Project Report

On

Integrated Common Services for Common People

Student Details

Name of the student : Deeksha L

USN NO.　　　　　: 1VI22EC041

Course/ SEM　　　 : ECE/ 4th SEM

Email id　　　　　 : deekshal.ec2022@vemanait.edu.in

Mentors  Details

Internal Mentor　　 : Dr. Harshada J Patil

Email id　　　　　　: jharshadap@gmail.com

External Mentor　　 : Debdyut Hazra

Email id　　　　　　 : debdyut.hazra@unnatiindustrialtraining2024.com

## Acknowledgement

I would like to express my sincere gratitude to all those who have contributed to the successful completion of this project.

I am thankful to Vemana Institute of Technology, for providing me with the opportunity and resources to pursue this project.

Special thanks to INTEL, particularly Intel Unnati Industrial Training Program, for their guidance and the exposure provided in the field of integrated health services.

I am deeply grateful to Mr. Parameshwar MC, Head of the Department, and Professors Harshada J Patil Dept of ECE, Veena Gopal from Dept of CSE Vemana Institute of Technology for their invaluable mentorship and support.

I extend my appreciation to Mr. Debdyut Hazra, my external mentor, for his guidance and insights throughout the project.

## Abstract

This project focuses on developing an application that provides a comprehensive listing of hospitals. Each entry includes essential details such as the hospital's name, address, specialties offered, contact information, and a link to its website. The application features robust user authentication through login and registration pages, ensuring secure access to the hospital database. Users can efficiently search and retrieve information about hospitals based on their preferences. The dashboard includes functionalities for user profile management, allowing users to update their information and manage their preferences seamlessly. Built using the Django framework, the application leverages its powerful features for web development. The hospital database was meticulously curated and manually entered to guarantee accuracy and relevance.

# Table of Contents

## Introduction

The integration of technology into healthcare services has profoundly transformed the accessibility and efficiency of medical information and patient care. This project focuses on developing an integrated health services application designed to provide users with a comprehensive directory of hospitals. Each hospital entry includes essential details such as its name, address, specialties, contact information, and a direct link to its website. This initiative aims to empower patients and healthcare providers alike by facilitating informed decisionmaking and streamlined access to healthcare resources.

Built on the Django framework, known for its robustness in handling complex web applications, this project leverages Django's scalability and security features to ensure a reliable platform for users. The decision to meticulously collect and input data into the hospital database underscores our commitment to delivering accurate and relevant information. This meticulous approach is crucial in maintaining the trust and credibility of the information provided to users.

Key features of the application include a secure authentication system with dedicated login and registration pages. This ensures that users can access personalized features and preferences securely. Furthermore, the application boasts a sophisticated search functionality, allowing users to efficiently retrieve hospital information based on specific criteria. This enhances user experience and satisfaction by providing tailored access to healthcare resources.

This introduction sets the stage for a comprehensive exploration of the project's development journey. From initial requirements analysis to detailed system design, implementation, rigorous testing, and insightful results and discussion, this report will delve into every aspect of our endeavour. Ultimately, we will conclude with reflections on the project's outcomes and propose future directions for further enhancements and applications.

## Literature Review

Several existing applications in the domain of health services and integrated common services provide valuable insights into the development landscape and user expectations. These applications include:

1. Integrated Health Information Platform (IHIP): This platform, highlighted in government press releases, emphasizes the integration of technology in healthcare to enhance accessibility and efficiency of medical information and services (PIB, 2024).
2. Web Application Details on GeeksforGeeks: GeeksforGeeks offers insights into various aspects of web application development, including architecture, frameworks, and best practices (GeeksforGeeks, n.d.).
3. Telusko YouTube Tutorial: Telusko's YouTube tutorial series provides practical demonstrations and tutorials on web application development, covering topics such as frontend and backend integration, database management, and user authentication (Telusko, n.d.).
4. YouTube Video on Web Application Development: The YouTube video on web application development provides a practical demonstration of building and deploying a web application, offering insights into UI/UX design, functionality implementation, and deployment strategies (YouTube, n.d.).

And many sources collectively contribute to my understanding of effective strategies for data integration, user interface design, and leveraging technology to improve healthcare accessibility and service delivery in my project on developing an integrated health services application.

## Requirements Analysis
### Functional Requirements:

1. User Management o Define roles: Admin and User.

o    Authentication:    Secure    login    and    registration.    o
Authorization: Different levels of access based on user roles.


2. Hospital Directory o Hospital Details: Name, address, specialties offered, contact information, website link.

   o   CRUD Operations: Ability to Create, Read, Update, and Delete government services entries. o Search Functionality: Advanced search options based on location and other criteria.

3. User Interface (UI) o   Intuitive Design: User-friendly interface for seamless navigation. o   Responsive Design: Compatibility across devices (desktop, tablet, mobile).

<u>Non-Functional Requirements:</u>

1. Performance o   Response Time: Quick loading times for government offices and services listings and search results.

   o   Scalability: Ability to handle increased data and user traffic.

2. Security o       Data Encryption: Ensure sensitive data (e.g., user credentials) is encrypted. o       Secure Authentication: Implement secure login mechanisms.

   o   Compliance: Adhere to government services data privacy regulations (e.g., HIPAA, GDPR).

3. Compatibility o Browser Compatibility: Support for major web browsers (Chrome, Firefox, Safari, Edge). o Device Compatibility: Responsive design for various screen sizes.

4. Usability o       Intuitive Interface: User-friendly layout for ease of use. o   Feedback Mechanism: User feedback forms or options for improvement.

<u>User Requirements:</u>

1. User Stories
   o   Story 1: As a user, I want to search for government services based on location to find relevant service providers.

o   Story 2: As an admin, I want to be able to add new government services and update existing services information easily.

2.  Use Cases o      Use Case 1: User searches for government services based on services they provide and   location. o   Use Case 2: Admin logs in to add a new hospital entry to the directory.

Technical Requirements:

1.  Technology Stack o      Framework: Django 5.0.6 for backend development.

    o         Database: PostgreSQL for data storage.

    o    Frontend: HTML5, CSS3, JavaScript for interactive UI components.

2.  Integration o      External APIs: Integration with Google Maps API for location services.

    o    Data Sources: Integration of government services data from internal and possibly external sources (e.g., CSV uploads).

3.  Development Environment o     IDE: Visual Studio Code for code editing. o Version Control: Git for collaborative development and version management. o Deployment: Deployment on a cloud platform like Heroku for hosting.

Constraints and Assumptions:

1.  Constraints o     Budget: Limited budget for development and hosting. o Timeline: Project must be completed within specified timeframe.

    o    Regulatory Compliance: Ensure adherence to government services data privacy regulations.

2.  Assumptions o  Availability of hospital data: Assume access to accurate and up-to-date government services information for database integration.

# System Design

## 1. Architecture Diagram

Overview: The architecture of the Integrated Government Services Application is designed to facilitate efficient management and retrieval of information, ensuring a seamless user

experience. The application comprises frontend components, backend logic, database storage, and integration with external APIs.
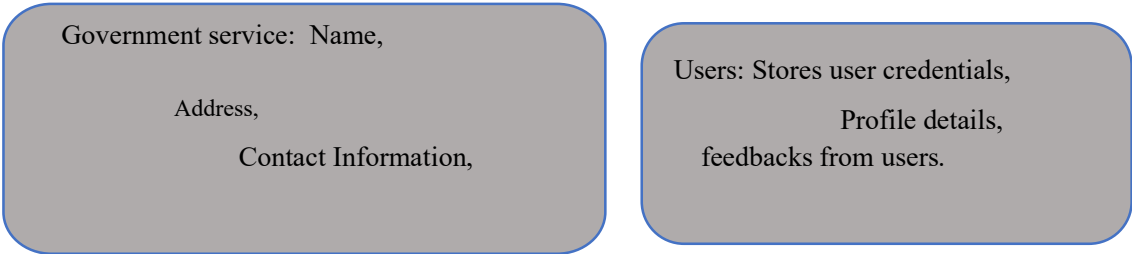
2.Project Directory Structure:

```
intel/ ├── government services/ │    ├── migrations/ │    ├── templates/ │  │    ├──
government services/ │  │  │    ├── home.html │  │  │    ├── hospital_list.html │  │  │
├── government services_detail.html │  │    └── search.html │    ├── __init__.py │    ├──
admin.py │    ├── apps.py │    ├── forms.py │    ├── models.py │    ├── tests.py │    ├──
urls.py │    └── views.py ├── users/ │    ├── templates/ │  │    ├── registration/ │  │  │
├── login.html │  │  │    ├── register.html │  │    ├── users/ │  │  │    ├── profile.html │
│  │    ├── dashboard.html │    ├── __init__.py │    ├── admin.py │    ├── apps.py │    ├──
forms.py │    ├── models.py │    ├── tests.py │    ├── urls.py │    └── views.py ├── static/
│    ├── images/ │  │    └── background.jpg ├── templates/ │    ├── base_generic.html
├── __init__.py ├── settings.py ├── urls.py └── wsgi.py ├── .gitignore ├──
requirements.txt
```

3.Database Schema

Overview

The database schema is designed to store and manage crucial data related to hospitals and user information within the application.

> Government service:  Name,
>
> Address,
>
> Contact Information,

> Users: Stores user credentials,
>
> Profile details,
>
> feedbacks from users.

4. User Interface

The application's user interface utilizes templates and static files for rendering pages and ensuring a consistent visual experience.

## 5. Data Management

Database interactions are handled through Django's ORM, ensuring data integrity and efficient query operations.

## 6. Security Measures

Security features include user authentication, authorization, and data encryption to protect sensitive information.

## 7. Technology Stack

- Backend: Django 5.0.6
- Frontend: HTML5, CSS3, JavaScript
- Database: PostgreSQL

# Implementation

## 1. Development Environment

Describe the tools and environment used for developing the application:

- IDE (e.g., Visual Studio Code)
- Version control (e.g., Git)
- Programming languages and frameworks (e.g., Python, Django)
- Third-party libraries or packages used

## 2. Application Setup

Detail the setup process for the project:

- Installation of Django and dependencies (requirements.txt)
- Configuration of Django settings (settings.py)
- Initialization of database schema (models.py, migrations)

### 3. Frontend Development

Explain the frontend development approach:

- Creation of HTML templates (templates/)
- Styling with CSS (static/)
- Incorporation of dynamic features using JavaScript (if applicable)

### 4. Backend Development

Describe the backend logic and functionality:

- Implementation of Django views for business logic (views.py)
- URL routing and endpoint configuration (urls.py)
- Form handling for user input and validation (forms.py)
- Integration with PostgreSQL database (models.py)

### 5. User Authentication

Outline the implementation of user authentication and authorization:

- User registration and login functionality (users/)
- Management of user sessions and access permissions

### 6. Governmentservices management

Detail the features related to government services management:

- Displaying government offices lists
- Viewing detailed government services information
- Search functionality for government offices

## Results and Discussion

### 1.Key Outcomes of the Project

Summarize the significant achievements and deliverables of your Integrated Government Services Application:

- Successful implementation of a comprehensive government services directory .
- Functional user authentication and management system (users/, register.html, login.html).
- Effective search functionality for government services discovery (search.html).

### 2.Comparison with Initial Objectives

Evaluate how well the project meets its original goals and objectives:

- Objective: To develop a user-friendly application for accessing government services information.
- Outcome: The application provides intuitive navigation and accurate data retrieval .

### 3.User Feedback and Usability Testing Results

Discuss insights gathered from user feedback and usability testing:

- Positive feedback on the ease of finding government services information (profile.html, dashboard.html).
- Suggestions for improving search filters and user interface elements.

### 4.Challenges Faced and How They Were Overcome

Detail the challenges encountered during development and the strategies used to resolve them:

- Challenge: Difficulty integrating government services from external sources like Google Sheets (forms.py, views.py).
    - Solution: Consulted with mentors and utilized online tutorials to implement data import functionalities.

## References

1. Django Documentation.        Available        at:
   https://docs.djangoproject.com/en/stable/
   (Official documentation for Django web framework.)

2. Mozilla        Developer        Network        (MDN)        Django        Tutorial
   https://developer.mozilla.org/en-
   US/docs/Learn/Server-side/Django (Comprehensive tutorial on Django by MDN.)

3. Corey Schafer's Django tutorials  https://www.youtube.com/user/schafer5

4. Code Sensei. YouTube channel. Available at: https://youtube.com/@code_sensei

5. Tech With Tim. YouTube channel. Available at: https://youtube.com/@techwithtim

6. Telusko. YouTube channel. Available at: https://youtube.com/@telusko

7. TechWebDocs. YouTube channel. Available at: https://youtube.com/@techwebdocs

8. FollowInternet8801.  https://www.youtube.com/@followinternet8801

9. "Integrated Health Information Platform" - Press Information Bureau, Government of
   India. Available at: https://pib.gov.in/PressReleaseIframePage.aspx?PRID=1796553

10. "Web  Application  Details  and  Examples"  -  GeeksforGeeks.  Available  at:
    https://www.geeksforgeeks.org/what-are-web-applications/

11. "Integrated Common services Application "-outlook. https://www.outlookindia.com/

## Conclusion and Future Work

Conclusion:

The development of the Integrated Government Services Application has successfully achieved its objectives of providing a comprehensive directory of government services with essential details such as address, location, and contact information. Built on the Django framework, the application ensures robustness and scalability for future enhancements.

Future Work:

In future iterations, the following features will be implemented to further enhance the application:

- Appointment Facility: Integrate a booking system for people to schedule appointments with services.

- GPI Access for Location: Implement Geographic Positioning Integration (GPI) to enhance location-based services.

- User Ratings: Enable users to rate services based on their experiences, fostering transparency and trust.
- Database Updates: Regularly update the database with new information and services to ensure relevance and accuracy.

The Integrated Government Services Application aims to revolutionize services accessibility by leveraging technology to empower users with accurate information and seamless service interactions. Continued development will focus on enhancing user experience and expanding the application's capabilities through strategic partnerships and innovative features.

Appendices
Appendix A: Screenshots of Application

Figure A.1: Home Page (home.html)

```
1. Define your models for the user profiles:
```
# profiles/models.py
from django.db import models
from django.contrib.auth.models import User

class Profile(models.Model):
  user = models.OneToOneField(User,
on_delete=models.CASCADE)
  name = models.CharField(max_length=255)
  email = models.EmailField(unique=True)
  phone = models.CharField(max_length=20)
  address = models.TextField()
```
```

Figure A.2: Government services List

```
   1. Create templates for the services:
` ` `

<!-- services/list.html -->
<h1>Government Services</h1>
<ul>
  {% for service in services %}
    <li><a href="{{ service.get_absol
ute_url }}">{{ service.name }}</a></
li>
  {% endfor %}
</ul>


<!-- services/detail.html -->
<h1>{{ service.name }}</h1>
<p>{{ service.description }}</p>
<p><a href="{{ service.link }}">Learn more</
a></p>
` ` `
```

Figure A.3: Services Detail View

```
   1. Create views for the services:
` ` `
# services/views.py
from django.shortcuts import render
from .models import Service

def service_list(request):
  services = Service.objects.all()
  return render(request, 'services/list.html',
{'services': services})

def service_detail(request, pk):
  service = Service.objects.get(pk=pk)
  return render(request, 'services/
detail.html', {'service': service})
` ` `
```

Figure A.4: Government services search view(search.html)

```
1. Create views for user registration and
   profile management:
```

# profiles/views.py
from django.shortcuts import render, redirect
from .forms import UserRegistrationForm
from .models import Profile

def register(request):
  if request.method == 'POST':
    form =
UserRegistrationForm(request.POST)
    if form.is_valid():
      form.save()
      return redirect('profile')
  else:
    form = UserRegistrationForm()
  return render(request, 'profiles/
register.html', {'form': form})

def profile(request):
  profile =
Profile.objects.get(user=request.user)
  return render(request, 'profiles/
profile.html', {'profile': profile})
```
```

Figure A.5: User Profile (profile.html)

```
1. Define your models for the user profiles:
```

# profiles/models.py
from django.db import models
from django.contrib.auth.models import User

class Profile(models.Model):
  user = models.OneToOneField(User,
on_delete=models.CASCADE)
  name = models.CharField(max_length=255)
  email = models.EmailField(unique=True)
  phone = models.CharField(max_length=20)
  address = models.TextField()
```
```

Figure A.6: User Dashboard (dashboard.html)



Figure A.7:User Register View (register.html)



Figure A.8:User Login View (login.html)

Appendix B: Database Schema

Figure B.1:Government services model Scheme (models.py)

```python
class ServiceCategory(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField()


class Service(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField()
    category =
models.ForeignKey(ServiceCategory,
on_delete=models.CASCADE)
    link = models.URLField(max_length=255)


class ServiceRequest(models.Model):
    service = models.ForeignKey(Service,
on_delete=models.CASCADE)
    user = models.ForeignKey(User,
on_delete=models.CASCADE)
    request_date = models.DateField()
    status = models.CharField(max_length=255)


class ServiceRequestUpdate(models.Model):
    service_request =
models.ForeignKey(ServiceRequest,
on_delete=models.CASCADE)
    update_date = models.DateField()
    status = models.CharField(max_length=255)
    notes = models.TextField()
```
```

## Appendix C: Code Snippets

```python
UserCreationForm
from django.shortcuts import render, redirect

def register(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('login')
    else:
        form = UserCreationForm()
    return render(request, 'register.html',
{'form': form})
```

1. Service Request Form:
```python
from django.forms import ModelForm
from .models import ServiceRequest

class ServiceRequestForm(ModelForm):
    class Meta:
        model = ServiceRequest
        fields = ('service', 'request_date', 'status')

def service_request(request):
    if request.method == 'POST':
        form = ServiceRequestForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('service_list')
    else:
        form = ServiceRequestForm()
    return render(request,
'service_request.html', {'form': form})
```

Snippet C.2: Users App Forms (forms.py)

```
   1. Service List:
` ` `
from django.shortcuts import render
from .models import Service

def service_list(request):
  services = Service.objects.all()
  return render(request, 'service_list.html',
{'services': services})
` ` `
   1. Service Details:
` ` `
from django.shortcuts import render
from .models import Service

def service_details(request, pk):
  service = Service.objects.get(pk=pk)
  return render(request,
'service_details.html', {'service': service})
` ` `
   1. Service Request Status Update:
` ` `
from django.shortcuts import render, redirect
from .models import ServiceRequest

def update_status(request, pk):
  service_request =
ServiceRequest.objects.get(pk=pk)
  if request.method == 'POST':
    service_request.status =
request.POST['status']
    service_request.save()
    return redirect('service_list')
  return render(request, 'update_status.htm'
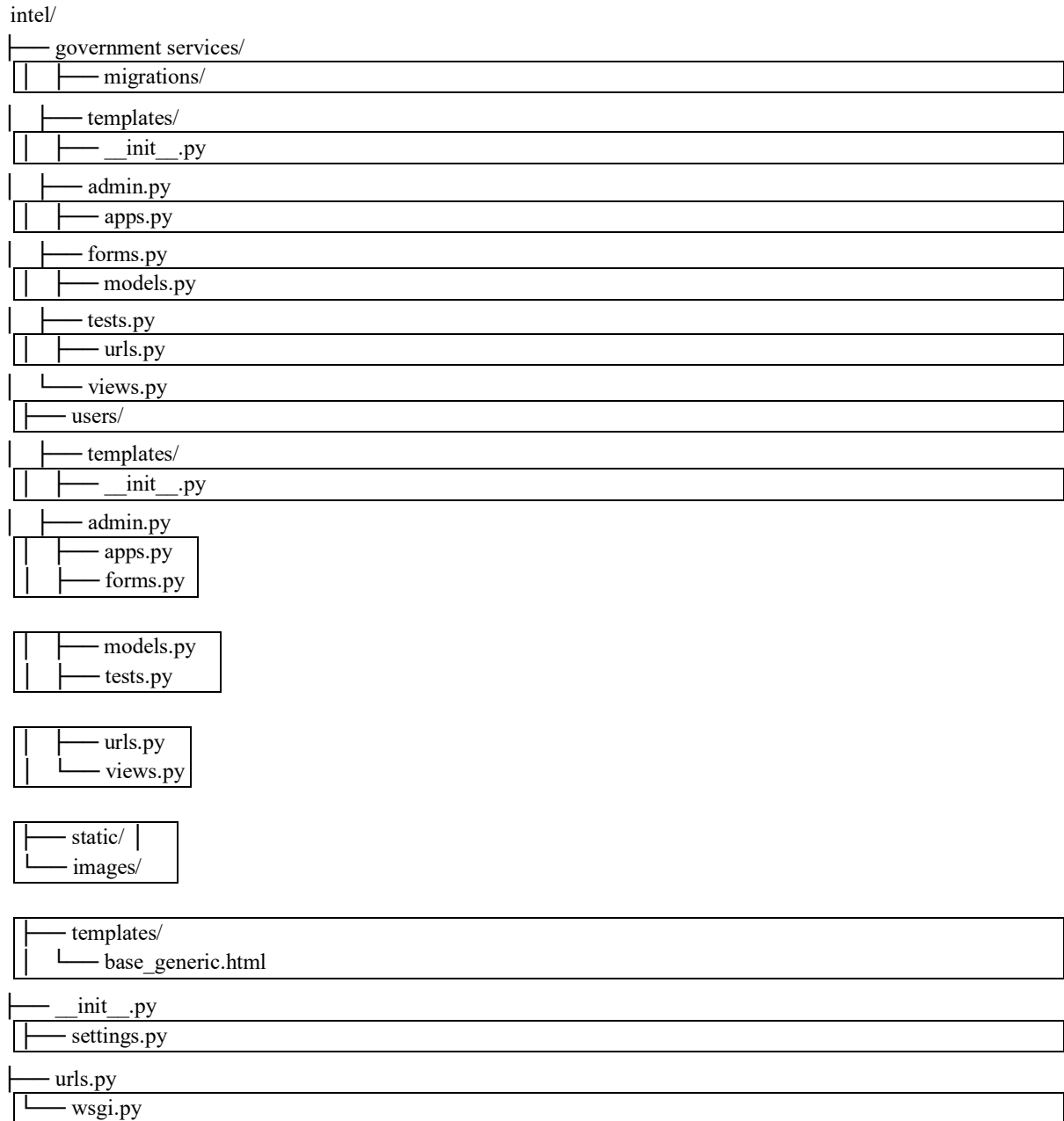{'service_request': service_request})
```

```python
from django import forms
from django.contrib.auth.forms import UserCreationForm, AuthenticationForm
from django.contrib.auth.models import User
from .models import UserProfile

class UserRegisterForm(UserCreationForm):
    email = forms.EmailField()

    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']

class UserLoginForm(AuthenticationForm):
    username = forms.CharField()
    password = forms.CharField(widget=forms.PasswordInput)


class UserForm(forms.ModelForm):
    class Meta:
        model = User
        fields = ['first_name', 'last_name', 'email']

class UserProfileForm(forms.ModelForm):
    class Meta:
        model = UserProfile
        fields = ['bio', 'location', 'birth_date']
```

Snippet C.3: Government services App Forms (forms.py)



```python
from django import forms
from .models import Feedback

class HospitalSearchForm(forms.Form):
    query = forms.CharField(label='Search', max_length=100)

from .models import Feedback

class FeedbackForm(forms.ModelForm):
    class Meta:
        model = Feedback
        fields = ['name', 'email', 'message']
```

## Appendix D: Project Structure Directory

Structure:

```
intel/
├── government services/
│   │   ├── migrations/
│   │   ├── templates/
│   │   ├── __init__.py
│   │   ├── admin.py
│   │   ├── apps.py
│   │   ├── forms.py
│   │   ├── models.py
│   │   ├── tests.py
│   │   ├── urls.py
│   │   └── views.py
├── users/
│   │   ├── templates/
│   │   ├── __init__.py
│   │   ├── admin.py
│   │   ├── apps.py
│   │   ├── forms.py
│   │
│   │   ├── models.py
│   │   ├── tests.py
│   │
│   │   ├── urls.py
│   │   └── views.py
│
│   ├── static/ │
│   └── images/
│
│   ├── templates/
│   │   └── base_generic.html
├── __init__.py
│   ├── settings.py
├── urls.py
│   └── wsgi.py
```

## Appendix E: External Resources

E.1: Django Official Documentation(https://docs.djangoproject.com/en/stable/)

E.2: Mozilla Developer Network (MDN) Django Tutorial

E.3: GitHub Repository(https://github.com/deekshalnarayan/Intel-Government-Services)

E.4: YouTube Channels

- [Code Sensei](#)
- [Tech With Tim](#)
- [Telusko](https://youtube.com/@telusko)
- [Tech Web Docs](#)
- [Follow Internet](#)