**Hazard Maps Project**

**1. Introduction**

**1.1. Context and Motivations**

意大利公共当局负责收集、处理和分析与自然灾害相关的数据集，以生成滑坡灾害地图。通过将这些灾害地图与意大利各区域的人口、家庭、建筑物、工业和服务、文化遗产及领土数据相结合，可以全面评估各区域暴露于滑坡灾害的情况。这种多维度的数据分析有助于评估滑坡对不同区域的潜在影响，从而为决策者提供关键的信息支持。

具体而言，分析涵盖以下几个方面的数据：

* **Territory**: 区域的地理和物理特征。
* **Population**: 各区域的人口分布和密度。
* **Families**: 家庭结构和分布。
* **Buildings**: 建筑物的数量和类型。
* **Industries and Services**: 工业和服务业的分布及其经济重要性。
* **Cultural Heritage**: 文化遗产地点的分布及其历史和文化价值。

本项目的分析范围包括国家级和区域级两个层面。通过结合这些多维度的数据，决策者可以更有效地访问、可视化和分析相关信息，从而制定科学的灾害管理和预防措施，提升灾害应对能力和公共安全水平。

**1.2. Definitions, Acronym, Abbreviations**

* **ISPRA**: Istituto Superiore per la Protezione e la Ricerca Ambientale (意大利环境保护与研究高级研究所)。ISPRA负责环境保护和研究，并提供与自然灾害相关的数据。
* **PIR**: Hazards and Risk Indicators (风险和危险指标)。这些指标用于评估和分析滑坡等自然灾害的风险。
* **REST API**: Representational State Transfer Application Programming Interface (表述性状态转移应用程序编程接口)。一种用于网络服务的接口设计风格，允许客户端与服务器之间进行交互。
* **GeoJSON**: Geospatial JavaScript Object Notation (地理空间JavaScript对象表示法)。一种用于编码地理数据结构的开放标准格式，常用于Web地图应用中。
* **WMS**: Web Map Service (网络地图服务)。一种标准协议，用于通过互联网请求地理信息服务以生成地图。
* **PostgreSQL**: 一种开源的对象关系型数据库管理系统，强调扩展性和标准化的合规性，广泛用于各种复杂的数据分析任务。
* **PostGIS**: PostgreSQL的一个扩展，为其增加了对地理对象的支持，使其能够处理地理信息系统(GIS)数据。
* **Flask**: 一种用Python编写的轻量级Web应用框架，采用简单、灵活的设计，适用于快速开发Web服务和REST API。
* **GeoServer**: 一种开源的服务器软件，允许用户共享和编辑地理数据。它设计用于与许多不同的数据源交互，并能输出数据到多种客户端，包括网页地图。
* **Pandas**: 一种Python数据分析库，提供高性能、易于使用的数据结构和数据分析工具。
* **GeoPandas**: 在Pandas基础上扩展的地理空间数据分析库，使Pandas支持地理数据类型的处理。

**1.3. Purpose**

本项目的目的是开发一个互动式的Web应用程序，以支持意大利防灾管理活动。该应用程序将整合来自ISPRA的PIR数据，通过用户友好的界面，展示滑坡风险和人口、建筑物、家庭、工业和服务、文化遗产及领土的暴露情况，帮助决策者进行有效的灾害管理。

应用程序的主要交互行为包括：

* **区域选择**: 用户可以从下拉菜单中选择不同的区域，查看该区域的详细统计信息和暴露情况。这些统计信息包括人口、家庭、建筑物、工业和服务、文化遗产及领土的分布和数量。
* **地图视图**: 应用程序提供滑坡风险地图，显示意大利各区域的滑坡风险分布情况。地图支持缩放和平移，用户可以详细查看特定区域的风险情况。
* **数据展示**: 通过数据卡片和堆叠柱状图形式展示选定区域的统计信息和滑坡风险数据。用户可以清晰地看到各类别的数据，如人口、家庭、建筑物、工业和服务、文化遗产等在不同风险级别下的分布情况。
* **数据下载**: 应用程序允许用户下载特定区域的统计信息和滑坡风险数据，支持数据的进一步分析和使用。
* **图例和样式**: 应用程序包含详细的图例，帮助用户理解地图上的各类风险标识和数据表示。
* **地图图层选择**: 用户可以选择不同的地图图层，以便更详细地查看各种信息。这些图层包括：
  + **OpenStreetMap**: 提供基本地图视图。
  + **Carto Positron**: 提供简洁明了的地图风格，适合背景地图使用。
  + **Esri Topographic**: 提供详细的地形图视图。
  + **Esri World Imagery**: 提供卫星影像视图。
  + **Landslide Hazard Map**: 显示滑坡风险地图层，标示出滑坡风险区域。
  + **Population Density**: 显示人口密度分布图层。
  + **Risk Indicator Buildings**: 显示建筑物风险指标图层。
  + **Risk Indicator Cultural Heritage**: 显示文化遗产风险指标图层。
  + **Risk Indicator Families**: 显示家庭风险指标图层。
  + **Risk Indicator Industries and Services**: 显示工业和服务风险指标图层。
  + **Risk Indicator Population**: 显示人口风险指标图层。
  + **Risk Indicator Territory**: 显示领土风险指标图层。
  + **Regions**: 显示区域边界图层。
  + **Selected Region**: 突出显示选定的区域。

通过这些交互功能，用户能够方便地访问和分析相关信息，从而制定科学的灾害管理和预防措施，提升灾害应对能力和公共安全水平。

**1.4. Scope and Limitations**

**Scope:**

* 开发一个包含数据库、REST API和互动仪表盘的客户-服务器应用程序。
* 数据将从ISPRA IdroGEO API获取，包含滑坡风险和人口、建筑物、家庭、工业和服务、文化遗产及领土的普查数据。
* 应用程序将支持国家级和区域级的数据展示和分析。

**Limitations:**

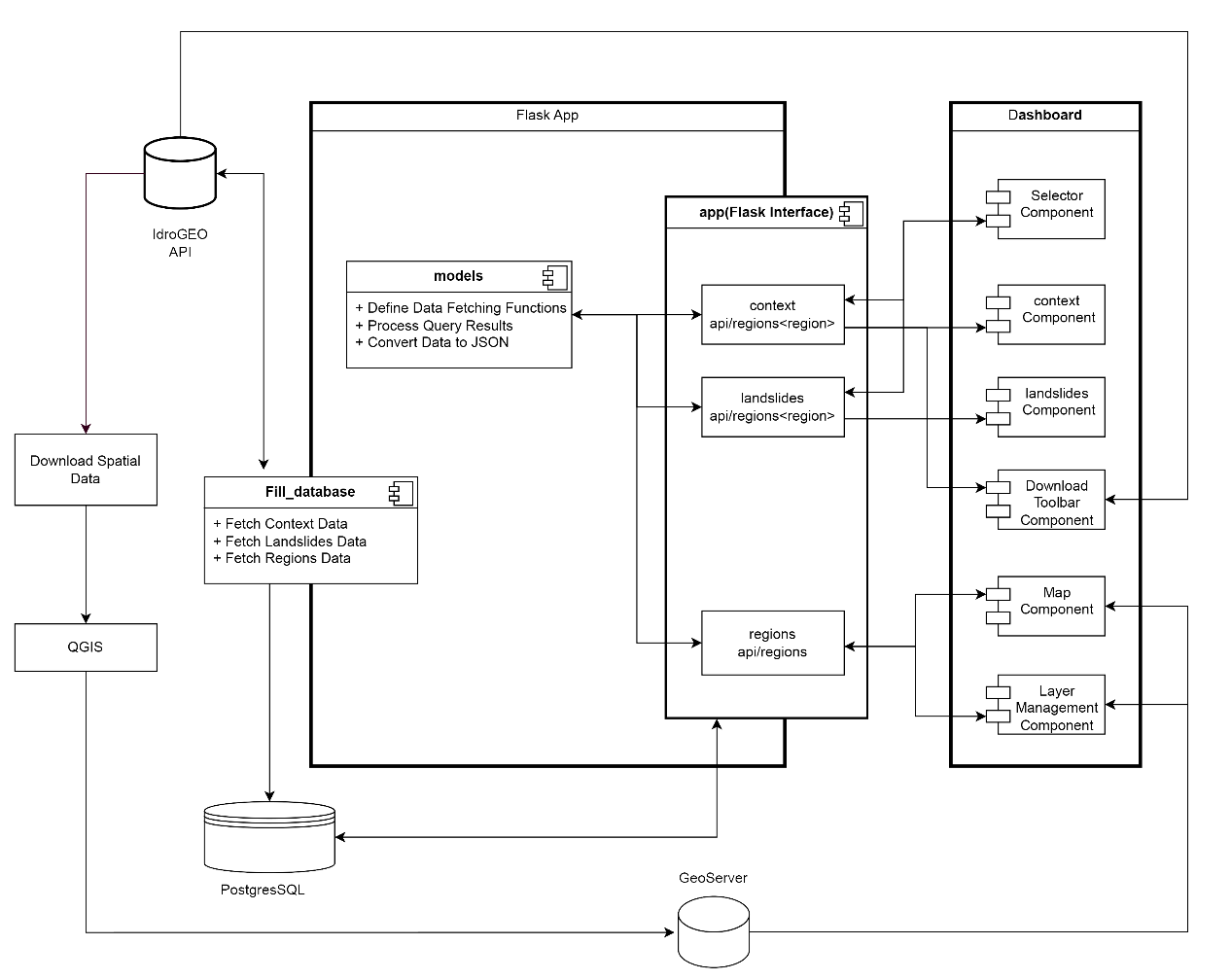
* 当前项目无法实现数据的实时集成功能，尽管未来可以作为进一步开发的方向。
* 仪表盘的设计和功能受限于Dash和相关图形库的技术能力。
* 由于分析主要集中在区域层面，可能不足以满足更小地理单元（如街区或社区）层面的数据展示和分析需求。

**2. Architectural Design**

**2.1. Overview**

本系统由三个主要组件构成：数据库、Web服务器（REST API）和交互式仪表盘（前端）。数据库负责存储和管理从ISPRA IdroGEO API获取的滑坡风险数据以及相关的统计信息。Web服务器通过REST API接口提供对数据库中数据的查询和检索服务。交互式仪表盘用于数据的可视化展示，支持用户在不同区域之间进行选择，并查看详细的统计信息和风险分析，进而实现交互式的数据分析和决策支持。

**2.2. Component Diagrams**

以下组件图展示了数据库、Web服务器和仪表盘之间的关系，以及各组件的交互和数据流动过程。

组件描述

IdroGEO API

功能: 提供滑坡风险和相关统计数据的API接口。

描述: 通过HTTP请求从ISPRA IdroGEO API获取滑坡风险数据和相关统计信息。

Download Spatial Data

功能: 提供空间数据的下载。

描述: 从外部来源下载空间数据，用于进一步的地理空间分析。

QGIS

功能: 处理和转换空间数据。

描述: 使用QGIS处理下载的空间数据，并通过GeoServer发布这些数据。

Fill\_database

功能: 从API获取数据并存储到数据库中。

描述: 通过fill\_database.py脚本从IdroGEO API获取统计数据、滑坡数据和区域数据，并存储到PostgreSQL数据库中。

Fetch Context Data: 获取统计数据。

Fetch Landslides Data: 获取滑坡数据。

Fetch Regions Data: 获取区域数据。

PostgreSQL with PostGIS

功能: 存储和管理地理空间数据。

描述: 使用PostgreSQL数据库扩展PostGIS来存储和管理地理空间数据。

GeoServer

功能: 发布地理空间数据。

描述: 通过GeoServer发布从QGIS处理的空间数据，提供地图图层服务。

Flask App

功能: 管理数据获取、处理和提供API接口。

描述: 作为系统的核心，负责数据的获取、处理和通过REST API提供数据服务。

Models:

Define Data Fetching Functions: 定义从数据库获取数据的函数。

Process Query Results: 处理查询结果。

Convert Data to JSON: 将数据转换为JSON格式。

API接口:

context api/regions<region>: 获取特定区域的统计数据。

landslides api/regions<region>: 获取特定区域的滑坡数据。

regions api/regions: 获取所有区域的数据。

Dashboard

功能: 数据的可视化展示和交互。

描述: 提供用户界面，用于选择不同区域并查看详细的统计信息和风险分析。

Selector Component: 允许用户选择不同的区域。

Context Component: 展示选定区域的统计数据。

Landslides Component: 展示选定区域的滑坡风险数据。

Download Toolbar Component: 提供数据下载功能。

Map Component: 显示滑坡风险地图和其他相关图层。

Layer Management Component: 允许用户管理和切换不同的地图图层。

**2.3. Technology Stack**

**数据库**

* **PostgreSQL**: 开源的对象关系型数据库管理系统，具有强大的扩展性和标准化的合规性，适用于复杂的数据管理任务。
* **PostGIS**: PostgreSQL的扩展，为其增加了对地理空间对象的支持，使其能够处理和存储地理信息系统(GIS)数据。

**Web服务器**

* **Flask**: 一个用Python编写的轻量级Web应用框架，采用简洁、灵活的设计，适合快速开发Web服务和REST API。
* **GeoServer**: 一个开源的服务器软件，用于发布地理空间数据。GeoServer设计用于与许多不同的数据源交互，并能输出数据到多种客户端，包括网页地图。

**数据处理**

* **Pandas**: 一个强大的Python数据分析库，提供高性能、易于使用的数据结构和数据分析工具，适用于各种数据处理任务。
* **GeoPandas**: 在Pandas基础上扩展的地理空间数据分析库，使Pandas支持地理数据类型的处理和分析。
* **Requests**: 一个简单易用的Python库，用于发送HTTP请求，适用于与API的交互和数据获取。
* **QGIS**: 一个开源的地理信息系统应用程序，用于处理和分析空间数据，并通过GeoServer发布数据。

**仪表盘**

* **Dash**: 一个用于构建分析型Web应用程序的Python框架，基于Flask和Plotly，适合快速开发交互式数据可视化应用。
* **Plotly**: 一个用于制作交互式图表的Python库，支持多种图表类型，包括散点图、折线图、柱状图等。
* **Leaflet**: 一个开源的JavaScript库，用于构建互动式地图应用，适用于显示和交互地理空间数据。
* **Dash Leaflet**: 将Leaflet集成到Dash中的扩展库，用于在Dash应用中创建和管理交互式地图。

**版本控制**

* **Git**: 一个分布式版本控制系统，用于跟踪代码变更和协作开发。
* **GitHub**: 基于Git的代码托管平台，提供版本控制和协作工具，适用于团队开发和代码管理。

**开发环境**

* **PyCharm**: 一个由JetBrains开发的专业Python集成开发环境，提供智能代码辅助、调试、集成版本控制和多种插件支持。

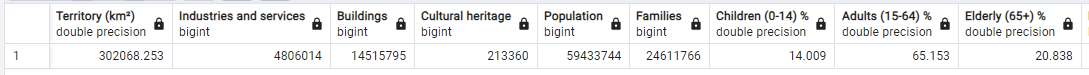
**3. Software Design**

**3.1. Database Design**

数据库将包括以下表：

 **regions**: 存储地理区域数据，包含区域的几何信息（geom）、区域名称（den\_reg）以及其他相关属性图形用户界面, 应用程序, 表格, Excel

描述已自动生成。

 **context\_data\_{id}**: 存储特定区域的统计数据，包括人口、家庭、建筑物、工业和服务、文化遗产及领土等信息。每个区域都有独立的表，id表示区域的唯一标识符。

 **landslides\_data\_{id}**: 存储特定区域的滑坡风险数据，按照不同风险类别分类。每个区域都有独立的表，id表示区域的唯一标识符。

图形用户界面, 应用程序

描述已自动生成

**数据表描述**

1. **regions 表**:
   * **字段**:
     + id: 主键，标识区域的唯一ID。
     + cod\_rip: 大区代码。
     + cod\_reg: 区域代码。
     + den\_reg: 区域名称。
     + shape\_leng: 区域边界长度。
     + shape\_area: 区域面积。
     + geom: 几何信息，用于存储区域的地理形状。
2. **context\_data\_{id} 表**:
   * **字段**:
     + Territory (km²): 领土面积。
     + Industries and services: 工业和服务数量。
     + Buildings: 建筑物数量。
     + Cultural heritage: 文化遗产数量。
     + Population: 人口数量。
     + Families: 家庭数量。
     + Children (0-14) %: 0-14岁儿童比例。
     + Adults (15-64) %: 15-64岁成年人比例。
     + Elderly (65+) %: 65岁及以上老人比例。
3. **landslides\_data\_{id} 表**:
   * **字段**:
     + Category: 风险类别（如非常高风险P4、高风险P3、中等风险P2、低风险P1、注意区AA）。
     + Territory: 受影响的领土面积（及其占总面积的百分比）。
     + Population: 受影响的人口数量（及其占总人口的百分比）。
     + Families: 受影响的家庭数量（及其占总家庭数的百分比）。
     + Buildings: 受影响的建筑物数量（及其占总建筑物数的百分比）。
     + Industries and services: 受影响的工业和服务数量（及其占总数的百分比）。
     + Cultural heritage: 受影响的文化遗产数量（及其占总数的百分比）。
4. 数据库ER图 这个图重新画！！！



**3.2. REST APIs**

REST API 提供了以下端点，以便客户端应用程序能够与服务器进行通信，从数据库中获取和处理数据：

1. **GET /context/string:id**
   * **描述**: 获取特定区域的统计数据。
   * **请求参数**:
     + id: 区域的唯一标识符（字符串格式）。
   * **响应**: 返回指定区域的统计数据，包括人口、家庭、建筑物、工业和服务、文化遗产及领土等信息。
   * **代码**:
   * @app.route("/context/string:id", methods=["GET"])
   * def get\_context\_endpoint(id):
   * table\_name = f'context\_data\_{id}'
   * return get\_context(conn=conn, table\_name=table\_name)
   * **models.py 中的函数**:
   * def get\_context(conn, table\_name):
   * df = pd.read\_sql\_table(table\_name=table\_name, con=conn) df\_json = df.to\_json(orient="records")
   * return df\_json
2. **GET /landslides/string:id**
   * **描述**: 获取特定区域的滑坡风险数据。
   * **请求参数**:
     + id: 区域的唯一标识符（字符串格式）。
   * **响应**: 返回指定区域的滑坡风险数据，包括不同风险类别的数据，如非常高风险（P4）、高风险（P3）、中等风险（P2）、低风险（P1）和注意区（AA）。
   * **代码**:
   * @app.route("/landslides/string:id", methods=["GET"])
   * def get\_landslides\_endpoint(id):
   * table\_name = f'landslides\_data\_{id}'
   * return get\_landslides(conn=conn, table\_name=table\_name)
   * **models.py 中的函数**:

def get\_landslides(conn, table\_name):

df = pd.read\_sql\_table(table\_name=table\_name, con=conn) df\_json = df.to\_json(orient="records")

return df\_json

1. **GET /regions**
   * **描述**: 获取所有区域的数据。
   * **请求参数**: 无。
   * **响应**: 返回所有区域的基本信息和几何数据。
   * **代码**:
   * @app.route("/regions", methods=["GET"])
   * def get\_regions\_endpoint():
   * return get\_regions(conn=conn)
   * **models.py 中的函数**:
   * def get\_regions(conn):
   * df = pd.read\_sql\_table(table\_name='regions', con=conn) df\_json = df.to\_json(orient="records")
   * return df\_json

**3.3. Dashboard Design**

仪表盘在灾害管理和决策支持中扮演着至关重要的角色。本系统的仪表盘设计旨在通过交互式数据可视化和直观的用户界面，提升滑坡风险分析和管理的效率。以下是仪表盘的主要功能及其详细描述：

仪表盘的布局图如下所示：

地图

描述已自动生成

初始界面

地图

描述已自动生成

用户选择某一个区域的界面

**仪表盘主要功能**

1. **地图视图**
   * **描述**: 显示滑坡风险地图，用户可以选择不同的区域查看详细信息。地图视图基于Leaflet和Dash Leaflet构建，提供高度交互式的地理数据展示。
   * **功能**:
     + **区域选择与高亮显示**: 用户可以点击地图中的区域以高亮显示该区域的边界，并查看详细信息。
     + **图层切换**: 提供多种地图图层，用户可以切换查看不同类型的数据，包括滑坡风险图层、人口密度图层、工业和服务分布图层等。
     + **地图缩放与平移**: 支持用户自由缩放和平移地图，以便详细查看特定区域的地理信息。
   * **技术实现**:
     + 采用Leaflet库构建基础地图视图，结合Dash Leaflet实现与Dash框架的无缝集成。
     + 使用GeoJSON格式的数据进行区域边界绘制和高亮显示。
2. **数据展示**
   * **描述**: 以图表和数据卡片形式展示选定区域的统计数据和滑坡数据。数据展示部分基于Dash和Plotly构建，提供多样化的图表和直观的数据卡片。
   * **功能**:
     + **统计数据展示**: 显示选定区域的详细统计数据，包括领土面积、人口、家庭数量、建筑物数量、工业和服务数量、文化遗产数量、儿童比例、成年人比例和老年人比例等。
     + **滑坡风险数据展示**: 展示选定区域的滑坡风险数据，涵盖不同风险等级（如非常高风险P4、高风险P3、中等风险P2、低风险P1和注意区AA）的受影响面积、人口、家庭、建筑物、工业和服务及文化遗产数量。
     + **数据图表**: 提供堆叠柱状图，直观展示滑坡风险数据的详细分类及其比例。
   * **技术实现**:
     + 使用Dash框架构建交互式数据展示界面。
     + 利用Plotly库创建多种类型的图表，实现数据的可视化。
     + 数据从后台通过REST API获取，并在前端进行动态渲染。
3. **下载功能**
   * **描述**: 允许用户下载特定区域的数据，便于离线分析和报告制作。下载功能基于Dash的下载组件实现，确保数据获取的便捷性和灵活性。
   * **功能**:
     + **数据选择与下载**: 用户可以选择下载选定区域的统计数据和滑坡风险数据，下载的数据文件格式为CSV，便于进一步处理和分析。
     + **下载按钮**: 在数据展示界面提供下载按钮，用户点击后即可获取相关数据文件。
   * **技术实现**:
     + 使用Dash的dcc.Download组件实现数据下载功能。
     + 后端提供数据接口，生成CSV文件并响应前端请求。

通过以上功能和布局设计，仪表盘将为用户提供一个直观、易用的界面，支持滑坡风险的分析和决策。用户可以通过地图视图、数据展示和下载功能，全面了解和分析特定区域的滑坡风险和相关统计数据。

**4. Implementation and Test Plan**

**4.1. Implementation Plan**

**Hazard Maps Project**

This project aims to provide hazard maps for disaster management using Python, Flask,GeoServer, and PostgreSQL with PostGIS extension. The following instructions will guide you through setting up and running the project.

**Prerequisites**

* Python 3.8+
* PostgreSQL 12+
* Node.js and npm (for frontend dependencies)
* pgAdmin (for database management)
* GeoServer (for map layer publishing)

**Setup Instructions**

1. **Clone the repository**

First, clone the repository to your local machine:

git clone https://github.com/tanxiaoo/Hazard\_Maps\_Project-SE4GEO.git

cd Hazard\_Maps\_Project-SE4GEO/APP

1. **Download large files**

Due to GitHub's file size limitations, the spatial\_datafolders contain large files that need to be downloaded separately. Please download the necessary files from the following links and place them in the backend/resources/geoserver/spatial\_datafolders:

* + [Download spatial\_data folder](https://polimi365-my.sharepoint.com/:f:/g/personal/10963273_polimi_it/EtD9xxptYb9Ai5hjSZd94dABqmt5mTafrHdKn9_7OXogUg?e=NJYVyi)

1. **Create the database and enable the PostGIS extension**

In pgAdmin:

* + Open pgAdmin and connect to your PostgreSQL server.
  + Create a database named se4g24.
  + Select the newly created database, open the query tool, and run the following command to enable the PostGIS extension:

CREATE EXTENSION postgis;

* + Run the fill\_database.py file to populate the database:
  + cd backend

python fill\_database.py

1. **Configure GeoServer**
   * Download and install GeoServer:
     + Visit the [GeoServer download page](http://geoserver.org/download/) and download the latest version of GeoServer.
     + Follow the installation instructions.
   * Start GeoServer and access the GeoServer Web Interface at:
   * http://localhost:8080/geoserver
   * Publish layers and styles:
     + In the left menu, select Data -> Layers.
     + Click Add new Layer, select the data store, and choose the data from backend/resources/geoserver/spatial\_data folder, then follow the prompts to complete the layer publishing.
     + In the left menu, select Style -> SLD.
     + Click Add a new style, upload the style files from backend/resources/geoserver/SLD directory, and apply them to the corresponding layers.
2. **Create a .env file**

Create a .env file in the APP directory and add the following content:

DATABASE\_URL=postgresql://postgres:yourpassword@localhost:5432/se4g24

GEOSERVER\_URL=http://localhost:8080/geoserver

Ensure to replace yourpassword with your PostgreSQL password.

1. **Install Python dependencies**

Ensure you are in the APP directory, then run the following commands to create and activate a virtual environment and install the required Python packages:

python -m venv venv

source venv/bin/activate # For Windows, use `venv\Scripts\activate`

pip install -r requirements.txt

1. **Install Node.js dependencies**

In the APP directory, install the Node.js dependencies:

npm install

1. **Run the Flask application**

In the APP directory, run the Flask application:

cd backend

python app.py

1. **Run the frontend dashboard**

In the APP directory, run the frontend dashboard:

cd frontend

python dashboard.py

按照上述步骤操作，您将能够正确设置和运行该项目。如果在过程中遇到任何问题，请参考项目文档或与项目开发团队联系以获取帮助

**4.2. Test Plan**

**Unit Testing**

* 编写单元测试覆盖各个模块，确保每个功能的正确性。
* 测试数据库连接、API端点、数据处理函数等。

**Integration Testing**

* 测试模块之间的集成，确保数据在不同组件之间的正确流动。
* 测试数据库填充脚本、API查询和仪表盘数据展示。

**User Testing**

* 邀请用户试用仪表盘，收集反馈并进行改进。
* 测试用户交互、数据可视化效果和下载功能。

**5. Bibliography**

* ISPRA IdroGEO API Documentation: <https://developers.italia.it/it/api/ispra-idrogeo.html>
* GeoPandas Documentation: <https://geopandas.org/>
* Flask Documentation: <https://flask.palletsprojects.com/en/2.2.x/>
* Dash Documentation: https://dash.plotly.com/
* GeoServer Documentation: <http://geoserver.org/>
* PostgreSQL Documentation: <https://www.postgresql.org/docs/>