POLITECNICO
MILANO 1863

DESIGN DOCUMENT

IDK

Marco Ambrosanio

Amirhossein Donyadidegan

Alessandro Violi

Hangyun Wang

DESIGN DOCUMENT IDK

Index

1) INTRODUCTION

PURPOSE OF OUR PROJECT

As technology advances, people have increasingly diverse ways of accessing information, and the volume of accessible information continues to grow. The inundation of information poses a challenge for ordinary citizens to discern accurate information, while experts encounter difficulties in efficiently utilizing those informations in their work.

During the XXV Winter Olympic Games, there is a significant need to provide high-quality information on landslide and floods to facilitate the safety and enjoyment of the event for tourists, citizens, and other general users. In response to this, we have developed a user-friendly data visualization and data querying functionality tailored to the specific needs and characteristics of the users. This streamlined system aims to ensure that individuals can easily access reliable information about potential hazards, enabling them to make informed decisions and navigate the Olympic environment safely.

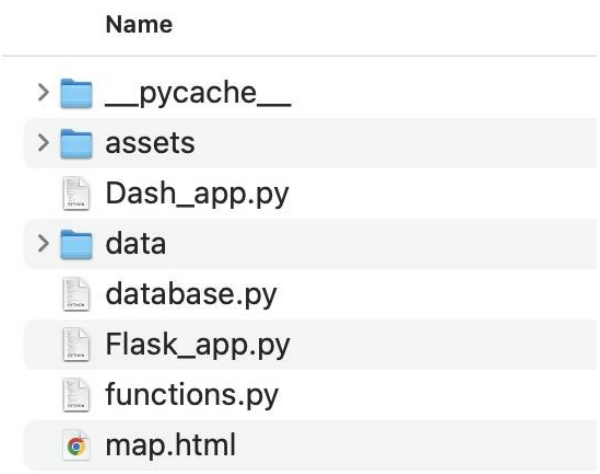
For our expert users, we have further empowered them with the ability to visualize data and access data. Expert users have the opportunity to delve into datasets, extract valuable insights, and arrive at strategic conclusions with higher precision and efficiency. This approach not only streamlines their workflow but also provides them with the necessary resources to tackle the inherent complexities of their respective fields, enabling them to optimize decision-making processes and drive impactful outcomes.

SCOPE

The application is specifically tailored to offer various degrees of data visualization and data accessibility to diverse user groups, including non-registered users, general users, and experts. Its primary objective is to educate these users about the occurrences of floods and landslides in their locality. This is achieved through the integration of interactive maps and the comprehensive analysis of historical data. By employing these features, the application strives to enhance user comprehension of potential hazards and empower them to make well-informed decisions regarding precautionary measures for ensuring safety. For example, non-registered users may be provided with basic visualizations and general information, while experts could have access to more detailed and specialized data analysis tools. This differentiation in access levels ensures that the application caters to the specific needs and expertise levels of its user base.

2) SYSTEM OVERVIEW

Architecture



- **__pycache__**: Contains compiled Python files.
- **assets**: Stores custom stylesheets and other static assets.
- **Dash_app.py**: Main application file for the Dash app.
- **data**: Used for storing datasets.
- **database.py**: Script for setting up the database.
- **Flask_app.py**: Contains the Flask server setup.
- **functions.py**: Contains helper functions for data retrieval and processing.
- **map.html**: HTML file for rendering the map.

Components

- **Dash**: Used for building the interactive web application.
- **Flask**: Serves as the web server for the application.
- **PostgreSQL**: Relational database management system.
- **PostGIS**: Extension for PostgreSQL to handle geospatial data.
- **Leaflet.js**: JavaScript library for interactive maps.
- **Folium**: Python library to create Leaflet maps.
- **Geopandas**: Library for geographic data manipulation.

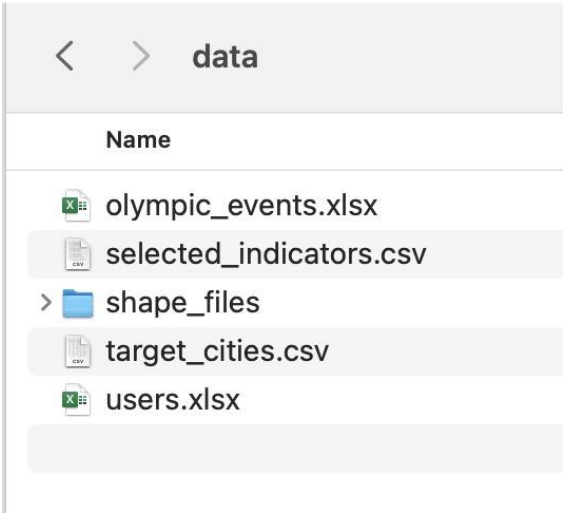
Selected indicators

The database is based on the use of various tables that contain information about the cities and the related events where the Milan and Cortina Winter Olympics will be held. In addition to this information, some of the indicators derived from IdrogeoAPI are considered useful in defining the hydrogeological risk of these cities.

Among the selected indicators are:

- **ar_kmq**: Area of the city in km²
- **ar_id_p3**: Surface area of high flood risk (km²)
- **ar_id_p2**: Surface area of medium flood risk (km²)
- **ar_id_p1**: Surface area of low flood risk (km²)
- **pop_idr_p3**: Population at high flood risk
- **pop_idr_p2**: Population at medium flood risk
- **pop_idr_p1**: Population at low flood risk
- **fam_idr_p3**: Families at high flood risk
- **fam_idr_p2**: Families at medium flood risk
- **fam_idr_p1**: Families at low flood risk
- **ed_idr_p3**: Buildings at high flood risk
- **ed_idr_p2**: Buildings at medium flood risk
- **ed_idr_p1**: Buildings at low flood risk
- **im_idr_p3**: Local business units at high flood risk
- **im_idr_p2**: Local business units at medium flood risk
- **im_idr_p1**: Local business units at low flood risk

3) DATABASE DESIGN



In the project, we set up a database to store selected city location and other historical flood situations, landslide data and so on. The database employs PostgreSQL for managing relational data and PostGIS for handling geospatial data.

Table	Usage	Primary Key
Target_cities	Stores coordinates for cities	UID INTEGER
Olympic events	Stores information about Olympic events	fid INTEGER
Users	Storing registered user information	User name TEXT
Indicators	Stores the indicator data	Indicator TEXT

TARGET CITIES

The table is designed to store comprehensive information about cities within Italy, including their geographical identifiers, location, and various indicators. The table leverages parameters provided by the IDROGEOapi portal, ensuring standardized identification and consistent data integration.

It contains:

- the city name and codes(rip,reg,prov)
- lat,lon(calculated from python codes)
- geometry(polygon) came from Com01012024_g_WGS84.shp
- indicators data(came from api url)

More in details, the table's columns are as follows:

- **cod_rip**: Zone identifier
- **cod_prov**: Province identifier
- **cod_reg**: Region identifier
- **geometry**: Polygon data from Com01012024_g_WGS84.shp
- **uid**: Unique identifier
- **name**: City name
- **ar_id_p3**, **ar_id_p2**, **ar_id_p1**: Hydraulic hazard areas
- **pop_idr_p3**, **pop_idr_p2**, **pop_idr_p1**: Population in hydraulic hazard areas
- **lat**, **lon**: Coordinates

	uid bigint	name text	ar_id_p3 double precision	ar_id_p2 double precision	ar_id_p1 double precision	pop_idr_p3 double precision	pop_idr_p2 double precision	pop_idr_p1 double precision	fam_idr_p3 double precision	fam_idr_p2 double precision	fam_idr_ double p
1	1272	Torino	4.711	10.163	20.893	1365	18287	80857	607	8550	
2	3106	Novara	8.113	12.807	32.061	522	733	8794	200	291	
3	16024	Bergamo	1.057	1.146	1.782	2635	2640	4050	1225	1227	
4	5005	Asti	18.005	19.259	29.345	869	1109	16043	336	421	
5	6003	Alessandria	34.441	36.279	67.922	644	667	21311	278	288	
6	10025	Genova	6.549	9.027	11.453	48546	78207	101864	24382	39490	
7	11015	La Spezia	0.955	2.138	2.162	5603	17755	17775	2674	8399	
8	12026	Busto Arsizio	0.018	0.041	0.339	0	0	46	0	0	
9	12133	Varese	5.522	6.049	7.299	254	348	799	104	147	
10	13075	Como	2.435	2.872	3.422	372	2996	6654	177	1503	
11	14009	Bormio	3.135	3.148	4.974	120	120	2213	50	50	
12	18110	Pavia	12.029	12.031	13.686	1524	1524	4752	714	714	
13	61022	Caserta	1.5	1.532	1.532	2410	2413	2413	886	888	
14	88009	Ragusa	0.397	0.397	0.409	0	0	0	0	0	
15	88012	Vittoria	2.764	2.764	2.764	61	61	61	29	29	

Total rows: 93 of 93 Query complete 00:00:00.173 Ln 1, Col 1

OLYMPIC EVENTS

This table provides information about the events' venues and games.

- CITY is the municipality where the event takes place
- VENUE is the place of the event
- SPORT is the game of the event
- city_uid is the unique identifier of the municipality

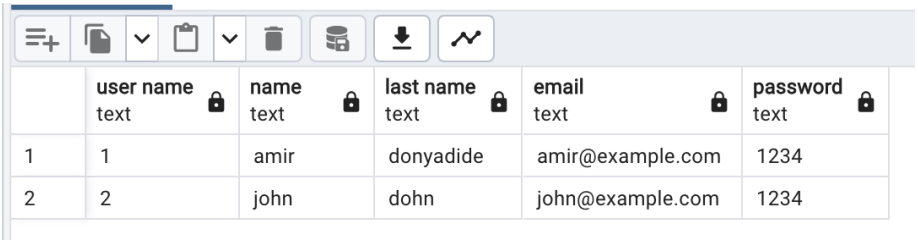
	fid bigint	CITY text	VENUE text	SPORT text	city_uid bigint
1	1	Milano	Stadio Giuseppe Meazza/San Siro	Opening Ceremony	15146
2	2	Milano	Palaltalia	Ice Hockey	15146
3	3	Assago	Mediolanum Forum	Figure Skating	15011
4	4	Assago	Mediolanum Forum	Short Track	15011
5	5	Rho	Rho Fiera Milano	Speed Skating	15182
6	6	Rho	Rho Fiera Milano	Ice Hockey	15182
7	7	Bormio	Stelvio	Men's Alpine Skiing	14009
8	8	Bormio	To be defined	Ski Mountaineering	14009
9	9	Livigno	Mottolini/Sitas, Tagliede/Carosello 3000	Freestyle	14037
10	10	Livigno	Mottolini/Sitas, Tagliede/Carosello 3000	Snowboarding	14037
11	11	Predazzo	Stadio del salto Giuseppe Del Ben	Ski Jumping	22147
12	12	Predazzo	Stadio del salto Giuseppe Del Ben	Nordic Combined	22147
13	13	Tesero	Centro del fondo e del biathlon Fabio Canal	Cross-Country Skiing	22196
14	14	Tesero	Centro del fondo e del biathlon Fabio Canal	Nordic Combined	22196
15	16	Cortina d'Ampezzo	Olimpia delle Tofane	Women's Alpine Skiing	25016
16	17	Cortina d'Ampezzo	Stadio olimpico del ghiaccio	Curling	25016
17	18	Cortina d'Ampezzo	To be defined	Bobsleigh	25016
18	19	Cortina d'Ampezzo	To be defined	Skeleton	25016
19	20	Cortina d'Ampezzo	To be defined	Luge	25016
20	21	Verona	Arena di Verona	Closing Ceremony	23091

USER

The table below provides information about the user. In particular, the user_type column specifies if the user is an expert or a general user.

Columns:

- **username**: User identifier
- **name**: User's first name
- **last_name**: User's last name
- **email**: User's email
- **password**: User's password

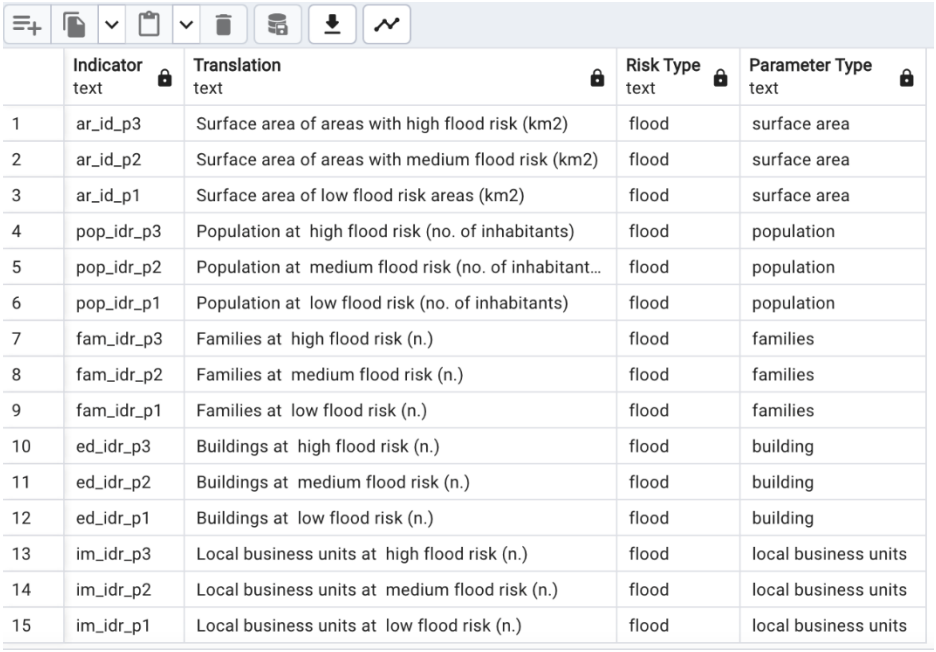


	user name text	name text	last name text	email text	password text
1	1	amir	donyadide	amir@example.com	1234
2	2	john	dohn	john@example.com	1234

INDICATORS TABLE

Columns:

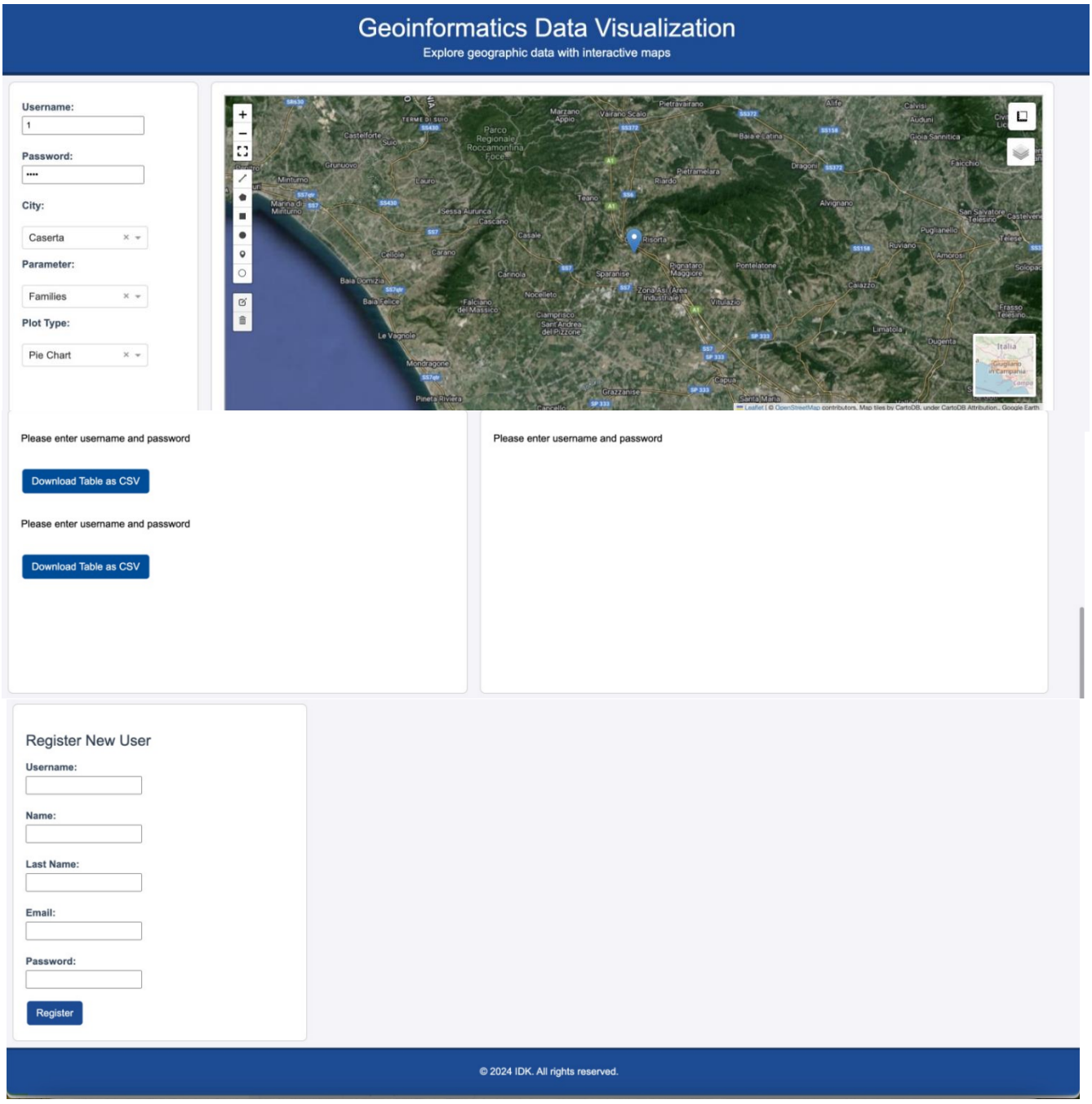
- **indicator_text**: Indicator identifier
- **translation_text**: Description of the indicator
- **risk_type**: Type of risk (flood, landslide)
- **parameter_type**: Parameter associated with the indicator



	Indicator text	Translation text	Risk Type text	Parameter Type text
1	ar_id_p3	Surface area of areas with high flood risk (km2)	flood	surface area
2	ar_id_p2	Surface area of areas with medium flood risk (km2)	flood	surface area
3	ar_id_p1	Surface area of low flood risk areas (km2)	flood	surface area
4	pop_idr_p3	Population at high flood risk (no. of inhabitants)	flood	population
5	pop_idr_p2	Population at medium flood risk (no. of inhabitant...	flood	population
6	pop_idr_p1	Population at low flood risk (no. of inhabitants)	flood	population
7	fam_idr_p3	Families at high flood risk (n.)	flood	families
8	fam_idr_p2	Families at medium flood risk (n.)	flood	families
9	fam_idr_p1	Families at low flood risk (n.)	flood	families
10	ed_idr_p3	Buildings at high flood risk (n.)	flood	building
11	ed_idr_p2	Buildings at medium flood risk (n.)	flood	building
12	ed_idr_p1	Buildings at low flood risk (n.)	flood	building
13	im_idr_p3	Local business units at high flood risk (n.)	flood	local business units
14	im_idr_p2	Local business units at medium flood risk (n.)	flood	local business units
15	im_idr_p1	Local business units at low flood risk (n.)	flood	local business units

4) WEBSITE INTERFACE AND STRUCTURE

Home page for a non-registered user:



DESCRIPTION:

Top left Section: User Interaction Panel

- **Login Section:**
  - **Position:** Top-left of the screen.
  - **Components:**
    - **Username Field:** A text input field where users can enter their username.
    - **Password Field:** A password input field to ensure secure entry of user passwords.
- **Search Bar:**
  - **Position:** Below the login section.
  - **Components:**
    - City selection field.
    - Parameters selection field**.**
    - Plot type field

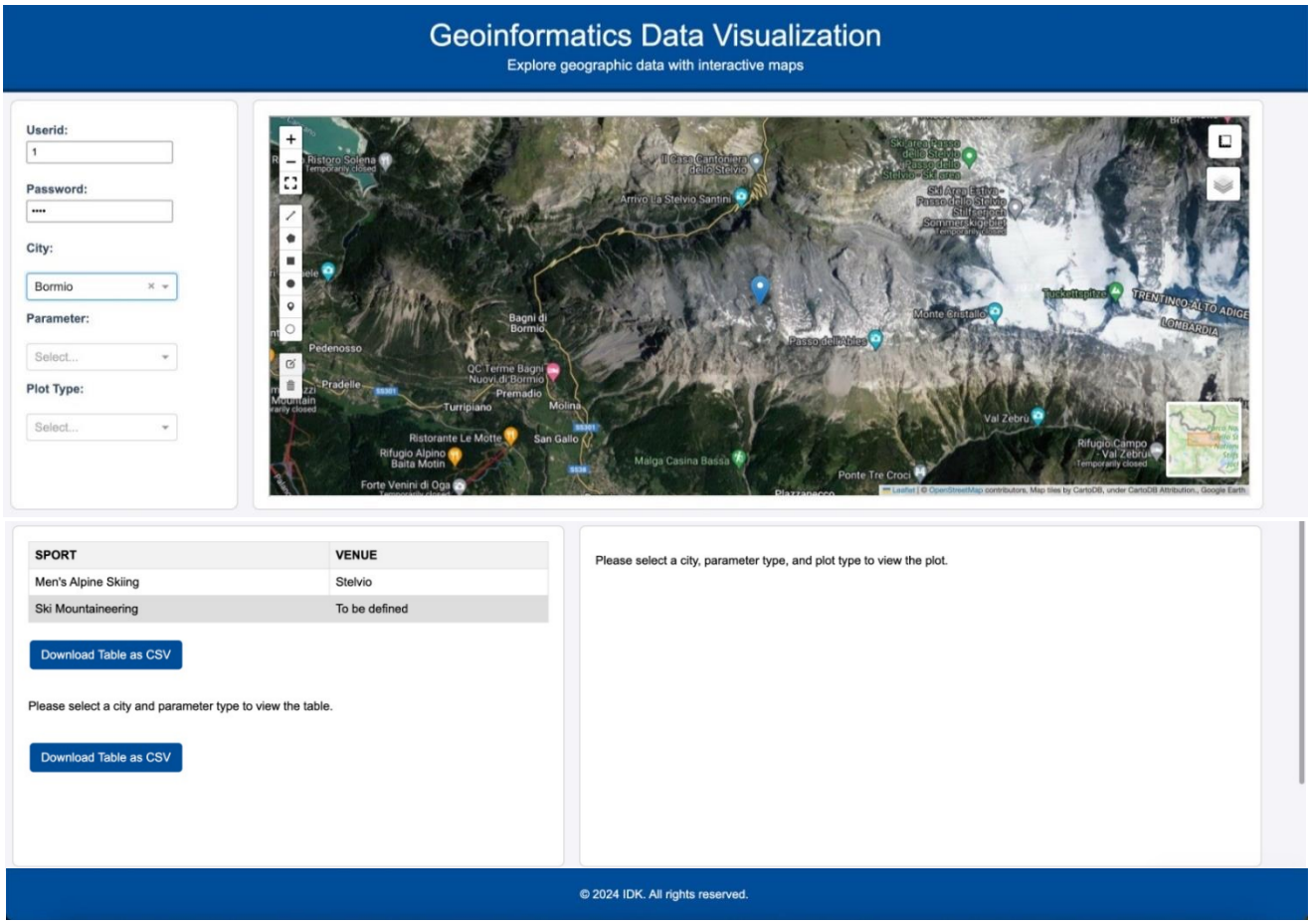
Top right Section: Interactive Map

- **Position:** Occupies the majority of the screen to the right.
- **Components:**
  - **Interactive Map:** A dynamic map interface that displays geographical data about cities in Italy.
    - **Zoom Controls:** Buttons to zoom in and out of the map for better navigation.
    - **Pan Controls:** Allows users to move the map view to different areas.
    - **Markers/Highlights:** Visual indicators on the map showing specific cities or points of interest.

Mid left: unavailable options because the user is not logged in.

Bottom: Registering section

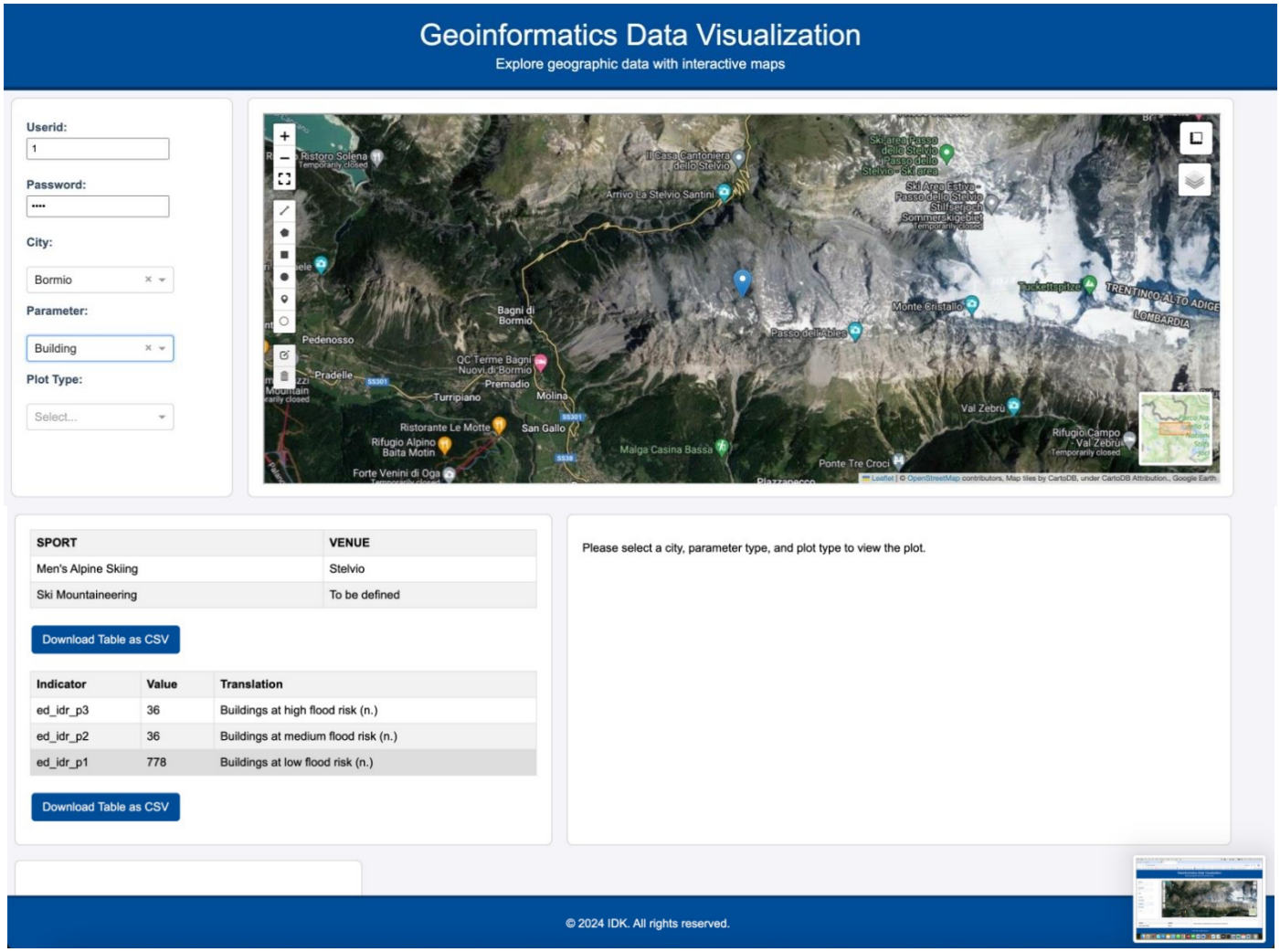
Home page for a registered user after a city selection:



- On the map it pops up a label with the selected city

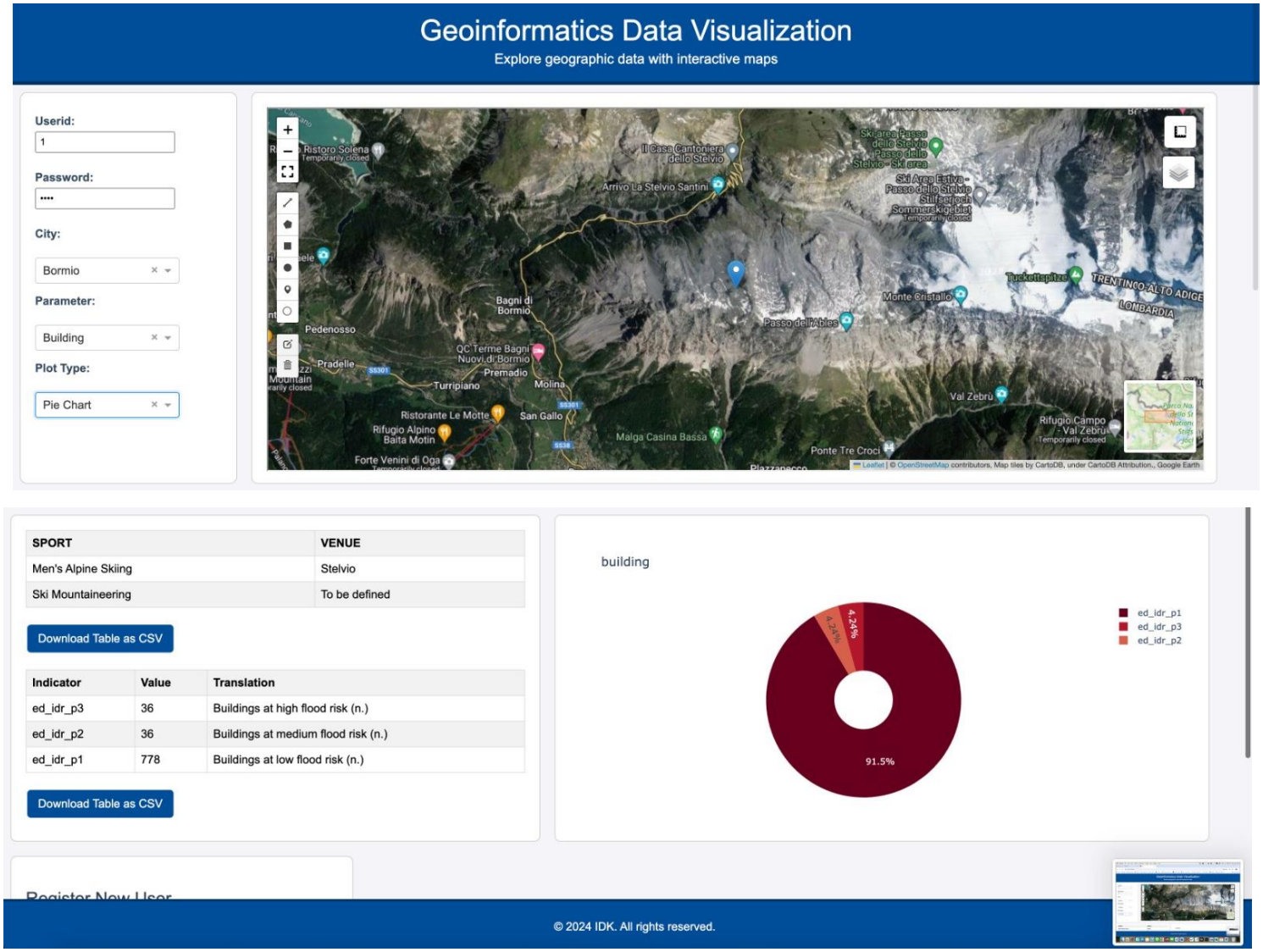
- in the mid-section there are the events of the selected city with the venues.

Home page for a registered user after a city selection and a parameter selection:



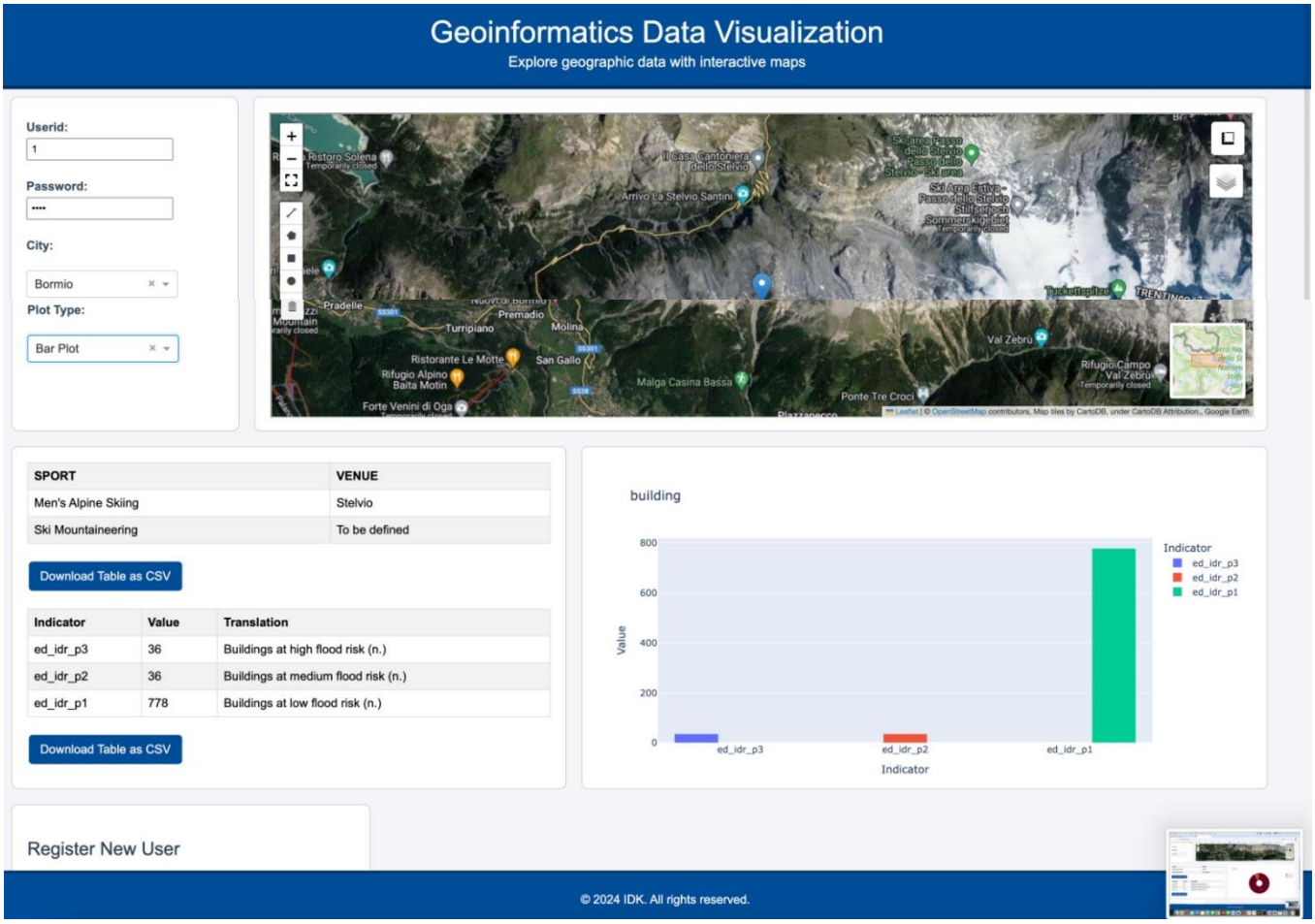
- after the parameter selection in the mid-section it pops up a table related to the selected parameter with the indicators (code and description) and the corresponding values.

Home page for a registered user after a city selection, a parameter selection and a plot type selection (Pie chart):

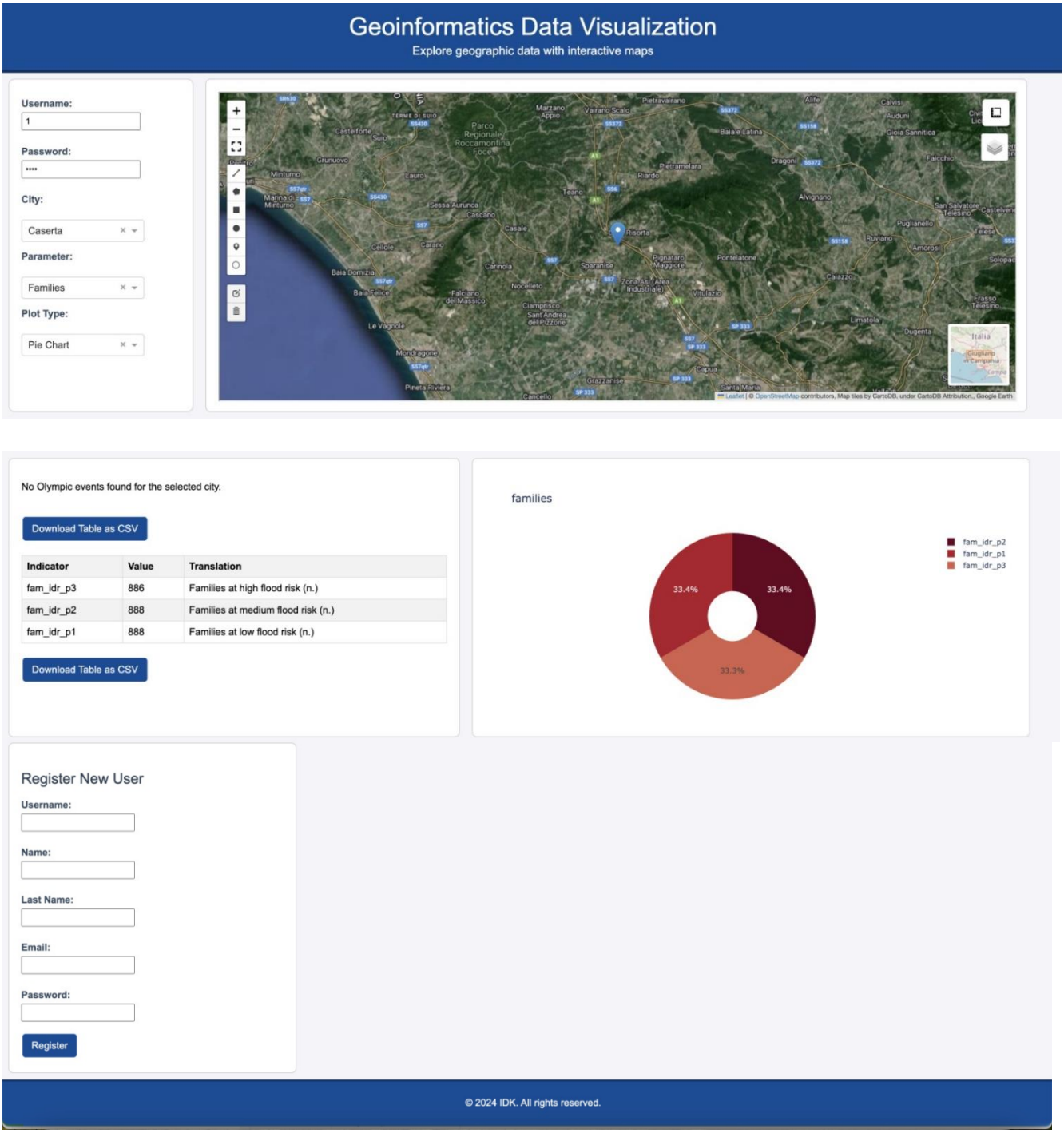


- the selected plot type gives a picture of the indicators: in the example the pie-chart describes what's the distribution of flood risk and it suggests that buildings of Bormio are more at high flood risk instead of medium/low.

Home page for a registered user after a city selection, a parameter selection and another plot type selection (Bar plot):



Home page for a registered user after a city selection with the city not hosting Olympic events:



5) USER CASES

1. **Register**
    - **ID Name:** DD1
    - **User:** Anyone who has not yet registered
    - **Input:** Go to the Home Page, click on Register, and follow the instructions
    - **Actions:** Enter first and last name, e-mail, password and select the type of user you want to be (generic or expert), click on Register and you will be redirected to the Log In page
2. **Log In**
    - **ID Name:** DD2
    - **User:** Anyone who is already registered
    - **Input:** Click on Log In
    - **Actions:** Enter your name and password and click on Log In; you will be redirected to the Home Page. If the password is incorrect, you will remain on the Log In page.
3. **Log Out**
    - **ID Name:** DD3
    - **User:** Anyone who has logged in
    - **Input:** Click on Log Out
    - **Actions:** Clicking on Log Out will exit the personal section and return you to the Home Page.
4. **Query Data**
    - **ID Name:** DD4
    - **User:** Anyone who is registered
    - **Input:**
        - **Cities:** The user can choose one or more cities from those hosting the XXVI Winter Olympics by selecting flags on the map or from the Selection Section.
        - **Indicators:** The user decides which data to analyze: landslide data, flood risk area data and population data.
    - **Actions:** After choosing the data to view, the user can display it in two different ways: through specific maps concerning the selected cities or through tables presenting the basic statistics of the searched data.
5. **Data Visualization**
    - **ID Name:** DD5
    - **User:** Anyone who has logged in
    - **Input:** By selecting one or more items from the Selection Section, the user can view the data of interest through an interactive map or an attribute table.
    - **Actions:** The user can decide on which information to focus regarding landslide and flood events in the cities hosting the Winter Olympics 2026 through the Selection Section.

6) ENDPOINTS

API Endpoints

1. **Get All Cities**
   - URL: /api/cities
   - Method: GET
   - Description: Returns a list of all cities in JSON format.
2. **Get City by Name and Parameter**
   - URL: /api/cities/<parameter>/<name>
   - Method: GET
   - Description: Retrieves details of a specific city based on the provided name and parameter.
3. **Get Olympic Events by City UID and Parameter**
   - URL: /api/olympic_events/<parameter>/<city_uid>
   - Method: GET
   - Description: Returns Olympic events for a specific city based on the provided UID and parameter.
4. **Add a New User**
   - URL: /api/user
   - Method: POST
   - Description: Adds a new user to the database and returns a success message in JSON format.

Flask Code Explanation

Imports and Initial Configuration

```
from flask import Flask, jsonify, request  # Lines 1-2
import psycopg2  # Lines 3-4
import logging
from psycopg2 import sql

app = Flask(__name__)  # Line 7
logging.basicConfig(level=logging.DEBUG)  # Line 8
```

This section imports the necessary modules and configures the Flask app with the logging level set to DEBUG.

Database Connection Function

```
def get_db_connection():  # Line 10
    try:
        conn = psycopg2.connect(  # Lines 11-15
            host="localhost",
            database="SE4G",
            user="postgres",
            password="Amir0440935784"
        )
        return conn  # Line 16
    except psycopg2.Error as e:
        app.logger.error(f"Error connecting to the database: {e}")  # Lines 17-18
        return None  # Line 19
```

This function attempts to connect to the PostgreSQL database and returns the connection object if successful; otherwise, it logs an error and returns **`None`**.

Endpoint to Get All Cities

```
@app.route('/api/cities', methods=['GET'])  # Line 21
def get_all_cities():  # Line 22
    conn = get_db_connection()  # Line 23
    if not conn:  # Lines 24-25
        return jsonify({'error': 'Database connection failed'}), 500

    try:
        with conn.cursor() as cur:  # Line 27
            cur.execute('SELECT * FROM public."CITY"')  # Line 28
            cities = [dict(zip([desc[0] for desc in cur.description], row)) for row in
cur.fetchall()]  # Line 29
            return jsonify(cities)  # Line 30
    except psycopg2.Error as e:
        app.logger.error(f"Error fetching cities: {e}")  # Lines 31-32
        return jsonify({'error': f"Error fetching cities: {e}"}), 500
    finally:
        conn.close()  # Line 33
```

This endpoint responds to GET requests to retrieve all cities from the "CITY" table and returns the results in JSON format.

Endpoint to Get a City by Name and Specific Parameter

```
@app.route('/api/cities/<string:parameter>/<string:name>', methods=['GET'])  # Line 35
def get_city_by_name_and_parameter(parameter, name):  # Line 36
    conn = get_db_connection()  # Line 37
    if not conn:  # Lines 38-39
        return jsonify({'error': 'Database connection failed'}), 500

    try:
        with conn.cursor() as cur:  # Line 41
            cur.execute(sql.SQL('SELECT    {}    FROM    public."CITY"    WHERE    name
= %s').format(sql.Identifier(parameter)), (name,))  # Line 42
            city = [dict(zip([desc[0] for desc in cur.description], row)) for row in
cur.fetchall()]  # Line 43
            return jsonify(city)  # Line 44
    except psycopg2.Error as e:
        app.logger.error(f"Error fetching city: {e}")  # Lines 45-46
        return jsonify({'error': f"Error fetching city: {e}"}), 500
    finally:
        conn.close()  # Line 48
```

This endpoint responds to GET requests to retrieve a specific city based on the name and specified parameter, and returns the results in JSON format.

Endpoint to Get Olympic Events by City UID and Specific Parameter

```
@app.route('/api/olympic_events/<string:parameter>/<string:city_uid>',  methods=['GET'])
# Line 50
def get_olympic_events_by_city_uid_and_parameter(parameter, city_uid):  # Line 51
    conn = get_db_connection()  # Line 52
    if not conn:  # Lines 53-54
        return jsonify({'error': 'Database connection failed'}), 500

    try:
        with conn.cursor() as cur:  # Line 56
            query = sql.SQL('SELECT {field} FROM public."OLYMPIC_EVENTS" WHERE "city_uid"
= %s').format(field=sql.Identifier(parameter))  # Line 57
            cur.execute(query, (city_uid,))  # Line 58
            olympic_events = [dict(zip([desc[0] for desc in cur.description], row)) for
row in cur.fetchall()]  # Line 59
            return jsonify(olympic_events)  # Line 60
    except psycopg2.Error as e:
        app.logger.error(f"Error fetching olympic events: {e}")  # Lines 61-62
        return jsonify({'error': f"Error fetching olympic events: {e}"}), 500
```

```
    finally:
        conn.close()  # Line 64
```

This endpoint responds to GET requests to retrieve Olympic events based on the city's UID and the specified parameter, and returns the results in JSON format.

Endpoint to Add a New User

```
@app.route('/api/user', methods=['POST'])  # Line 66
def add_user():  # Line 67
    conn = get_db_connection()  # Line 68
    if not conn:  # Lines 69-70
        return jsonify({'error': 'Database connection failed'}), 500

    try:
        data = request.json  # Line 72
        with conn.cursor() as cur:  # Line 73
            cur.execute('INSERT INTO public."USER" (username, name, last_name, email,
password) VALUES (%s, %s, %s, %s, %s)',  # Lines 74-75
                        (data['username'], data['name'], data['last_name'], data['email'],
data['password']))
            conn.commit()  # Line 76
            return jsonify({'message': 'User added successfully'})  # Line 77
    except psycopg2.Error as e:
        app.logger.error(f"Error adding user: {e}")  # Lines 78-79
        return jsonify({'error': f"Error adding user: {e}"}), 500
    finally:
        conn.close()  # Line 81
```

This endpoint responds to POST requests to add a new user to the "USER" table and returns a success message in JSON format.

Starting the Flask Application

```
if __name__ == '__main__':  # Line 83
    try:
        app.run(port=5005, debug=False)  # Line 84
    except Exception as e:
        print(f"An error occurred: {e}")  # Lines 85-86
```

This section starts the Flask application on port 5005. If an error occurs during startup, it prints an error message.

7) INSTALLATION AND SETUP

Requirements

- Python 3.x
- Dash
- Geopandas
- Plotly
- Folium
- Requests
- Flask
- Psycopg2
- Pandas
- Pyproj
- SQLAlchemy
- Geoalchemy2
- Geopy

Steps to Install

1. **Clone the repository:**

   ```
   git clone <repository_url>
   ```

2. **Navigate to the project directory:**

   ```
   cd <project_directory>
   ```

3. **Create a virtual environment:**

   ```
   python -m venv venv
   ```

4. **Activate the virtual environment**:

   ```
   source venv/bin/activate
   ```

5. **Install the dependencies:**

   ```
   pip install -r requirements.txt
   ```

6. **Set up the database:**

Ensure PostgreSQL is installed and running.

Create the database and necessary tables using `database.py`.

7. **Run the application:**

   Start the Flask server:

   ```
   python Flask_app.py
   ```

8) LICENSE

This project is licensed under the MIT License. See the LICENSE file for details .

This design document provides a comprehensive overview of the IDK application, detailing its purpose, architecture, database design, user interactions, API endpoints, installation instructions, and licensing information. It ensures a clear understanding of the system’s functionality and usage scenarios for both developers and users.