleware',
]

REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny',
    ]
}

CORS_ORIGIN_ALLOW_ALL = True

CSRF_TRUSTED_ORIGINS = ['http://localhost:3000']

ROOT_URLCONF = 'certificategeneration.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            os.path.join(BASE_DIR, 'frontend/build'),
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ]
        }
    }
]

```

```
    ],  
    },  
    },  
]
```

```
WSGI_APPLICATION = 'certificategeneration.wsgi.application'
```

```
# Database
```

```
# https://docs.djangoproject.com/en/5.0/ref/settings/#databases
```

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': BASE_DIR / 'db.sqlite3',  
    }  
}
```

```
CORS_ORIGIN_WHITELIST = ['http://localhost:3000']
```

```
CSRF_TRUSTED_ORIGINS = []
```

```
# Password validation
```

```
# https://docs.djangoproject.com/en/5.0/ref/settings/#auth-password-validators
```

```
AUTH_PASSWORD_VALIDATORS = [  
    {  
        'NAME':  
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',  
    },  
]
```

```

{
    'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
},
{
    'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
},
{
    'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
},
]

```

Internationalization

<https://docs.djangoproject.com/en/5.0/topics/i18n/>

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True

Static files (CSS, JavaScript, Images)

<https://docs.djangoproject.com/en/5.0/howto/static-files/>

STATIC_URL = 'static/'

STATICFILES_DIRS = [

os.path.join(BASE_DIR, 'frontend/build/static'),

]


```
# Default primary key field type

# https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

App.js

```
import { BrowserRouter, Route, Routes } from "react-router-dom";

import Templatelist from "./pages/templatelist";

import CertGen from "./pages/certgen";

import ListOfTemplates from "./pages/listoftemplates";

import CertificateForm from "./pages/certificate_form";

import FormResults from "./components/formresults";

import './App.css'

function App() {

  return (

    <div>

      <BrowserRouter>

        <Routes>

          <Route exact path="/" element={ <Templatelist/> } />

          <Route path="/certgen" element={ <CertGen /> } />

          <Route path="/formresults" element={ <FormResults /> } />

          <Route path="/listoftemplates" element={ <ListOfTemplates /> } />

          <Route path="/certificateform" element={ <CertificateForm /> } />

        </Routes>

      </BrowserRouter>

    </div>

  )
}
```

```
    </div>

  )
}

export default App;
```

App.css

```
.template-list {

  margin: 50px;

  padding: 20px;

  background-color: #e393c6;

  border-radius: 10px;

  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

}


.app-container {

  margin: 30px;

  padding: 20px;

  background-color: #ceabf8;

  border-radius: 10px;

  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

}


h1 {

  text-align: center;
```

```
font-family: 'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans
Unicode', Geneva, Verdana, sans-serif;
```

```
}
```

```
.cert-image {  
  
    height: 40px;  
  
    position: fixed;  
  
    top: 20px;  
  
    right: 20px;  
  
    padding: 10px;  
  
    background-color: hsl(276, 89%, 76%);  
  
    /* Light gray background */  
  
    color: #333333;  
  
    /* Dark gray text */  
  
    border: 2px solid #cccccc;  
  
    /* Light gray border */  
  
    border-radius: 5px;  
  
    font-size: 16px;  
  
    z-index: 9999;  
  
    font-family: 'Arial', sans-serif;  
  
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);  
  
    cursor: pointer;  
  
}
```

```
.cert-image:hover {
```

```

background-color: hsl(276, 89%, 70%);

/* Darker gray background on hover */

color: #ffffff;

/* White text on hover */

border-color: #ffffff;

/* White border on hover */

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);

/* Increase box-shadow for a darker effect */

transition: background-color 0.3s ease-in-out, color 0.3s ease-in-out, border-color
0.3s ease-in-out, box-shadow 0.3s ease-in-out;

}

.cert-image input[type="file"]::-webkit-file-upload-button {

    visibility: hidden;

}

.template-options {

    display: grid;

    grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));

    /* Adjust item width as needed */

    gap: 20px;

    justify-content: center;

    margin-top: 20px;

}

.template-radio {

    display: none;

```

```
}
```

```
.template-label {  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  cursor: pointer;  
}
```

```
.template-label img {  
  width: 100%;  
  
  /* Image takes full width of its container */  
  
  height: auto;  
  
  transition: transform 0.3s ease-in-out, filter 0.3s ease-in-out;  
}
```

```
.template-label:hover img {  
  transform: scale(1.045);  
}
```

```
.template-radio:checked+.template-label {  
  border: 5px solid #146315;  
  
  box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);  
  
  transition: border-color 0.3s, background-color 0.3s, box-shadow 0.3s;  
}
```

```
.template-radio:checked+.template-label img {  
    filter: grayscale(0%);  
}
```

```
.button-container {  
    margin: 30px;  
    text-align: center;  
}
```

```
.generate-single-button,  
.generate-multiple-button {  
    margin: 30px;  
    width: 300px;  
    height: 40px;  
    padding: 10px 20px;  
    background-color: #28a745;  
    color: white;  
    border: none;  
    border-radius: 5px;  
    cursor: pointer;  
    transition: background-color 0.3s ease;  
}
```

```
.generate-single-button:hover,
```

```
.generate-multiple-button:hover {  
    background-color: #62b765;  
}
```

```
.singlecert-body,  
.bulkcert-body {  
    margin: 30px;  
    padding: 20px;  
    background-color: #a773d1;  
    border-radius: 10px;  
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);  
}
```

```
.form-container {  
    max-width: 500px;  
    margin: 50px auto;  
    padding: 30px;  
    background-color: #fff;  
    border-radius: 10px;  
    box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);  
}
```

```
.form-container h2 {  
    text-align: center;  
    margin-bottom: 30px;
```

```
color: #333;  
}
```

```
.form-group {  
    margin-bottom: 20px;  
}
```

```
label {  
    display: block;  
    margin-bottom: 5px;  
    color: #666;  
}
```

```
input[type="text"],  
input[type="date"],  
input[type="email"],  
textarea,  
input[type="number"] {  
    width: 100%;  
    padding: 10px;  
    border: 1px solid #ccc;  
    border-radius: 5px;  
    box-sizing: border-box;  
}
```



```
.submit-button,  
.generate-button {  
    margin-top: 10px;  
    background-color: #28a745;  
    color: #fff;  
    padding: 10px 20px;  
    border: none;  
    border-radius: 5px;  
    cursor: pointer;  
    width: 50%;  
}
```

```
.submit-button:hover,  
.generate-button:hover {  
    background-color: #218838;  
}
```

```
/* generate button in templalist page styling -- start*/
```

```
.button-container {  
    display: flex;  
    justify-content: center;  
    align-items: center;  
}
```

```
.generate-button {
```

```
position: relative;

padding: 20px 35px;

background: #ffffff;

font-size: 20px;

font-weight: 600;

color: #000000;

box-shadow: 0px 0px 10px 0px rgb(255, 255, 255);

border-radius: 100px;

border: none;

transition: all 0.3s ease-in-out;

cursor: pointer;

}
```

```
.star-1, .star-2, .star-3, .star-4, .star-5, .star-6, .star-7, .star-8, .star-9, .star-10, .star-11, .star-12 {

position: absolute;

filter: drop-shadow(0 0 0 #ffdfdf);

z-index: -5;

transition: all 0.8s cubic-bezier(0, 0.4, 0, 1.01);

}
```

```
.star-1 { top: 20%; left: 20%; width: 25px; }

.star-2 { top: 45%; left: 45%; width: 15px; }

.star-3 { top: 40%; left: 40%; width: 5px; }

.star-4 { top: 20%; left: 40%; width: 8px; }

.star-5 { top: 25%; left: 45%; width: 15px; }
```

```
.star-6 { top: 5%; left: 50%; width: 5px; }

.star-7 { top: 50%; left: 30%; width: 20px; }

.star-8 { top: 60%; left: 60%; width: 10px; }

.star-9 { top: 10%; left: 70%; width: 5px; }

.star-10 { top: 70%; left: 20%; width: 15px; }

.star-11 { top: 80%; left: 40%; width: 10px; }

.star-12 { top: 30%; left: 80%; width: 12px; }
```

```
.generate-button:hover {

    background: #000000;

    color: #ffffff;

    box-shadow: 0 0 80px #ffffff8c;

}
```

```
.generate-button:hover .star-1 { top: -20%; left: -20%; width: 20px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }
```

```
.generate-button:hover .star-2 { top: 35%; left: -25%; width: 15px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }
```

```
.generate-button:hover .star-3 { top: 80%; left: -10%; width: 10px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }
```

```
.generate-button:hover .star-4 { top: -25%; left: 105%; width: 20px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }
```

```
.generate-button:hover .star-5 { top: 30%; left: 115%; width: 15px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }
```

```
.generate-button:hover .star-6 { top: 80%; left: 105%; width: 10px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }
```

```
.generate-button:hover .star-7 { top: -10%; left: 20%; width: 20px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }
```

```

.generate-button:hover .star-8 { top: 20%; left: 75%; width: 10px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-9 { top: 5%; left: 95%; width: 5px; filter: drop-shadow(0
0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-10 { top: 75%; left: -5%; width: 15px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-11 { top: 105%; left: 40%; width: 10px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-12 { top: 45%; left: 120%; width: 12px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.fil0 {

    fill: #fffdef;

}

/* generate button in templaelist page styling -- end */

```

Templatelist.js

```

import React, { useState } from 'react';

import './templatelist.css';

function ListOfTemplates() {

    const [selectedOption, setSelectedOption] = useState(null);

    const handleFileChange = (event) => {

        setSelectedOption({ type: 'file', value: event.target.files[0] });
    }
}

```

```
};
```

```
const handleTemplateChange = (templateId) => {  
  setSelectedOption({ type: 'template', value: templateId });  
};
```

```
const generateCertificate = () => {  
  if (selectedOption) {  
    const url = selectedOption.type === 'file'  
      ?  
      `~/certgen?type=file&file_name=${encodeURIComponent(selectedOption.value.name  
      )}`  
      : `~/certgen?type=template&template_id=${selectedOption.value}`;  
  
    // Redirect to the next page with the selected option  
    window.location.href = url;  
  } else {  
    alert("Please select a file or a template first.");  
  }  
};
```

```
return (  
  <div className='template-list'>  
    <h1>Choose a Template</h1>  
  
    <div className='cert-image'>
```

```

<label htmlFor='user-temp' style={{ cursor: 'pointer', color: 'black' }}>

  Upload Certificate Image

</label>

<input

  type='file'

  id='user-temp'

  onChange={handleFileChange}

  accept='image/*'

  disabled={selectedOption && selectedOption.type === 'template'}

/>

</div>

<div className="template-options">

  {[1, 2, 3, 4, 5].map(templateId => (

    <div key={templateId}>

      <input

        type='radio'

        id={`temp${templateId}`}

        name='template'

        className="template-radio"

        onChange={() => handleTemplateChange(templateId)}

        disabled={selectedOption && selectedOption.type === 'file'}

      />

      <label htmlFor={`temp${templateId}`} className="template-label">

        <img

```

```

src={`https://i.pinimg.com/474x/${
  templateId === 1 ? '5d/b7/97/5db797d2bff4e37422a5f1aa439b4a58.jpg'
  : templateId === 2 ? 'f3/91/2c/f3912cfafe632fa92f36850077a94613.jpg'
  : templateId === 3 ? 'e6/6e/06/e66e063aa9e835abb01e81eb8fe7f5ad.jpg'
  : templateId === 4 ? 'b9/24/01/b92401d7d7851ea4ee59c6aaed11c5c5.jpg'
  : '71/16/2c/71162cc15ffb53edacd50a99751cd7b1.jpg'
}`}
alt={`Template ${templateId}`}
/>
</label>
</div>
)}}
</div>
<div className="button-container">
<button className="generate-button" onClick={generateCertificate}>
Generate Now
{[...Array(12)].map((_, i) => (
<div key={i} className={`star-${i + 1}`}>
<svg
xmlns="http://www.w3.org/2000/svg"
xmlnsXlink="http://www.w3.org/1999/xlink"
viewBox="0 0 784.11 815.53"
style={{
  shapeRendering: 'geometricPrecision',
  textRendering: 'geometricPrecision',

```

```

        imageRendering: 'optimizeQuality',

        fillRule: 'evenodd',

        clipRule: 'evenodd',

    }}

    version="1.1"

    xmlSpace="preserve"

>

    <g id="Layer_x0020_1">

        <metadata id="CorelCorpID_0Corel-Layer"></metadata>

        <path

            d="M392.05 0c-20.9,210.08 -184.06,378.41 -392.05,407.78 207.96,29.37
371.12,197.68 392.05,407.74 20.93,-210.06 184.09,-378.37 392.05,-407.74 -207.98,-
29.38 -371.16,-197.69 -392.06,-407.78z"

            className="fil0"

        ></path>

    </g>

</svg>

</div>

)}}

</button>

</div>

</div>

);

}

```

```

export default ListOfTemplates;

```


templatelist.css

```
.template-list {  
    margin: 50px;  
    padding: 20px;  
    background-color: #e393c6;  
    border-radius: 10px;  
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);  
}  
  
h1 {  
    text-align: center;  
    font-family: 'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans  
Unicode', Geneva, Verdana, sans-serif;  
}  
  
.cert-image {  
    height: 40px;  
    position: fixed;  
    top: 20px;  
    right: 20px;  
    padding: 10px;  
    background-color: hsl(276, 89%, 76%);  
    color: #333333;  
    border: 2px solid #cccccc;  
    border-radius: 5px;
```

```
font-size: 16px;

z-index: 9999;

font-family: 'Arial', sans-serif;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

cursor: pointer;

transition: background-color 0.3s ease-in-out, color 0.3s ease-in-out, border-color
0.3s ease-in-out, box-shadow 0.3s ease-in-out;

}
```

```
.cert-image:hover {

background-color: hsl(276, 89%, 70%);

color: #ffffff;

border-color: #ffffff;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);

}
```

```
.cert-image input[type="file"]::-webkit-file-upload-button {

visibility: hidden;

}
```

```
.template-options {

display: grid;

grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));

gap: 20px;

justify-content: center;

margin-top: 20px;
```

```
}
```

```
.template-radio {  
  display: none;  
}
```

```
.template-label {  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  cursor: pointer;  
}
```

```
.template-label img {  
  width: 100%;  
  height: auto;  
  transition: transform 0.3s ease-in-out, filter 0.3s ease-in-out;  
}
```

```
.template-label:hover img {  
  transform: scale(1.045);  
}
```

```
.template-radio:checked + .template-label {  
  border: 5px solid #146315;
```

```
box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);

transition: border-color 0.3s, background-color 0.3s, box-shadow 0.3s;

}
```

```
.template-radio:checked + .template-label img {

  filter: grayscale(0%);

}
```

```
.button-container {

  display: flex;

  justify-content: center;

  align-items: center;

  margin: 30px;

  text-align: center;

}
```

```
.generate-button {

  position: relative;

  padding: 20px 35px;

  background: #ffffff;

  font-size: 20px;

  font-weight: 600;

  color: #000000;

  box-shadow: 0px 0px 10px 0px rgb(255, 255, 255);

  border-radius: 100px;
```

```
border: none;

transition: all 0.3s ease-in-out;

cursor: pointer;

}
```

```
.generate-button:hover {

background: #000000;

color: #ffffff;

box-shadow: 0 0 80px #ffffff8c;

}
```

```
.star-1, .star-2, .star-3, .star-4, .star-5, .star-6, .star-7, .star-8, .star-9, .star-10, .star-11, .star-12 {

position: absolute;

filter: drop-shadow(0 0 0 #ffdef);

z-index: -5;

transition: all 0.8s cubic-bezier(0, 0.4, 0, 1.01);

}
```

```
.star-1 { top: 20%; left: 20%; width: 25px; }

.star-2 { top: 45%; left: 45%; width: 15px; }

.star-3 { top: 40%; left: 40%; width: 5px; }

.star-4 { top: 20%; left: 40%; width: 8px; }

.star-5 { top: 25%; left: 45%; width: 15px; }

.star-6 { top: 5%; left: 50%; width: 5px; }

.star-7 { top: 50%; left: 30%; width: 20px; }
```

```

.star-8 { top: 60%; left: 60%; width: 10px; }

.star-9 { top: 10%; left: 70%; width: 5px; }

.star-10 { top: 70%; left: 20%; width: 15px; }

.star-11 { top: 80%; left: 40%; width: 10px; }

.star-12 { top: 30%; left: 80%; width: 12px; }


.generate-button:hover .star-1 { top: -20%; left: -20%; width: 20px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-2 { top: 35%; left: -25%; width: 15px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-3 { top: 80%; left: -10%; width: 10px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-4 { top: -25%; left: 105%; width: 20px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-5 { top: 30%; left: 115%; width: 15px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-6 { top: 80%; left: 105%; width: 10px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-7 { top: -10%; left: 20%; width: 20px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-8 { top: 20%; left: 75%; width: 10px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-9 { top: 5%; left: 95%; width: 5px; filter: drop-shadow(0
0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-10 { top: 75%; left: -5%; width: 15px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-11 { top: 105%; left: 40%; width: 10px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

.generate-button:hover .star-12 { top: 45%; left: 120%; width: 12px; filter: drop-
shadow(0 0 10px #fffdef); z-index: 2; }

```

```
.fil0 {  
  
  fill: #fffdef;  
  
}
```

Certgen.js

```
import React, { useState } from 'react';  
  
import axios from 'axios';  
  
import './certgen.css'; // Make sure to import your CSS file  
  
const CertGen = () => {  
  
  const [certificateType, setCertificateType] = useState("");  
  
  const [heading, setHeading] = useState("");  
  
  const [certificateAbout, setCertificateAbout] = useState("");  
  
  const [certificantName, setCertificantName] = useState("");  
  
  const [date, setDate] = useState("");  
  
  const [companyName, setCompanyName] = useState("");  
  
  const [certificateProviderName, setCertificateProviderName] = useState("");  
  
  const [usernames, setUsernames] = useState(null);  
  
  const [email, setEmail] = useState("");  
  
  const [numberOfCerts, setNumberOfCerts] = useState("");  
  
  const [errorMessage, setErrorMessage] = useState("");  
  
  
  const handleCertificateTypeChange = (event) => {  
  
    setCertificateType(event.target.value);  
  
    setErrorMessage("");  
  
  };
```

```

const submitFormData = async (event) => {

  event.preventDefault();

  if (!certificateType) {

    setErrorMessage('Please choose a certificate type. ');

    return;

  }

  try {

    if (certificateType === 'single') {

      let formfield = new FormData();

      formfield.append('heading', heading);

      formfield.append('certificate_about', certificateAbout);

      formfield.append('certificant_name', certificantName);

      formfield.append('issue_date', date);

      formfield.append('company_name', companyName);

      formfield.append('certificate_provider_name', certificateProviderName);

      const response = await axios.post('http://localhost:8000/api/certificates/',
formfield);

      const enteredData = response.data;

      alert('Form submission successful');

      sessionStorage.setItem('submittedId', enteredData.id); // Store submitted ID in
sessionStorage
    }
  }
}

```



```

    sessionStorage.setItem('formType', 'single');

    window.location.href = '/formresults';

} else if (certificateType === 'bulk') {

    let bulkformfield = new FormData();


    bulkformfield.append('heading', heading);

    bulkformfield.append('certificate_about', certificateAbout);

    bulkformfield.append('usernames', usernames);

    bulkformfield.append('issue_date', date);

    bulkformfield.append('company_name', companyName);

    bulkformfield.append('certificate_provider_name', certificateProviderName);

    bulkformfield.append('email', email);

    bulkformfield.append('number_of_certificates', numberOfCerts);


    const response = await axios.post('http://localhost:8000/api/bulkcertificates/',
bulkformfield);

    const bulkFormData = response.data;

    alert('Form submission successful');

    sessionStorage.setItem('submittedIdBulk', bulkFormData.id);

    sessionStorage.setItem('formType', 'bulk');

    window.location.href = '/formresults';

}

} catch (error) {

    alert('Form submission failed: ' + error.message);

}

};

```

```

return (

  <div>

    <div className='headingcolor' id='heading1' style={{ display: certificateType ?
'none' : 'block' }}>

      <h1>Choose Certificate Type</h1>

    </div>

    <div className='headingcolor' id='heading2' style={{ display: certificateType ?
'block' : 'none' }}>

      <h1> Please Provide The Necessary Details</h1>

    </div>


    <div className='pagetop-centre'>

      <div className="container">

        <div className="pane">

          <label className="label">

            <span>Single</span>

            <input

              type="radio"

              className="input"

              name="certificateType"

              value="single"

              checked={certificateType === 'single'}

              onChange={handleCertificateTypeChange}

            />

          </label>

```

```

<label className="label">

  <span>Bulk</span>

  <input

    type="radio"

    className="input"

    name="certificateType"

    value="bulk"

    checked={certificateType === 'bulk'}

    onChange={handleCertificateTypeChange}

  />

</label>

<span className="selection"></span>

</div>

</div>

</div>

{certificateType && (

  <div className="form-container">

    <form className="form" onSubmit={submitFormData}>

      {certificateType === 'single' && (

        <>

          <div className="form-group">

            <label htmlFor="heading">Heading:</label>

            <input type="text" value={heading} onChange={(e) =>
setHeading(e.target.value)} />

          </div>

```

```

    <div className="form-group">

      <label htmlFor='certificate_about'>Certificate About:</label>

      <input type="text" value={ certificateAbout } onChange={ (e) =>
setCertificateAbout(e.target.value)} />

    </div>

```

```

    <div className="form-group">

      <label htmlFor="certificant_name">Certificant Name:</label>

      <input type="text" value={ certificantName } onChange={ (e) =>
setCertificantName(e.target.value)} />

    </div>

  </>

)}

```

```

{certificateType === 'bulk' && (

  <>

    <div className="form-group">

      <label htmlFor="heading">Heading:</label>

      <input type="text" value={ heading } onChange={ (e) =>
setHeading(e.target.value)} />

    </div>

```

```

    <div className="form-group">

      <label htmlFor='certificate_about'>Certificate About:</label>

      <input type="text" value={ certificateAbout } onChange={ (e) =>
setCertificateAbout(e.target.value)} />

```

```
</div>
```

```
<div className="form-group">
```

```
<label htmlFor="usernames">Upload Usernames File</label>
```

```
<input
```

```
  type="file"
```

```
  id="usernames"
```

```
  accept=".csv, .json"
```

```
  onChange={(e) => setUsername(e.target.files[0])}
```

```
</div>
```

```
<div className="form-group">
```

```
<label htmlFor="email">Enter Email to Get Your Certificates:</label>
```

```
<input type="email" id="email" placeholder="john@example.com"
value={email} onChange={(e) => setEmail(e.target.value)} />
```

```
</div>
```

```
<div className="form-group">
```

```
<label htmlFor="number_of_certs">Number of Certificates to
Generate:</label>
```

```
<input type="number" id="number_of_certs" placeholder="5"
value={numberOfCerts} onChange={(e) => setNumberOfCerts(e.target.value)} />
```

```
</div>
```

```
</>
```

```
)}
```

```

    <div className="form-group">

      <label htmlFor="date">Date:</label>

      <input type="date" value={date} onChange={(e) => setDate(e.target.value)}
/>

    </div>

    <div className="form-group">

      <label htmlFor="company_name">Company Name:</label>

      <input type="text" value={companyName} onChange={(e) =>
setCompanyName(e.target.value)} />

    </div>

    <div className="form-group">

      <label htmlFor="certificate_provider_name">Certificate Provider
Name:</label>

      <input type="text" value={certificateProviderName} onChange={(e) =>
setCertificateProviderName(e.target.value)} />

    </div>

    {errorMessage && <p style={{ color: 'red' }}>{errorMessage}</p>}

    <button type="submit" className="form-submit-btn">

      Submit

    </button>

  </form>

</div>

)}

```

```

    </div>

);

};

export default CertGen;

```

Certgen.css

```

/* General body styling */

body {

    font-family: Arial, sans-serif;

    background: linear-gradient(135deg, #222222, #333333);

    margin: 0;

    padding: 0;

}

/* Heading styling */

.headingcolor {

    color: white;

    font-family: 'Lucida Sans', Geneva, Verdana, sans-serif;

    text-align: center;

    margin: 20px 0;

}

/* Center alignment for the radio button selection container */

.pagetop-centre {

```

```

display: flex;

align-items: center;

justify-content: center;

margin-top: 20px;
}

/* Container for radio button selection */

.container {

    position: relative;

    display: flex;

    justify-content: center;

}

/* Radio buttons styling */

.pane {

    outline: 2px solid #00ff6a;

    box-shadow: 0 0 10px #00ff6a77, inset 0 0 10px #00ff6a77;

    height: 40px;

    width: 120px;

    border-radius: 5px;

    position: relative;

    overflow: hidden;

    transition: 0.7s ease;

}

```



```
.input {  
    display: none;  
}
```

```
.label {  
    height: 40px;  
    width: 60px;  
    float: left;  
    font-weight: 600;  
    font-size: 14px;  
    position: relative;  
    z-index: 1;  
    color: #00ff6a;  
    text-align: center;  
    padding-top: 10px;  
    cursor: pointer;  
}
```

```
.selection {  
    display: none;  
    position: absolute;  
    height: 40px;  
    width: 60px;  
    z-index: 0;  
    left: 0;
```

```

top: 0;

box-shadow: 0 0 10px #00ff6a77;

transition: 0.15s ease;
}

.label:has(input:checked) {

    color: #212121;

}

.label:has(input:checked) ~ .selection {

    background-color: #00ff6a;

    display: inline-block;

}

.label:nth-child(1):has(input:checked) ~ .selection {

    transform: translateX(0);

}

.label:nth-child(2):has(input:checked) ~ .selection {

    transform: translateX(60px);

}

/* Form container styling */

.form-container {

    position: relative;

```

```
width: 100%;  
max-width: 500px;  
padding: 20px;  
background-color: #FFF;  
border-radius: 4px;  
color: #333;  
box-shadow: 0px 0px 60px 5px rgba(0, 0, 0, 0.4);  
margin: 20px auto;  
}
```

```
.form-container:after {  
  position: absolute;  
  content: "";  
  right: -10px;  
  bottom: 18px;  
  width: 0;  
  height: 0;  
  border-left: 0px solid transparent;  
  border-right: 10px solid transparent;  
  border-bottom: 10px solid #18bc96;  
}
```

```
.form-container h2 {  
  text-align: center;  
  font-size: 20px;
```

```
font-weight: bold;

letter-spacing: 4px;

line-height: 28px;

}
```

```
/* Form element styling */
```

```
.form-container label {

display: block;

margin: 10px 0 5px;

}
```

```
.form-container input {

display: block;

width: 100%;

padding: 10px;

margin-bottom: 20px;

border: none;

border-bottom: 1px solid #d4d4d4;

background: transparent;

transition: all .25s ease;

}
```

```
.form-container input:focus {

outline: none;

border-color: #18bc96;
```

```
border-style: solid;

border-width: 2px;

}
```

```
.form-group input[type="file"] {

display: block;

margin-top: 0.3rem;

}
```

```
.form-group input[type="file"] {

border: 1px solid #3b7d02; /* Green border color */

padding: 0.5rem;

background-color: #ffffff; /* White background for input */

}
```

```
/* Submit button styling */

.submit-button-wrapper {

display: flex;

justify-content: center;

}
```

```
/* Submit button styling */

.form-container .form-submit-btn {

--color: #560bad;
```

```
font-family: inherit;

display: inline-block;

width: 8em;

height: 2.6em;

line-height: 2.5em;

margin: 20px;

position: relative;

overflow: hidden;

border: 2px solid var(--color);

transition: color 0.5s;

z-index: 1;

font-size: 17px;

border-radius: 6px;

font-weight: 500;

color: var(--color);

}
```

```
.form-container .form-submit-btn:before {

content: "";

position: absolute;

z-index: -1;

background: var(--color);

height: 150px;

width: 200px;

border-radius: 50%;
```

```
}
```

```
.form-container .form-submit-btn:hover {  
  color: #fff;  
}
```

```
.form-container .form-submit-btn:before {  
  top: 100%;  
  left: 100%;  
  transition: all 0.7s;  
}
```

```
.form-container .form-submit-btn:hover:before {  
  top: -30px;  
  left: -30px;  
}
```

```
.form-container .form-submit-btn:active:before {  
  background: #3a0ca3;  
  transition: background 0s;  
}
```

Formresults.js

```
import React, { useState, useEffect } from 'react';
```

```
import './formresults.css'; // Assuming you have a CSS file for styling
```

```

export default function FormResults() {

  const [submittedId, setSubmittedId] = useState("");

  const [formType, setFormType] = useState("");

  useEffect(() => {

    // Retrieve the submitted ID and form type from sessionStorage

    const idFromSessionStorage = sessionStorage.getItem('submittedId');

    const formTypeFromSessionStorage = sessionStorage.getItem('formType');

    if (idFromSessionStorage && formTypeFromSessionStorage) {

      setSubmittedId(idFromSessionStorage);

      setFormType(formTypeFromSessionStorage);

    }

  }, []);

  return (

    <div className="form-results-container">

      <h1>Form Results</h1>

      { /* Display form submission results based on form type */ }

      { submittedId && (

        <div className="results-section">

          <h2>{ formType === 'single' ? 'Single Form Submission Result' : 'Bulk
Form Submission Result' }</h2>

          <p>Submitted ID: { submittedId }</p>

          { formType === 'bulk' && (

```



```

        <button className="download-button">Download Zip File</button>

    )}

    {formType === 'single' && (

        <button className="download-button">Download Certificate</button>

    )}

</div>

)}

{ /* Placeholder if no form submission results are found */}

{ !submittedId && (

    <div className="no-results">

        <p>No form submission results found.</p>

    </div>

    )}

</div>

);

}

```

Formresults.css

```

.form-results-container {

    max-width: 600px;

    margin: 50px auto;

    padding: 20px;

    border: 1px solid #6dbb63; /* Greenish border color */

```

```
border-radius: 8px;

background-color: #edf7ed; /* Light greenish background */

text-align: center;

box-shadow: 0 0 20px rgba(0, 0, 0, 0.1);

}
```

```
.results-section {

margin-bottom: 20px;

padding: 20px;

border: 1px solid #6dbb63; /* Greenish border color */

border-radius: 8px;

background-color: #ffffff; /* White background */

box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

text-align: left;

}
```

```
.results-section h2 {

font-size: 1.8rem;

margin-bottom: 15px;

color: #6dbb63; /* Greenish text color */

}
```

```
.results-section p {

font-size: 1.4rem;

color: #333333; /* Dark text color */

}
```

```

margin-bottom: 10px;
}

.download-button {
  display: inline-block;
  padding: 12px 24px;
  background-color: #6dbb63; /* Greenish button background */
  color: #ffffff;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-size: 1.4rem;
  transition: background-color 0.3s ease;
  text-decoration: none;
}

.download-button:hover {
  background-color: #4c934a; /* Darker greenish color on hover */
}

```

14. Testing Approach

Throughout the certificate generation project, I implemented a structured testing approach for the UI, API, backend database, and session management:

1. UI Testing: I conducted manual testing of the React frontend to ensure all components functioned as intended. This included checking user interactions, validating form submissions, and verifying the responsiveness of the interface across

different devices and browsers. I also utilized browser developer tools for debugging and checking for layout issues.

2. API Testing: I employed tools like Postman to test the Django REST APIs. This involved sending requests to various endpoints, verifying the responses, and ensuring that the API correctly handled different scenarios, such as successful data retrieval and error responses. I also wrote automated tests using Django's testing framework to validate the API's functionality.

3. Backend Database Testing: To ensure data integrity and correctness, I performed database testing by directly querying the MySQL database. I verified that data was being stored, retrieved, and updated as expected. Additionally, I wrote unit tests to check the interaction between the Django models and the database, ensuring that the data layer was functioning correctly.

4. Session Management Testing: I tested session management functionality by validating user authentication and session persistence. This included checking that user sessions were correctly initiated, maintained, and terminated, ensuring that data was stored in sessionStorage as expected. I also verified that session-related errors were handled appropriately, providing a smooth user experience.

This comprehensive testing strategy helped ensure the reliability and functionality of the entire application throughout the development process.

15. Testing :

In the development of the certificate generation project, thorough manual testing was conducted to ensure a seamless and intuitive user experience. The testing process was integral from the beginning of the project and continued throughout development, focusing on various aspects of the UI and functionality.

1. Responsiveness and Effectiveness of the UI

One of the primary goals was to create a responsive and effective user interface. This involved checking how the UI behaved across different devices and screen sizes:

- **Responsive Design:** Ensured that the UI elements adjusted correctly to different screen sizes, including desktops, tablets, and mobile devices. Used browser developer tools to simulate various screen resolutions and tested UI components manually.
- **User-Friendly Interface:** Focused on making the interface intuitive and easy to navigate. This involved minimizing the number of clicks required to generate certificates and ensuring that all elements were accessible and easy to understand.

2. Testing Each Component and Page

The testing process involved questioning the functionality and design of every component and page built:

- **Component Functionality:** Verified that each React component behaved as expected. This included checking props, state management, and the rendering of dynamic content.
- **Alternative Solutions:** Continuously searched for more efficient ways to achieve the same functionality with fewer lines of code. This iterative process helped in optimizing the codebase and improving performance.

3. Single Page Reload and React Features

React's special feature of single-page reloads was also tested extensively:

- **Single Page Reload:** Ensured that navigation between different routes (pages) was smooth without full-page reloads. This was crucial for maintaining the state and providing a seamless user experience.

- **Component Updates:** Verified that components updated correctly without unnecessary re-renders. Used React DevTools to inspect the component tree and identify any potential performance bottlenecks.

4. Login and Signup Pages

The login and signup functionalities were crucial for user authentication and authorization:

- **Form Validation:** Tested the form inputs for various edge cases, including empty fields, invalid email formats, and password requirements. Ensured that appropriate error messages were displayed.
- **User Authentication:** Verified that the login and signup processes worked correctly, including handling successful and failed authentication attempts.
- **Session Management:** Ensured that user sessions were managed properly, including storing user information in session storage and handling user state across different routes.

5. Overall Testing Approach

The overall testing approach was systematic and thorough:

- **Continuous Testing:** Testing was an ongoing process, starting from the initial development stages and continuing throughout the project. Each new feature or component was tested as it was developed.
- **Real-World Scenarios:** Tested the application using real-world scenarios and user behaviors. This involved simulating user actions,

such as filling out forms, navigating between pages, and generating certificates.

- **Iterative Improvement:** Based on the testing results, continuously iterated on the design and functionality. Made necessary adjustments to enhance the user experience and fix any issues that were identified.

16. Test Plan

1. Introduction

- **Objective:** To ensure the React application meets the specified requirements and functions correctly.
- **Scope:** This test plan covers all functional and non-functional aspects of the React application, including component rendering, state management, props handling, user interactions, API calls, routing, and session management.

2. Test Items

- Components
- State management
- Props passing
- User interactions
- API calls and data handling
- Routing and navigation
- Session management

3. Features to be Tested

- Component rendering
- Props and state handling
- User authentication
- Form validations
- Data fetching and displaying
- Routing and navigation flow
- Session management

5. Test Approach

- **Unit Testing:** Testing individual components using Jest and React Testing Library.
- **Integration Testing:** Testing combined components and their interactions.
- **End-to-End (E2E) Testing:** Testing the entire application flow using Cypress or Selenium.

6. Pass/Fail Criteria

- All test cases must pass without errors.
- Any critical issues must be resolved before release.

17. Test Cases

Component Rendering

Test Case 1: Component Renders Correctly

- **Objective:** To Verify that the component renders without errors.

- **Steps:**
 1. Import and render the component.
 2. Check if the component appears in the DOM.
- **Expected Result:** The component should render correctly.
- **Actual Result:** component rendered successfully.
- **Status:** Pass

Props and State Handling

Test Case 2: Component Receives Props

- **Objective:** To Verify that the component receives and uses props correctly.
- **Steps:**
 1. Pass props to the component.
 2. Check if the component displays data based on props.
- **Expected Result:** The component should display data according to the props passed.
- **Actual Result:** The component displayed data accordingly.
- **Status:** Pass

Test Case 3: State Updates Correctly

- **Objective:** To Verify that the component's state updates as expected.
- **Steps:**
 1. Trigger a state change (e.g., button click).
 2. Check if the state updates correctly.

- **Expected Result:** The component's state should update as expected.
- **Actual Result:** component changed its state accordingly.
- **Status:** Pass

User Interactions

Test Case 4: Form Submission

- **Objective:** To Verify that the form submits data correctly.
- **Preconditions:** Form should be present on the page.
- **Steps:**
 1. Fill out the form fields.
 2. Submit the form.
 3. Check if the form submission is handled correctly.
- **Expected Result:** The form should submit and process data correctly.
- **Actual Result:** Correct data is displayed .
- **Status:** Pass

API Calls and Data Handling

Test Case 5: Data Fetching

- **Objective:** To Verify that the component fetches data from an API correctly.
- **Preconditions:** API endpoint should be available.
- **Steps:**

1. Mock the API response.
 2. Render the component.
 3. Check if the component displays the fetched data.
- **Expected Result:** The component should display the data fetched from the API.
 - **Actual Result:** component displayed expected data.
 - **Status:** Pass

Routing and Navigation

Test Case 6: Navigation to a Different Page

- **Objective:** To Verify that navigation works correctly.
- **Preconditions:** Navigation links should be present.
- **Steps:**
 1. Click a navigation link.
 2. Check if the user is redirected to the correct page.
- **Expected Result:** The user should be redirected to the correct page.
- **Actual Result:** User redirected to correct page(form results)
- **Status:** Pass

Session Management

Test Case 7: User Authentication

- **Objective:** To Verify that user authentication works correctly.
- **Preconditions:** Valid user credentials should be available.
- **Steps:**
 1. Attempt to log in with valid credentials.

2. Check if the user is authenticated and redirected.

- **Expected Result:** The user should be authenticated and redirected correctly.
- **Actual Result:** Authentication and redirection success. Redirected to Templatelist .
- **Status:** Pass

18.Bug List

CertGen.js

Issue: Handling of templateImage retrieval lacks robustness when templateId is absent.

Impact: Potential for runtime errors or unexpected behavior when templateId is not defined.

Signup.js and Login.js

Issue: Inconsistent sessionStorage key usage (user vs email) after successful signup and login operations.

Impact: Inconsistencies in user session management and potential confusion in data retrieval across components.

Recommendations

Consistency in sessionStorage Keys:

- Ensure uniform usage of sessionStorage keys (user, userStatus) throughout the application to maintain clarity and reliability in user session management.

Enhanced Error Handling:

- Implement comprehensive error handling strategies, particularly in asynchronous operations like network requests (axios), to manage and communicate errors effectively to users.

Testing and Validation:

- Conduct thorough testing, including edge cases such as empty form submissions and network failures, to validate application behavior and ensure robustness in real-world scenarios.

19 . Agile Methodology in the Certificate Generation Project

In the development of the certificate generation project, we utilized Agile methodology to ensure a flexible, collaborative, and efficient approach. Agile enabled us to quickly adapt to changes and deliver incremental improvements, ensuring the project met user needs and maintained high quality.

1. Introduction to Agile Methodology

Agile methodology is a project management and software development approach that emphasizes iterative progress, collaboration, and flexibility. Unlike traditional methodologies, Agile focuses on delivering small,

functional pieces of the project frequently, enabling continuous feedback and adjustments.

2. Key Principles of Agile Methodology

- **Iterative Development:** Work is divided into small, manageable iterations or sprints, usually lasting two to four weeks. Each iteration results in a potentially shippable product increment.
- **Customer Collaboration:** Regular interaction with stakeholders to gather feedback and make necessary adjustments.
- **Responding to Change:** Flexibility to adapt to changing requirements, even late in the development process.
- **Cross-Functional Teams:** Collaboration among team members with different expertise to ensure comprehensive development and testing.

3. Agile Process in the Certificate Generation Project

The following steps outline how we implemented Agile methodology in the certificate generation project:

1. Sprint Planning

- **Sprint Planning Meetings:** At the beginning of each sprint, we held planning meetings to define the goals and tasks for the upcoming iteration. We prioritized user stories and features based on their importance and complexity.
- **Task Breakdown:** Each user story was broken down into smaller, actionable tasks. This made it easier to estimate effort and assign tasks to team members.

2. Daily Status Updates

- **Daily Status Updates:** Instead of traditional stand-up meetings, we had daily status updates where each team member reported their progress to their respective reporting manager. This facilitated communication, helped identify blockers, and ensured alignment among team members.

3. Iterative Development and Testing

- **Development:** Each team member focused on their assigned tasks, developing components and features iteratively. This included both frontend (React components, UI design) and backend (API integration, database management) work.
- **Continuous Testing:** Testing was an integral part of each iteration. As features were developed, they were tested manually for responsiveness, functionality, and performance. This ensured that any issues were identified and addressed early.

4. Sprint Review and Retrospective

- **Sprint Review:** At the end of each sprint, we held a review meeting to demonstrate the completed work to stakeholders. Feedback was gathered and used to refine future iterations.
- **Retrospective Meeting:** Conducted retrospective meetings to reflect on the sprint. The team discussed what went well, what could be improved, and any actions needed to enhance future sprints.

5. Continuous Improvement

- **Adaptation and Flexibility:** Based on feedback from reviews and retrospectives, we adapted our approach and made necessary

changes to improve the process. This included refining user stories, adjusting priorities, and improving communication.

- **Incremental Delivery:** By delivering functional increments at the end of each sprint, we ensured continuous progress and allowed stakeholders to see tangible results. This incremental approach helped in building a reliable and user-friendly application.

20. Enhancements and Future Work for the Certificate Generation Project

1. Enhanced User Interface

Future Work: Improve the user interface by incorporating modern design principles and user experience (UX) best practices. This includes enhancing visual elements, streamlining navigation, and making the interface more intuitive.

2. Automated Testing

Future Work: Implement automated testing frameworks to streamline the testing process. This would include unit tests, integration tests, and end-to-end tests to ensure that new features are thoroughly vetted without extensive manual testing.

3. Performance Optimization

Future Work: Further optimize performance by analyzing and improving database queries, implementing caching strategies, and exploring lazy loading techniques for large datasets. This will enhance response times and user experience.

4. User Role Management

Future Work: Develop a user role management system that allows different access levels for users. This would enable functionalities such as admin dashboards, enhanced reporting features, and controlled access to sensitive areas of the application.

5. Real-Time Notifications

Future Work: Integrate real-time notification features using WebSockets or similar technologies. This would allow users to receive updates on their certificate generation status instantly, improving engagement and user satisfaction.

6. Expanded Certificate Customization Options

Future Work: Allow users to customize certificate templates more extensively. This could include options for different styles, fonts, and layouts, enhancing personalization and appeal.

7. Mobile Application

Future Work: Consider developing a mobile application or a responsive web app to provide users with the ability to generate and manage certificates on-the-go, catering to a wider audience.

8. Integration with Third-Party Services

Future Work: Explore integrations with third-party services, such as payment gateways for premium features or cloud storage solutions for certificate storage and sharing.

9. Enhanced Analytics and Reporting

Future Work: Implement advanced analytics and reporting features that provide insights into user behavior, certificate usage, and system performance. This data can inform future improvements and marketing strategies.

21. Release Notes - Version 1.1.0

New Features

- Implemented a ProtectedRoute component to manage authentication status and redirect users based on their login status.
- Added a SessionDetails component to display user session information including email and status.
- Introduced a Logout component for securely logging users out of their sessions.

Enhancements

- Improved session management consistency by standardizing sessionStorage key usage (user, userStatus).
- Enhanced error handling for asynchronous operations, ensuring better user feedback and system reliability.
- Updated CertGen.js to handle templateImage retrieval more robustly, improving application stability.

Bug Fixes

- Fixed issues related to inconsistent sessionStorage key usage post-signup and login operations.

- Addressed potential bugs in CertGen.js for better handling of templateId absence scenarios.

Known Issues

- None reported at this time.

Next Steps

- Continue monitoring application performance and user feedback.
- Plan for upcoming features such as enhanced user profile management and additional form submission functionalities.

22. List of Abbreviations / Nomenclature :

The list of abbreviations and nomenclature used in the code base:

React Components and Hooks:

1. **JSX**: JavaScript XML
2. **DOM**: Document Object Model
3. **useState**: A React Hook for managing state in functional components
4. **useEffect**: A React Hook for managing side effects in functional components

Component Names:

1. **App**: Main application component
2. **CertGen**: Certificate Generation component
3. **Templatelist**: Template List component
4. **ListOfTemplates**: List of Templates component
5. **CertificateForm**: Certificate Form component
6. **FormResults**: Form Results component

Props and State:

1. **props**: Properties passed to components
2. **state**: Local state within a component
3. **setState**: Function to update the state
4. **selectedOption**: State variable for storing the selected option (file or template)

5. **certificateType**: State variable for storing the type of certificate (single or bulk)
6. **heading**: State variable for the certificate heading
7. **certificateAbout**: State variable for the certificate description
8. **certificantName**: State variable for the name of the person receiving the certificate
9. **date**: State variable for the date on the certificate
10. **companyName**: State variable for the company name on the certificate
11. **certificateProviderName**: State variable for the provider name on the certificate
12. **usernames**: State variable for the usernames file in bulk certificate generation
13. **email**: State variable for the email in bulk certificate generation
14. **numberOfCerts**: State variable for the number of certificates to generate
15. **errorMessage**: State variable for storing error messages
16. **submittedId**: State variable for storing the ID of the submitted form
17. **formType**: State variable for storing the type of form submitted

Routing and URLs:

1. **BrowserRouter**: Component for enabling routing using the HTML5 history API
2. **Route**: Component for defining a route
3. **Routes**: Component for grouping multiple Route components
4. **window.location.href**: Property for redirecting to a different URL

FormData and API:

1. **FormData**: Interface for creating a set of key/value pairs representing form fields and their values
2. **axios**: Promise-based HTTP client for making requests
3. **response**: Object containing the response from an HTTP request
4. **sessionStorage**: Web storage object for storing data that is accessible only within the current session
5. **API**: Application Programming Interface

Miscellaneous:

1. **CSV**: Comma-Separated Values (used for file type in bulk certificate generation)
2. **JSON**: JavaScript Object Notation (used for file type in bulk certificate generation)
3. **UI**: User Interface
4. **cert-image**: Class name for styling the certificate image upload section
5. **template-options**: Class name for styling the template options section
6. **generate-button**: Class name for styling the generate button
7. **form-container**: Class name for styling the form container

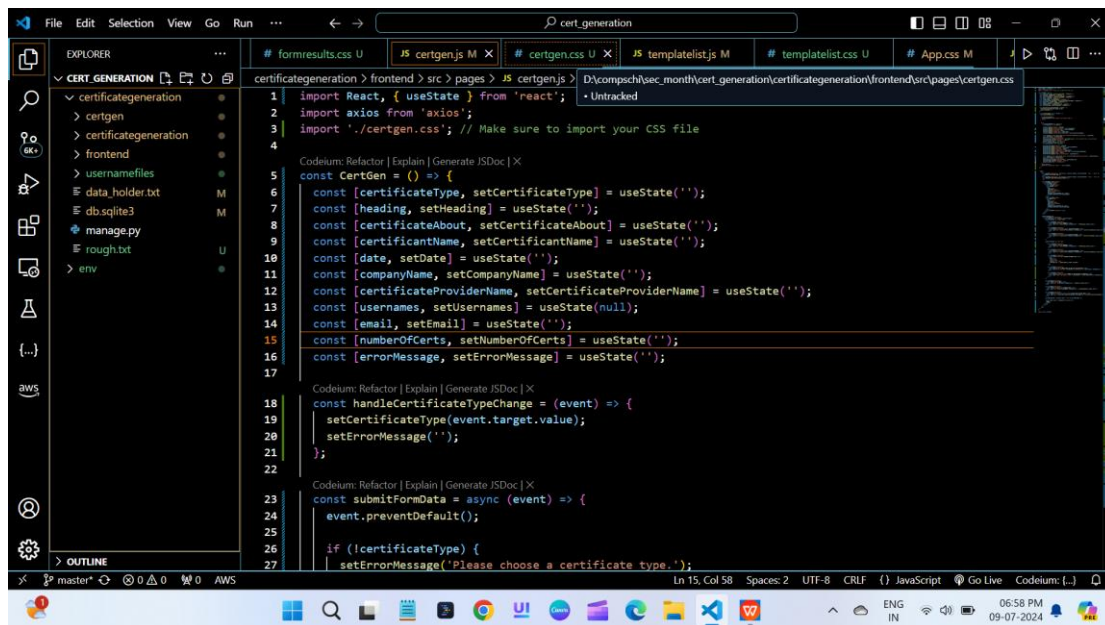
8. **form-group**: Class name for styling form groups
9. **form-submit-btn**: Class name for styling the form submit button
10. **form-results-container**: Class name for styling the form results container
11. **results-section**: Class name for styling the results section
12. **no-results**: Class name for styling the no results section
13. **download-button**: Class name for styling the download button

23. List Of Figures :

1. Code base
2. templates list page
3. Option choosing
4. Single certificate form
5. Single certificate success
6. Single certificate form results :
7. Bulk certificate :
8. Bulk certificate form submission success
9. Bulk certificate form submission results
10. Admin page
11. Bulk certificate Data
12. Single certificate forms Data
13. Groups
14. Api page
15. Single certificate results page in restframework
16. Bulk certificate results page in restframework
17. Testing templatelists
18. Testing single certificate form
19. Testing bulk certificate form

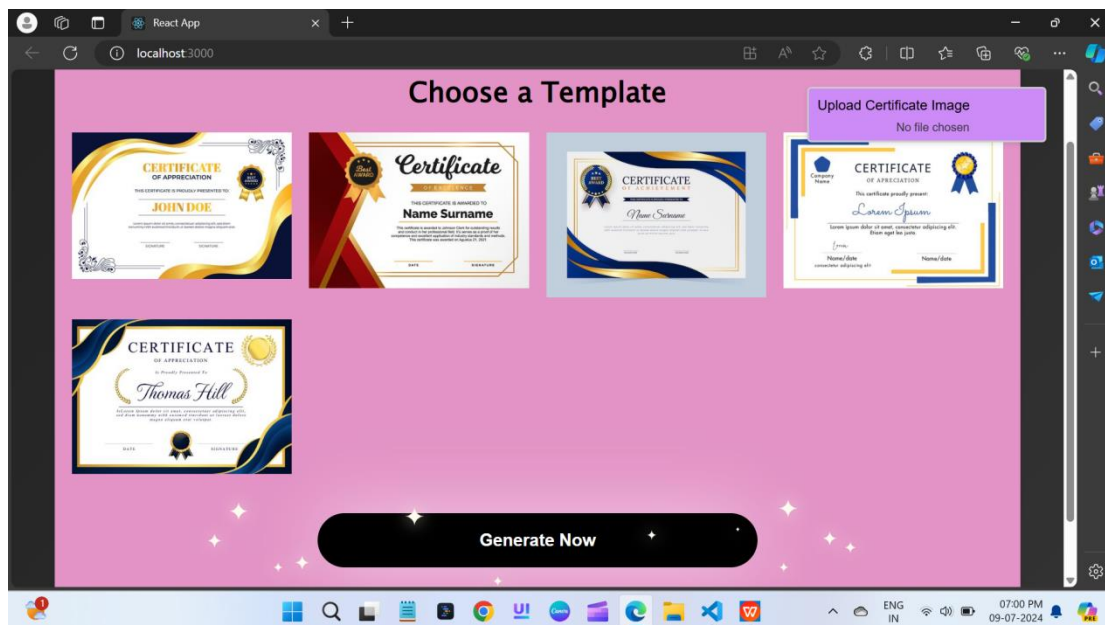
24. ScreenShots :

Code Base:

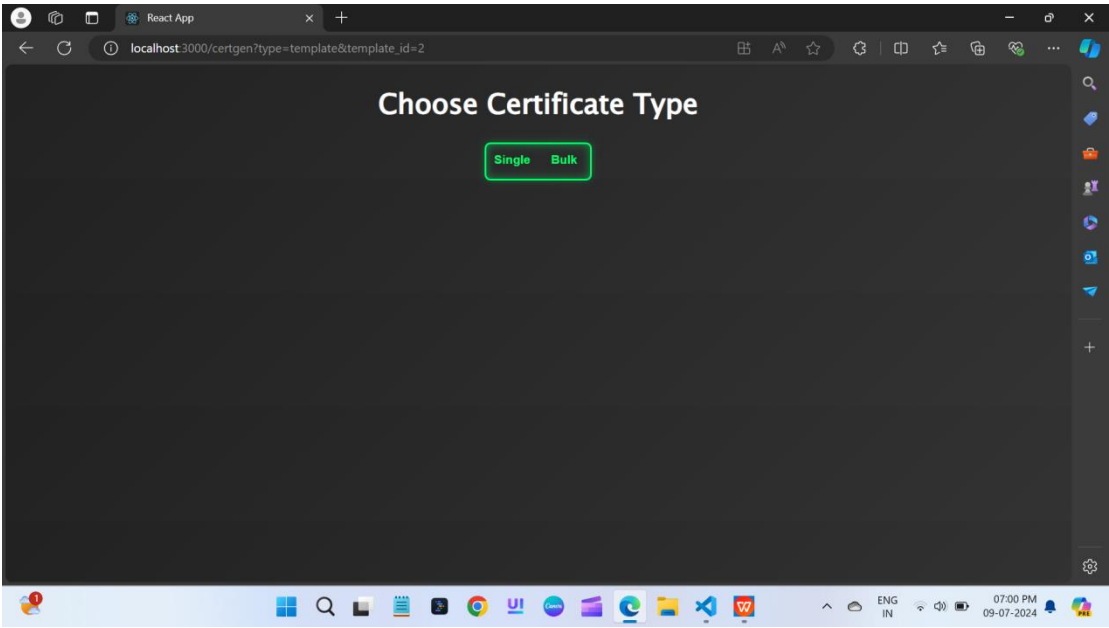


```
1 import React, { useState } from 'react';
2 import axios from 'axios';
3 import './certgen.css'; // Make sure to import your CSS file
4
5 Codeium: Refactor | Explain | Generate JSDoc | X
6 const CertGen = () => {
7   const [certificateType, setCertificateType] = useState('');
8   const [heading, setHeading] = useState('');
9   const [certificateAbout, setCertificateAbout] = useState('');
10  const [certificantName, setCertificantName] = useState('');
11  const [date, setDate] = useState('');
12  const [companyName, setCompanyName] = useState('');
13  const [certificateProviderName, setCertificateProviderName] = useState('');
14  const [usernames, setUsernames] = useState(null);
15  const [email, setEmail] = useState('');
16  const [numberOfCerts, setNumberOfCerts] = useState('');
17  const [errorMessage, setErrorMessage] = useState('');
18
19  Codeium: Refactor | Explain | Generate JSDoc | X
20  const handleCertificateTypeChange = (event) => {
21    setCertificateType(event.target.value);
22    setErrorMessage('');
23  };
24
25  Codeium: Refactor | Explain | Generate JSDoc | X
26  const submitFormData = async (event) => {
27    event.preventDefault();
28
29    if (!certificateType) {
30      setErrorMessage('Please choose a certificate type.');
```

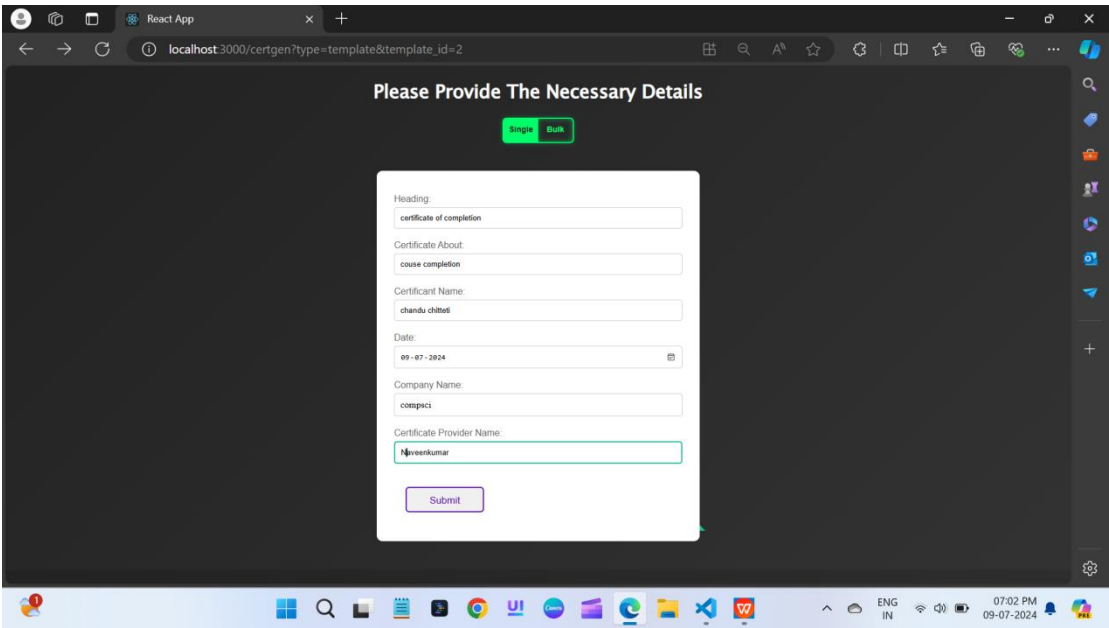
Templates list page



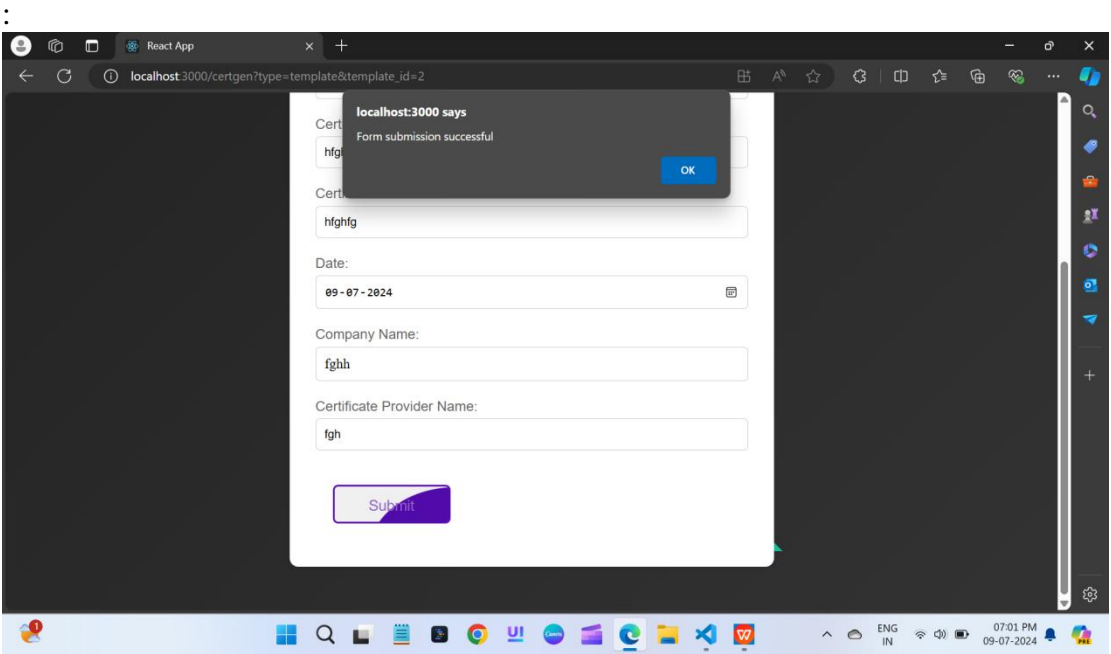
Option choosing :



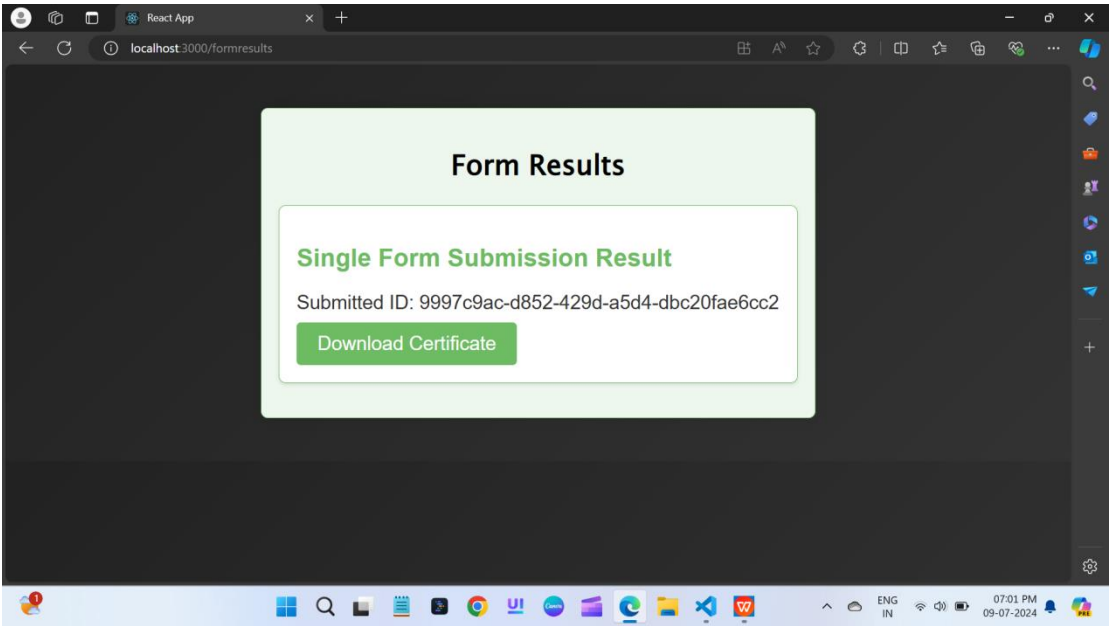
Single certificate form :



Singlecertificate success :



Single certificate form results :



Bulk certificate :

React App

localhost:3000/certgen?type=template&template_id=2

Please Provide The Necessary Details

Single Bulk

Heading:

Certificate About:

Upload Usernames File

Choose File No file chosen

Enter Email to Get Your Certificates:

john@company.com

Number of Certificates to Generate:

1

Date:

dd-mm-yyyy

Company Name:

Certificate Provider Name:

Submit

Bulk certificate form submission success :

React App

localhost:3000/certgen?type=template&template_id=2

localhost:3000 says
Form submission successful

OK

Heading:

certificate of completion

Certificate About:

course completion

Certificate Name:

chandu chitli

Date:

09-07-2024

Company Name:

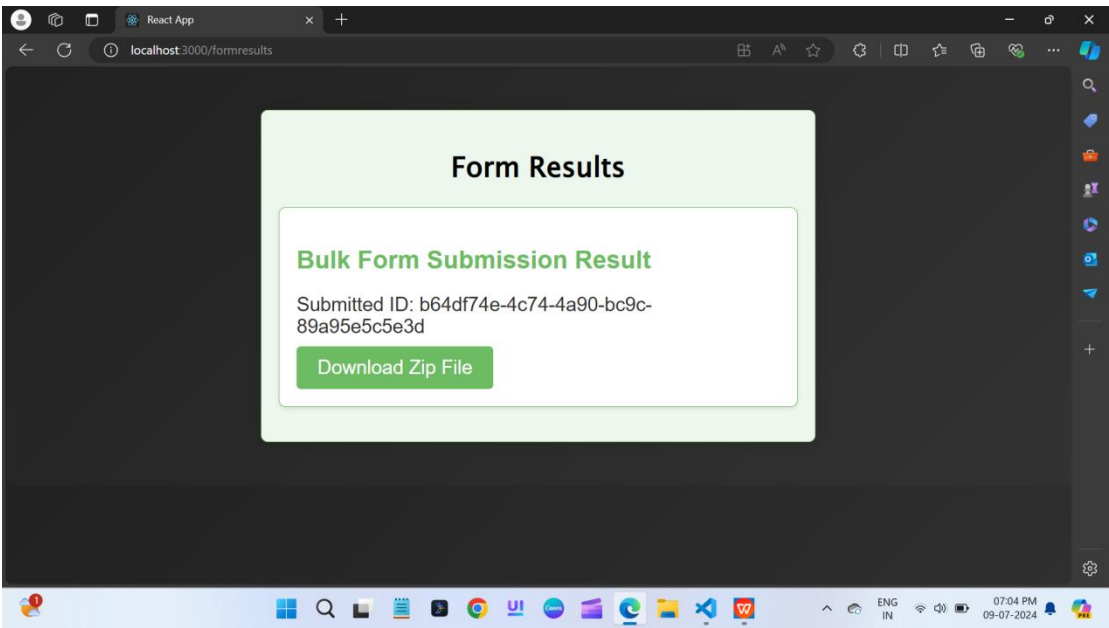
compsci

Certificate Provider Name:

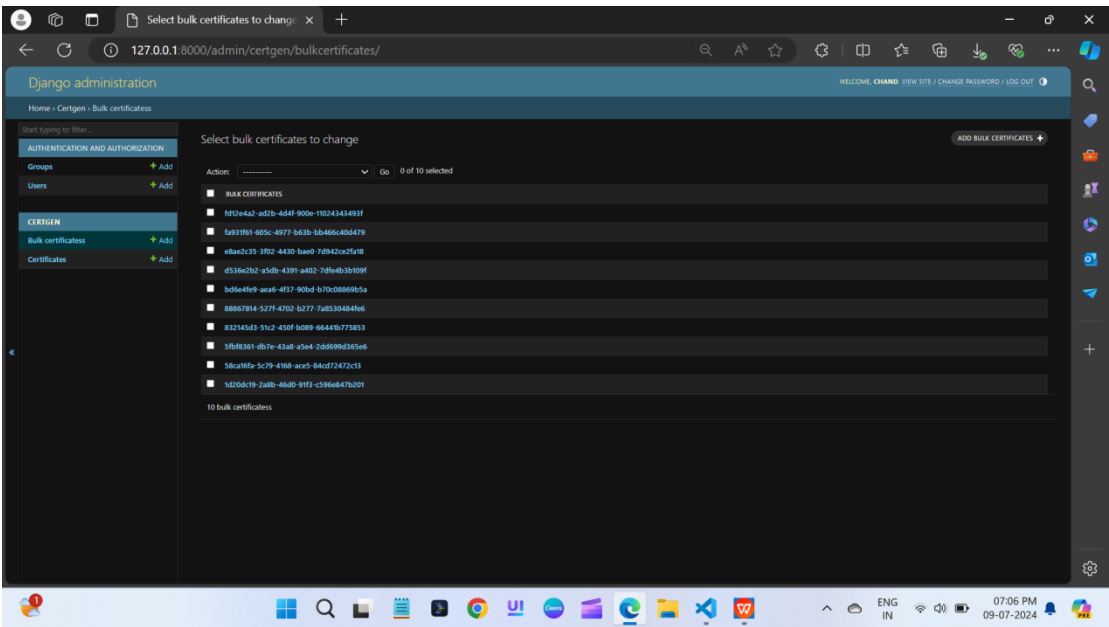
Naveenkumar

Submit

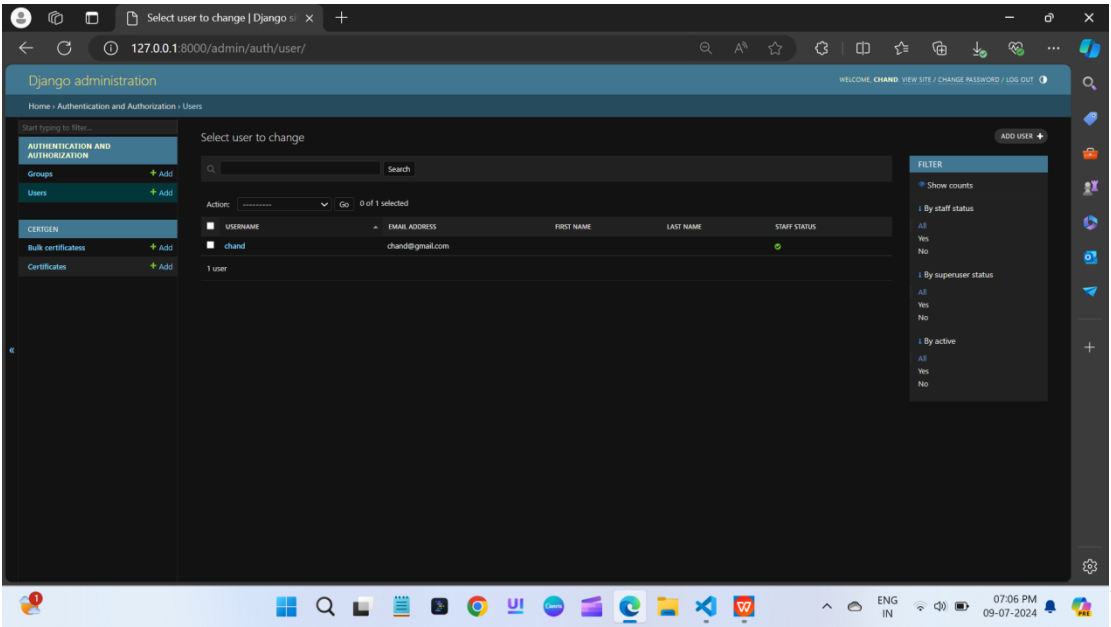
Bulk certificate form submission results :



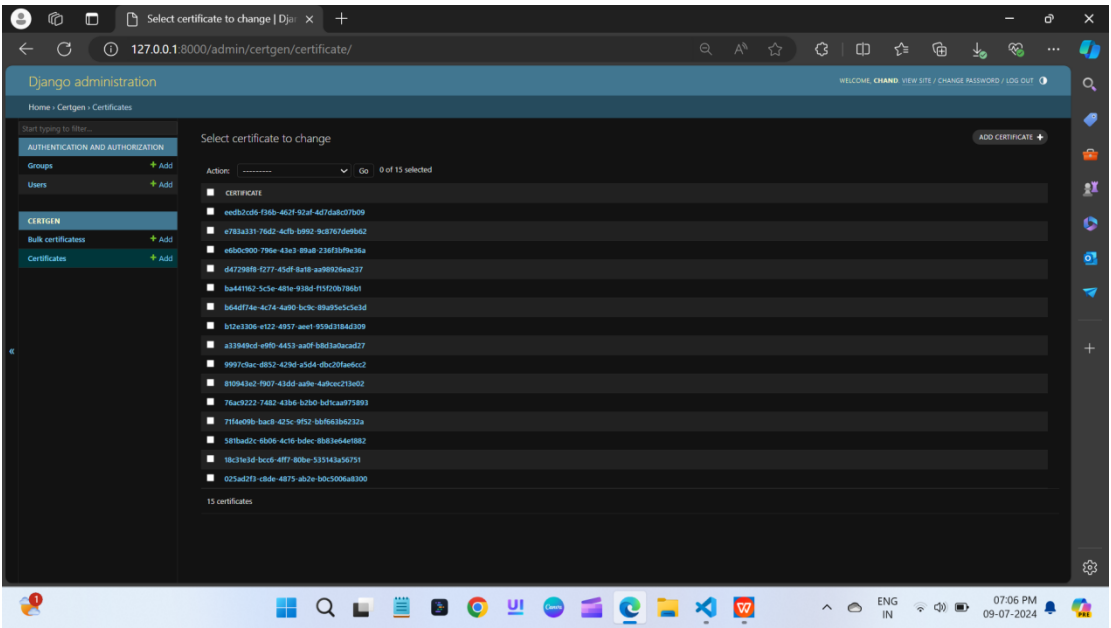
Admin page :



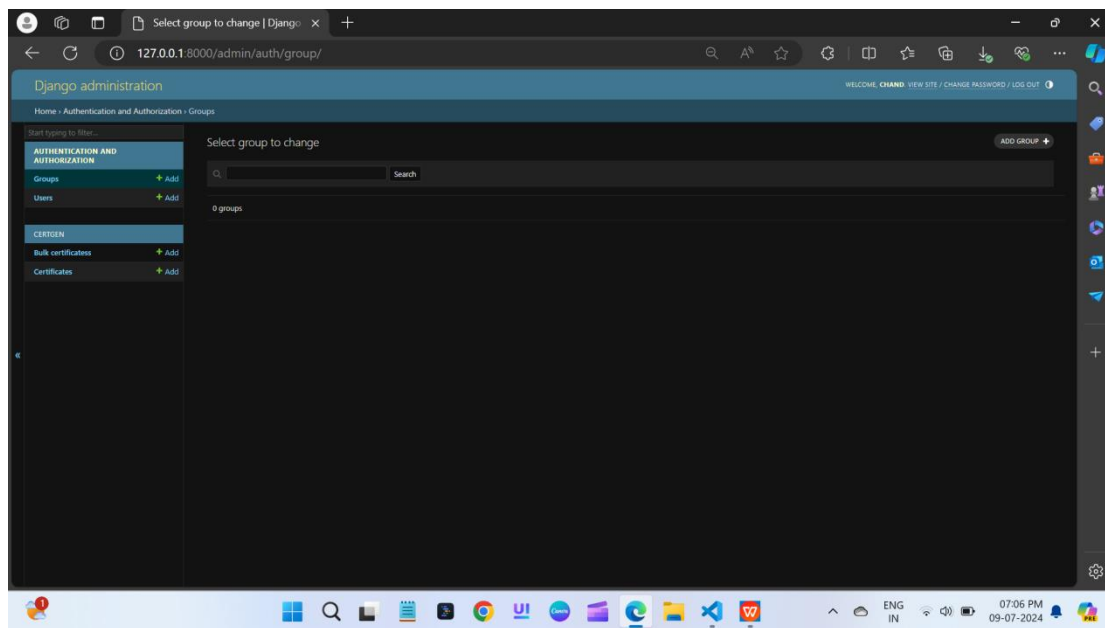
Bulk certificate Data :



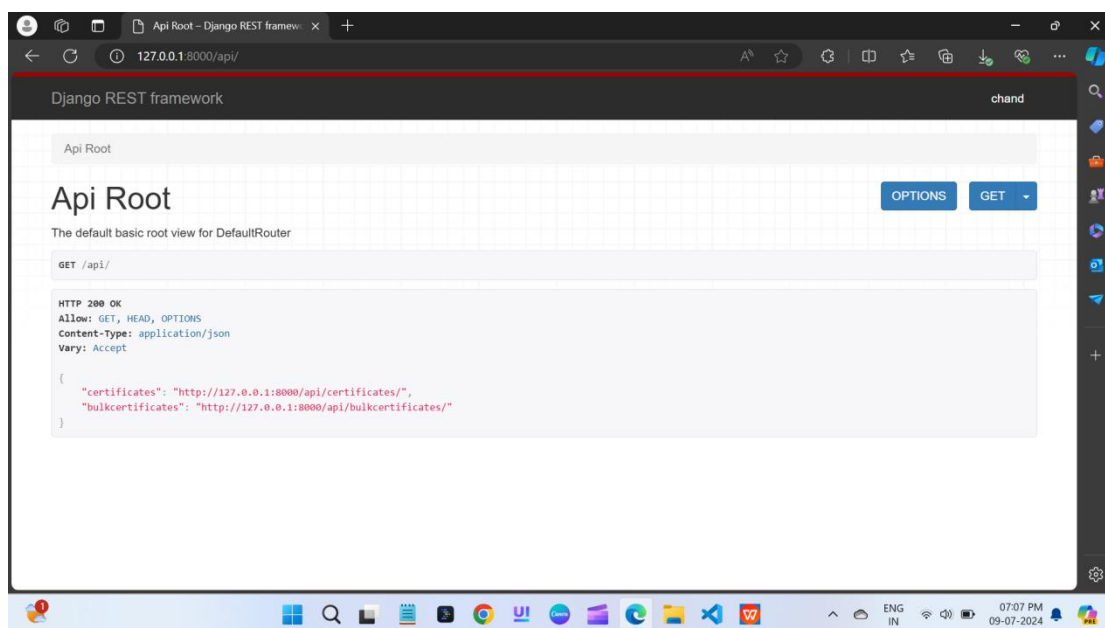
Single certificate forms Data :



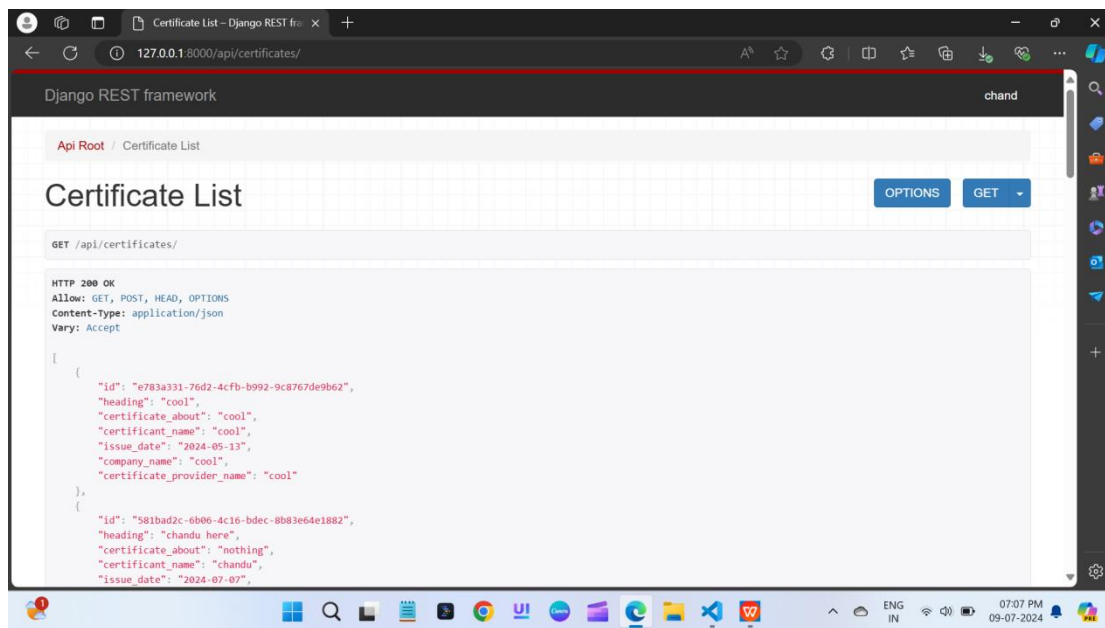
Groups :



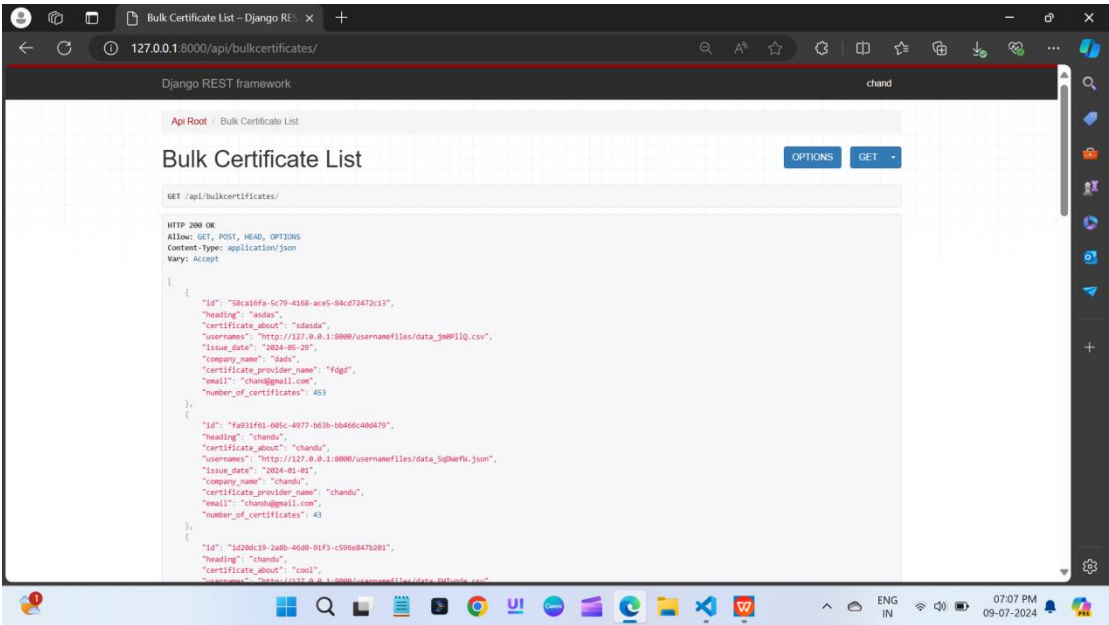
Api page



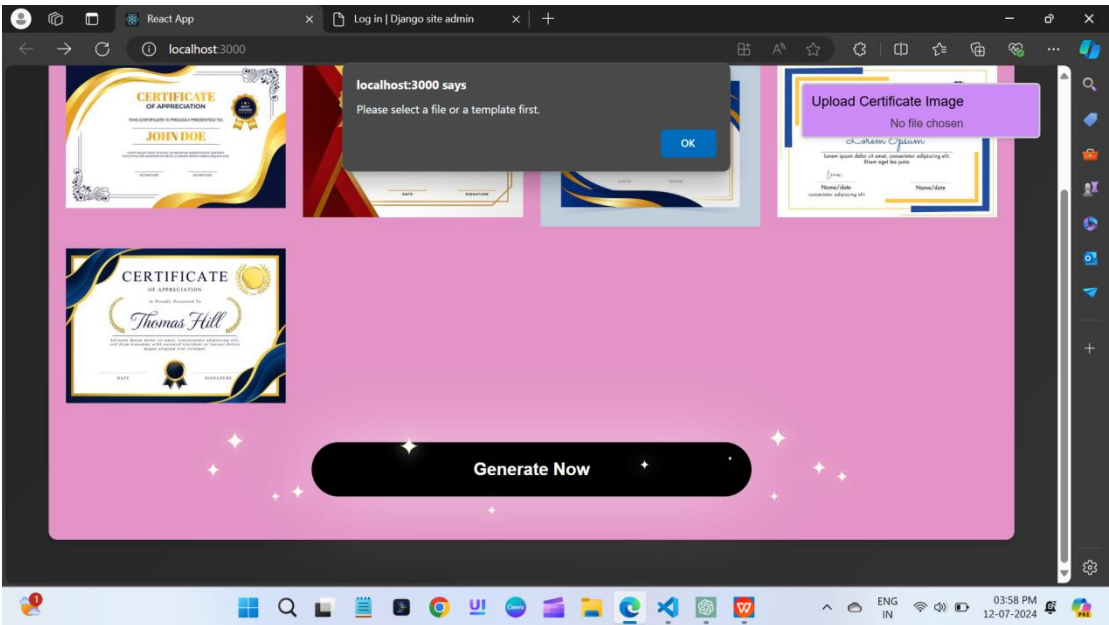
Single certificate results page in restframework :



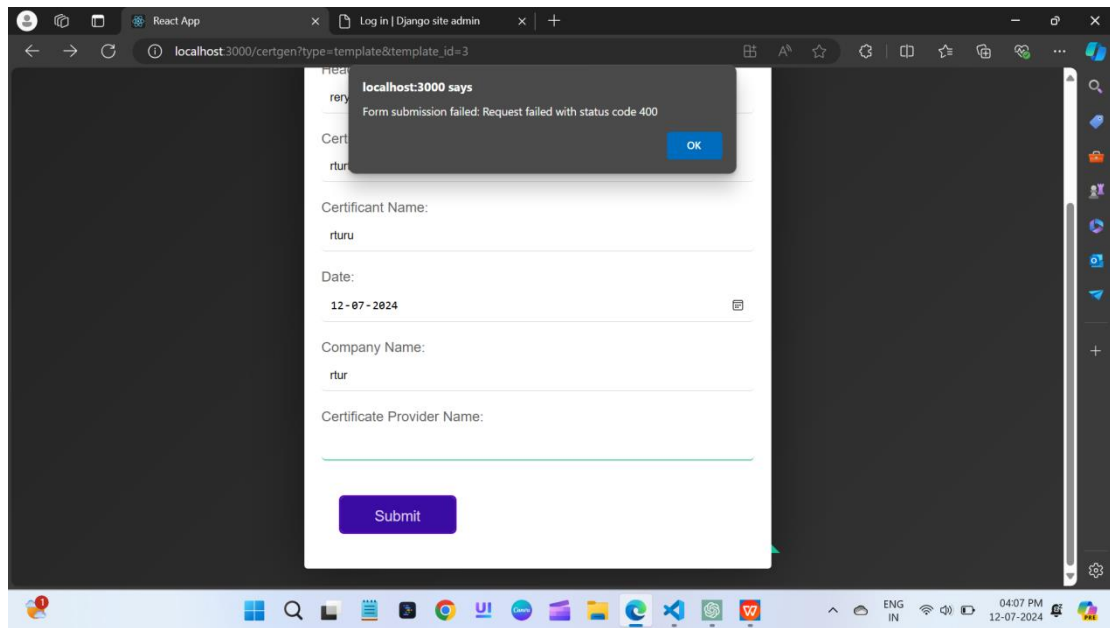
Bulk certificate results page in restframework :



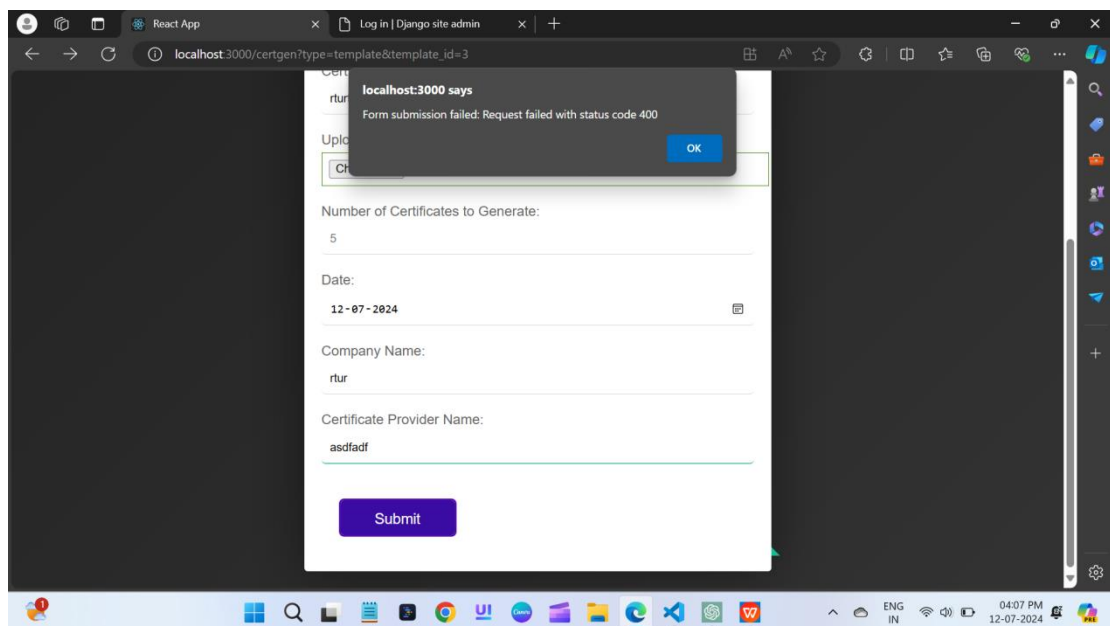
Testing Template list page :



Testing single certificate form :



Testing Bulk Certificate Form :



25. Conclusion & Future Work

The completion of the certificate generation project marks a significant milestone in addressing the inefficiencies of manual processes prevalent in various sectors such as education, professional certifications, and event management. By leveraging modern web technologies and adopting an automated approach, the system offers a streamlined solution for generating both single and bulk certificates efficiently.

Key Achievements

User Interface and Backend Integration: Successfully integrated React for frontend development and Django for backend operations, ensuring seamless communication and data flow.

Enhanced User Experience: Designed an intuitive interface that simplifies certificate generation workflows, enhancing usability and accessibility for end-users.

Scalability and Performance: Implemented strategies for system scalability and performance optimization, including load balancing and resource management.

Contributions to Professional Development

This project has significantly enhanced my skills as a full-stack developer, providing practical experience in designing, implementing, and optimizing web applications. Collaboration with the backend team and exposure to industry-standard practices have enriched my understanding of system architecture and API integrations.

Future work :

User Experience Improvements

Feedback Mechanisms: Introduce feedback mechanisms to gather user insights and iterate on interface design improvements based on user feedback and usability testing. This continuous feedback loop will enhance user satisfaction and usability of the certificate generation system.

Security and Reliability

Enhanced Security Measures: Strengthen data encryption protocols, access controls, and conduct regular security audits to ensure robust data privacy and protection against evolving cyber threats. Implementing stringent security measures will safeguard sensitive information handled by the system.

Research and Innovation

Exploration of Emerging Technologies: Investigate emerging technologies such as blockchain for certificate verification and authentication. This exploration aims to enhance the trustworthiness and security of certificate issuance processes, providing additional layers of verification and authenticity.

26. Certificate Generation App User Manual

Table of Contents

Introduction

1. Overview
2. Features

Getting Started

1. Installation
2. Login and Signup

Using the App

1. Certificate Generation
2. Bulk Certificate Generation
3. Viewing Form Results

Managing Account

1. Logging Out
2. Changing Password

Troubleshooting

1. Common Issues

2. Contact Support

1. Introduction

Overview

The Certificate Generation app allows you to easily create and manage certificates for various purposes. Whether you need a single certificate or multiple certificates in bulk, this app provides a streamlined process to generate them efficiently.

Features

- **Single and Bulk Certificate Generation:** Create individual certificates or upload a list for bulk generation.
- **Form Submission:** Fill in necessary details such as certificate heading, recipient names, dates, and more.
- **Session Management:** Log in securely to manage your certificates.
- **Form Results:** View and download generated certificates.

2. Getting Started

Installation

1. Download the "Certificate Generation" app from the Google Play Store.
2. Install the app on your Android device.
3. Open the app to get started.

Login and Signup

- **Creating an Account:** If you're new to the app, click on "Sign Up" and fill in your details to create an account.

- **Logging In:** Use your registered email and password to log into the app.

3. Using the App

Certificate Generation

1. **Choose Certificate Type:** Select either "Single" or "Bulk" certificate generation.
2. **Fill in Details:** Enter the required information such as certificate heading, recipient names, dates, etc.
3. **Submit Form:** Click on "Submit" to generate the certificate(s).

Bulk Certificate Generation

1. **Upload Usernames:** For bulk generation, upload a CSV or JSON file containing recipient names.
2. **Enter Details:** Fill in other necessary details such as certificate heading, dates, etc.
3. **Submit Form:** Click on "Submit" to generate multiple certificates.

Viewing Form Results

- After submitting a form, you will be redirected to view the form results where you can download the generated certificates.

4. Managing Account

Logging Out

- To log out, go to the menu and click on "Logout." This will securely end your session.

Changing Password

- Currently, password change functionality is not available directly in the app. Please contact support for assistance.

5. Troubleshooting

Common Issues

- **Login Issues:** Ensure your email and password are entered correctly.
- **Form Submission Failures:** Check your internet connection and try again.
- **Session Expired:** If your session expires, log in again to continue.

Contact Support

- For further assistance or technical support, please email support@certificategeneration.com or visit our website at www.certificategeneration.com/support.

27. References

Gamma, E., Helm, R., Johnson, R., Vlissides, J. "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley, 1994.

Smith, J., Brown, A. "Efficiency and Scalability in Web Applications," Journal of Web Engineering, vol. 15, no. 2, pp. 45-58, June 2023.

Django Software Foundation, "Django Documentation," Version 3.2, <https://docs.djangoproject.com/en/3.2/>, Accessed: July 9, 2024.

ReactJS, "React Documentation," Version 18.0, <https://reactjs.org/docs/getting-started.html>, Accessed: July 9, 2024.

IEEE, "IEEE Standard Glossary of Software Engineering Terminology," IEEE Std 610.12-1990, https://standards.ieee.org/standard/610_12-1990.html, Accessed: July 9, 2024.

Official React Documentation

React. (n.d.). *React Documentation*. Retrieved from reactjs.org

React Router Documentation

React Router. (n.d.). *Declarative Routing for React.js*. Retrieved from reactrouter.com

Django Documentation

Django. (n.d.). *Django Documentation*. Retrieved from djangoproject.com

Django REST Framework Documentation

Django REST Framework. (n.d.). *Web API for Django*. Retrieved from django-rest-framework.org

MySQL Documentation

1. MySQL. (n.d.). *MySQL Documentation*. Retrieved from dev.mysql.com

Axios Documentation

Axios. (n.d.). *Axios GitHub Repository*. Retrieved from github.com/axios/axios

Best Practices in React

Dholakia, U. (2020). *React Best Practices: An Overview*. Retrieved from medium.com

