DATABASE MANAGEMENT SYSTEMS

GRADUATE THESIS MANAGEMENT

Furkan Baran

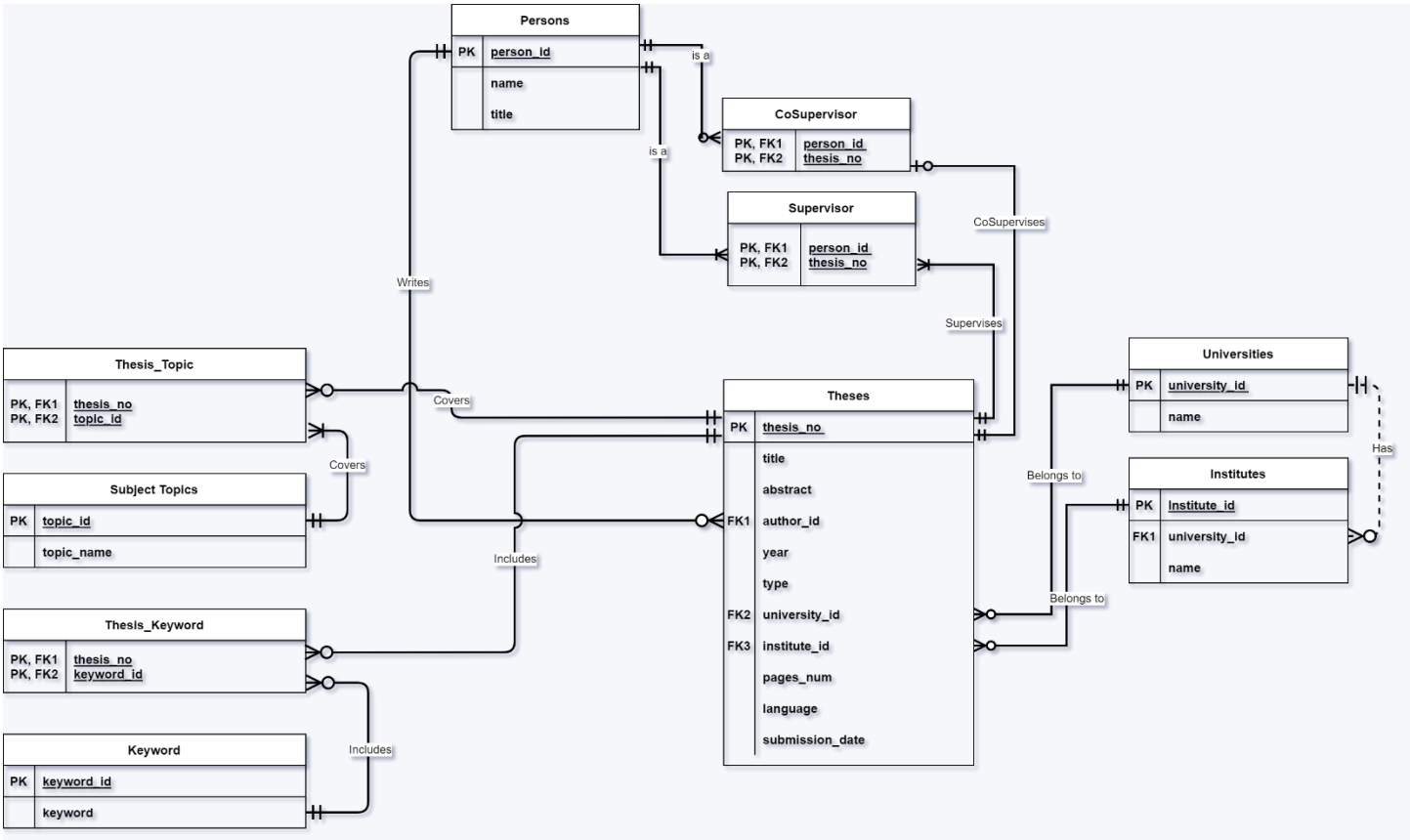
12-2023

Table of Contents

1. Introduction

This Graduate Thesis System (GTS) project involves designing a **PostgreSQL** database to efficiently manage data related to completed graduate theses. The goal is to create a robust and normalized relational database, ensuring data accuracy, efficient querying, and seamless scalability. By leveraging PostgreSQL, the design aims to provide a user-friendly interface for effective interaction with the database, improving the overall management of graduate thesis information.

2. ER



3. Creating Tables-Relations

These SQL commands were utilized to create each table and define the relationships between them. Logical assumptions were made during the table creation process, ensuring the integrity of the overall database design.

3.1. Persons Table

Stores information about individuals involved in the thesis process, with a unique identifier (person_id), title, and name.

```
CREATE TABLE Persons (
    person_id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    NAME VARCHAR(255) NOT NULL
);
```

3.2. Universities Table

Represents Universities, storing a unique identifier (university_id) and the name of each university.

```
CREATE TABLE Universities (
    university_id SERIAL PRIMARY KEY,
    NAME VARCHAR(255) NOT NULL
);
```

3.3. Institutes Table

Contains information about academic institutes, including a unique identifier (institute_id), name, and a reference to the associated university.

```
CREATE TABLE Institutes (
    institute_id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    university_id INTEGER REFERENCES Universities(university_id) CHECK (university_id IS NOT NULL)
);
```

3.4. Theses Table

Central table for storing thesis information, including a unique thesis number (thesis_no), title, abstract, author information (person), year, type, university, institute, page count, language, and submission date.

```
CREATE TABLE Theses (
    thesis_no SERIAL PRIMARY KEY CHECK (thesis_no >=0 AND thesis_no<= 9999999),
    title VARCHAR(500) NOT NULL,
    abstract VARCHAR(5000) NOT NULL,
    author_id INTEGER REFERENCES Persons(person_id),
    year INTEGER NOT NULL,
    type VARCHAR(50) CHECK (TYPE IN ('Master', 'Doctorate', 'Specialization in Medicine', 'Proficiency in
Art')),
    university_id INTEGER REFERENCES Universities(university_id),
    institute_id INTEGER REFERENCES Institutes(institute_id),
    num_pages INTEGER,
    language VARCHAR(50) CHECK (LANGUAGE IN ('Turkish', 'English', 'French')),
    submission_date DATE
);
```

3.4. Supervisors - CoSupervisors Tables

Captures information about thesis supervisors-co supervisors, including the thesis they supervise and the supervisor's ID.

```
CREATE TABLE Supervisors (
    thesis_no INTEGER REFERENCES Theses(thesis_no) ,
    supervisor INTEGER REFERENCES Persons(person_id) ,
    PRIMARY KEY (thesis_no, supervisor)
);

CREATE TABLE CoSupervisors (
    thesis_no INTEGER REFERENCES Theses(thesis_no),
    cosupervisor INTEGER REFERENCES Persons(person_id),
    PRIMARY KEY (thesis_no, cosupervisor)
);
```

3.5. SubjectTopics Table

Stores various subject topics with a unique identifier (topic_id) and a descriptive name.

```
CREATE TABLE SubjectTopics (
    topic_id SERIAL PRIMARY KEY,
    topic_name VARCHAR(255) NOT NULL
);
```

3.6. ThesisSubjectTopics Table

Establishes a relationship between theses and subject topics.

```
CREATE TABLE ThesisSubjectTopics (
    thesis_no INTEGER REFERENCES Theses(thesis_no),
    topic_id INTEGER REFERENCES SubjectTopics(topic_id),
    PRIMARY KEY (thesis_no, topic_id)
);
```

3.7. Keywords Table

Stores different keywords associated with the theses.

```
CREATE TABLE Keywords (
    keyword_id SERIAL PRIMARY KEY,
    keyword VARCHAR(255) NOT NULL
);
```

3.8. ThesisKeywords Table

Represents the relationship between theses and keywords.

```
CREATE TABLE ThesisKeywords (
    thesis_no INTEGER REFERENCES Theses(thesis_no),
    keyword_id INTEGER REFERENCES Keywords(keyword_id),
    PRIMARY KEY (thesis_no, keyword_id)
);
```

4. Logical Restricts and Constraints

4.1. Thesis Publish Date Trigger

Checks if an authors theses's dates are in logical order.

```
-- Trigger Creation
CREATE OR REPLACE FUNCTION check_thesis_publish_date()
RETURNS TRIGGER AS $$
BEGIN
    -- Master's Thesis Publication Date Check
    IF NEW.type = 'Master' THEN
        IF EXISTS (SELECT 1 FROM Theses WHERE author_id = NEW.author_id AND type = 'Doctorate' AND
submission_date <= NEW.submission_date) THEN
            RAISE EXCEPTION 'Master thesis cannot be published after the doctoral thesis submission date.';
        END IF;
    END IF;

    -- Doctoral Thesis Publication Date Check
    IF NEW.type = 'Doctorate' THEN
        IF EXISTS (SELECT 1 FROM Theses WHERE author_id = NEW.author_id AND type = 'Master' AND
submission_date >= NEW.submission_date) THEN
            RAISE EXCEPTION 'Doctoral thesis cannot be published before the master thesis submission date.';
        END IF;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Activate the Trigger
CREATE TRIGGER thesis_publish_date_trigger
BEFORE INSERT OR UPDATE ON Theses
FOR EACH ROW EXECUTE FUNCTION check_thesis_publish_date();
```

4.2. Are Author and Superviso the Same Person Trigger

Checks if author and supervisor/cosupervisor are the same person

```
-- Trigger Creation
CREATE OR REPLACE FUNCTION check_author_and_supervisors()
RETURNS TRIGGER AS $$
BEGIN
    -- Author-supervisor check
    IF NEW.author_id= NEW.supervisor OR NEW.author_id =NEW.cosupervisor THEN
        RAISE EXCEPTION 'Thesis author cannot be the same as the supervisor or co-supervisor.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Activate the Trigger
CREATE TRIGGER author_and_supervisors_trigger
BEFORE INSERT OR UPDATE ON Theses
FOR EACH ROW EXECUTE FUNCTION check_author_and_supervisors();
```

5. Filling Tables

```
-- Persons
INSERT INTO Persons (title, NAME) VALUES
('Dr.', 'Ahmet Yılmaz'),
('Prof.', 'Mehmet Demir'),
('MSc.', 'Mustafa Baran'),
('Ph.D.', 'Emine Kılıç'),
('Dr.', 'Ali Özcan');
-- Universities
INSERT INTO Universities (NAME) VALUES
('University of California'),
('University of Oxford'),
('Massachusetts Institute of Technology'),
('Ege University'),
('Maltepe University');
-- Institutes
INSERT INTO Institutes (NAME, university_id) VALUES
('Science Institute', 1),
('Engineering Institute', 3),
('Business School', 2),
('Medical Institute', 4),
('Arts Institute', 5);
-- Theses
INSERT INTO Theses (title, abstract, author_id, year, TYPE, university_id, institute_id, num_pages, LANGUAGE,
submission_date)
VALUES
('Exploring the Universe', 'This is an abstract about the universe...', 1, 2022, 'Master', 1, 1, 123, 'English', '2022-01-
15'),
('Medical Breakthroughs', 'Research on breakthroughs in medicine...', 4, 2023, 'Doctorate', 4, 4, 150, 'Turkish', '2023-
05-20'),
('Business Innovations', 'Innovative strategies for businesses...', 5, 2021, 'Master', 3, 3, 120, 'French', '2021-08-10'),
('Artistic Expressions', 'Expressing emotions through art...', 2, 2023, 'Proficiency in Art', 5, 5, 809, 'Turkish', '2023-
11-30'),
('Environmental Sustainability', 'Sustainable practices for a better world...', 3, 2022, 'Master', 2, 2, 90, 'English',
'2022-04-25');
```

5.1 Filling Tables (cont'd)

```sql
-- Assumed thesis_no's value will increment from 1
-- Supervisors
INSERT INTO Supervisors (thesis_no, supervisor)
VALUES
(1, 2), -- Dr. Ahmet Yılmaz is the supervisor of the thesis '1'
(2, 1), -- Prof. Mehmet Demir is the supervisor of the thesis '2'
(3, 3), -- MSc. Mustafa Baran is the supervisor of the thesis '3'
(4, 4), -- Ph.D. Emine Kılıç is the supervisor off the thesis '4'
(5, 5); -- Dr. Ali Özcan is the supervisor of the thesis '5'

-- CoSupervisors
INSERT INTO CoSupervisors (thesis_no, cosupervisor)
VALUES
(1, 3), -- MSc. Mustafa Baran is the co-supervisor of the thesis '1'
(3, 2), -- Prof. Mehmet Demir is the co-supervisor of the thesis '3'
(4, 1); -- Dr. Ahmet Yılmaz is the co-supervisor of the thesis 4'

-- SubjectTopics
INSERT INTO SubjectTopics (topic_name) VALUES
('Astronomy and Space Sciences'),
('Emergency Medicine'),
('Business Administration'),
('Fine Arts'),
('Environmental Engineering');

-- ThesisSubjectTopics
INSERT INTO ThesisSubjectTopics (thesis_no, topic_id) VALUES
(1, 1), -- Thesis '1' -> 'Astronomy and Space Sciences'
(2, 2), -- Thesis '2' -> 'Emergency Medicine'
(3, 3), -- Thesis '3' -> 'Business Administration'
(4, 4), -- Thesis '4' -> 'Fine Arts'
(5, 5); -- Thesis '5' -> 'Environmental Engineering'

-- Keywords
INSERT INTO Keywords (keyword) VALUES
('Space Exploration'),
('Medical Research'),
('Innovation'),
('Art'),
('Sustainability');

-- ThesisKeywords
INSERT INTO ThesisKeywords (thesis_no, keyword_id) VALUES
(1, 1), -- Thesis '1' -> 'Space Exploration'
(2, 2), -- Thesis '2' -> 'Medical Research'
(3, 3), -- Thesis '3' -> 'Innovation'
(4, 4), -- Thesis '4' -> 'Art'
(5, 5); -- Thesis '4' -> 'Sustainability'
```

6. Sample Queries - Tests

6.1. Sample SELECT Query

Getting all thesis with their thesis no, title, abstract, author name, supervisor name, year, submission date, topic and keywords

```sql
SELECT
    t.thesis_no,
    t.title,
    t.abstract,
    p.name AS author_name,
    sp.name AS supervisor_name,
    t.year,
    t.submission_date,
    st.topic_name AS subject,
    STRING_AGG(k.keyword, ', ') AS keywords -- concatenation different keywords
FROM
    Theses t
JOIN Persons p ON t.author_id =p.person_id
JOIN Supervisors s ON t.thesis_no= s.thesis_no
JOIN Persons sp ON s.supervisor=sp.person_id
LEFT JOIN ThesisSubjectTopics tst ON t.thesis_no= tst.thesis_no
LEFT JOIN SubjectTopics st ON tst.topic_id=st.topic_id
LEFT JOIN ThesisKeywords tk ON t.thesis_no=tk.thesis_no
LEFT JOIN Keywords k ON tk.keyword_id= k.keyword_id
GROUP BY
    t.thesis_no, t.title, t.abstract, p.name, sp.name, t.year, t.submission_date, st.topic_name;
```

6.2. Sample Output:

Query | Query History

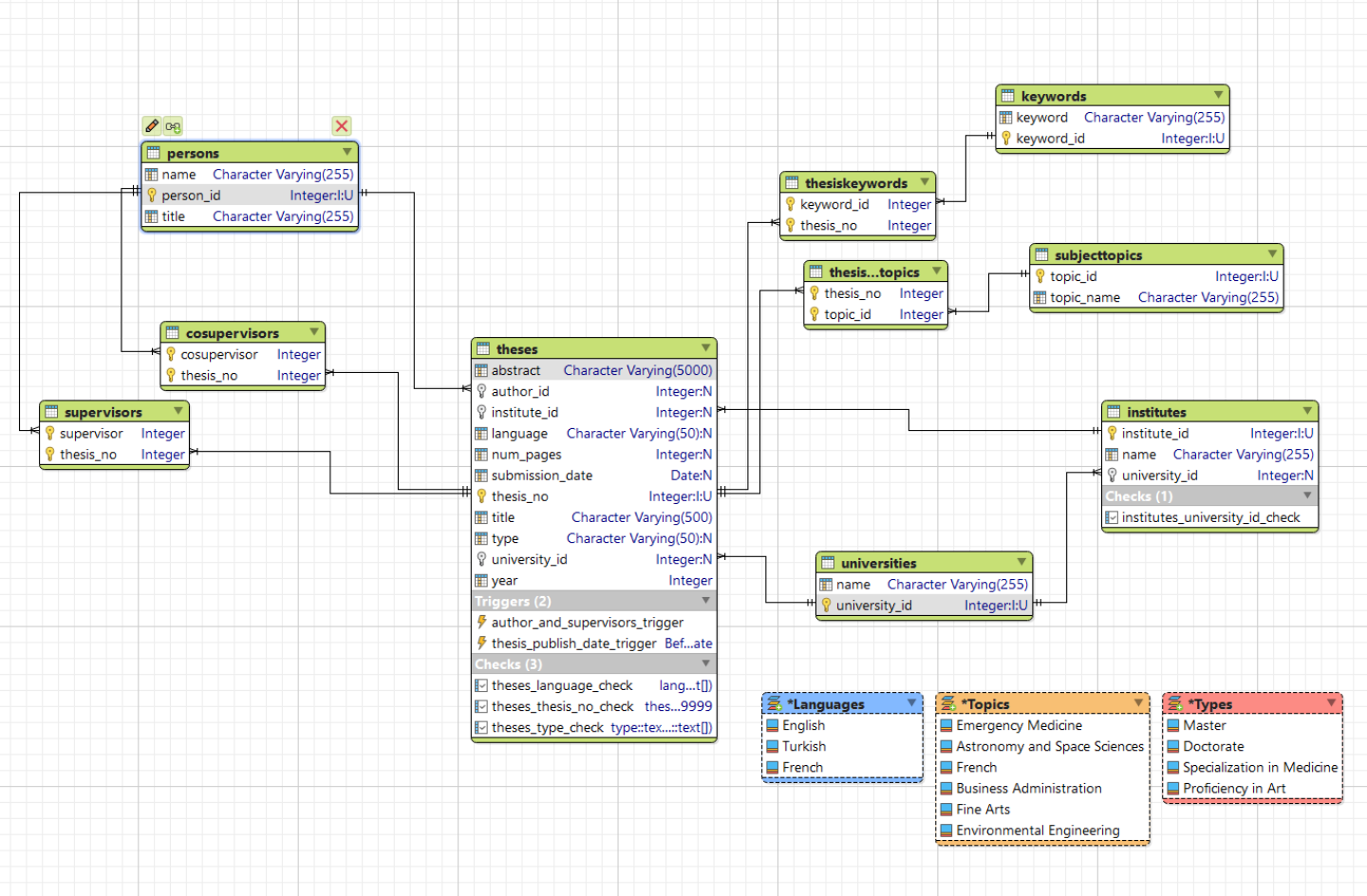
```sql
-- Getting all thesis with their thesis no, title, abstract, author name, supervisor name, year, submission date, topic and keywords
SELECT
    t.thesis_no,
    t.title,
    t.abstract,
    p.name AS author_name,
    sp.name AS supervisor_name,
    t.year,
    t.submission_date,
    st.topic_name AS subject,
    STRING_AGG(k.keyword, ', ') AS keywords -- concatenation different keywords
FROM
    Theses t
JOIN Persons p ON t.author_id=p.person_id
JOIN Supervisors s ON t.thesis_no=s.thesis_no
JOIN Persons sp ON s.supervisor=sp.person_id
LEFT JOIN ThesisSubjectTopics tst ON t.thesis_no=tst.thesis_no
LEFT JOIN SubjectTopics st ON tst.topic_id=st.topic_id
LEFT JOIN ThesisKeywords tk ON t.thesis_no=tk.thesis_no
LEFT JOIN Keywords k ON tk.keyword_id=k.keyword_id
GROUP BY
    t.thesis_no, t.title, t.abstract, p.name, sp.name, t.year, t.submission_date, st.topic_name;

```

Data Output | Messages | Notifications

	thesis_no integer	title character varying (500)	abstract character varying (5000)	author_name character varying (255)	supervisor_name character varying (255)	year integer	submission_date date	subject character varying (255)	keywords text
1	1	Exploring the Universe	This is an abstract about the universe...	Ahmet Yılmaz	Mehmet Demir	2022	2022-01-15	Astronomy and Space Sciences	Space Exploration, Innovation
2	2	Medical Breakthroughs	Research on breakthroughs in medicine...	Emine Kılıç	Ahmet Yılmaz	2023	2023-05-20	Emergency Medicine	Medical Research
3	3	Business Innovations	Innovative strategies for businesses...	Ali Özcan	Mustafa Baran	2021	2021-08-10	Business Administration	Innovation
4	4	Artistic Expressions	Expressing emotions through art...	Mehmet Demir	Emine Kılıç	2023	2023-11-30	Fine Arts	Art
5	5	Environmental Sustainability	Sustainable practices for a better world...	Mustafa Baran	Ali Özcan	2022	2022-04-25	Environmental Engineering	Sustainability

6. Final Relational Database Diagram



7. Design and Implementation of Graduate Thesis System (GTS) Web App

7.1. Overview of the Application:

The application developed as part of the Graduate Thesis System (GTS) is a web-based platform designed to facilitate the management and retrieval of graduate thesis records. Its primary aim is to offer a streamlined interface for users to interact with the underlying PostgreSQL database, handling a wide array of data related to theses, such as author details, university and institute affiliations, and specific thesis attributes like title, abstract, and keywords.

Built using the Python Flask framework, the application is structured to serve as a comprehensive tool for users ranging from academic staff to students. It simplifies tasks such as entering new thesis records, updating existing ones, and querying the database for specific thesis information. The design focuses on user-friendliness and efficiency, ensuring that users can easily navigate through different functionalities and perform necessary operations with minimal complexity.

The application's connection to a PostgreSQL database allows it to handle data robustly and securely, providing reliable performance even as the database scales. This backend setup ensures that all interactions with the database are smooth, consistent, and reflect the most up-to-date information.

With a modular design, the application is structured to be expandable and maintainable, allowing for future enhancements and integration of additional features as the needs of the GTS evolve. Overall, the application stands as a testament to the power of combining a robust database system with a flexible web framework to create a dynamic, efficient, and user-centric platform for managing graduate theses.

7.2. Technologies Used:

In the development of the Graduate Thesis System application, several technologies and frameworks have been employed to ensure robust functionality, ease of use, and system scalability. Below are the key technologies utilized:

Programming Language: Python

Python is a versatile and widely-used programming language known for its readability and efficiency. It serves as the backbone of the application, providing a powerful yet user-friendly environment for developing the web application's logic and functionalities.

Web Framework: Flask

Flask is a lightweight and flexible web framework for Python, known for its ease of use and simplicity. It is utilized in this project to handle web requests, route users to different parts of the application, and render the frontend interface. Flask's extensibility and modular design make it an ideal choice for building scalable web applications.

Database Adapter: psycopg2

Psycopg2 is a PostgreSQL adapter for the Python programming language. It allows Python applications to connect to a PostgreSQL database, making it possible to execute SQL queries, fetch data, and handle database operations directly from Python code. In this application, psycopg2 is used to facilitate the communication between the Flask application and the PostgreSQL database, ensuring that data flows seamlessly from the user interface to the database and back.

7.3. Application Structure:

The Flask application for the Graduate Thesis System is organized to facilitate smooth interaction with the database while providing a user-friendly interface. Below are the primary components of the application's structure:

7.3.1. Routes:

Routes or endpoints are the URLs the application responds to. Each route is associated with a Python function, which is executed when the route is accessed. The Flask application defines several routes to handle different functionalities:

@app.route('/'): The index route, which displays a summary of all thesis records.

@app.route('/add_data'): Allows users to add new entities like universities, institutes, or persons.

@app.route('/add_person', methods=['POST']): Handles the addition of a new person into the database.

@app.route('/search', methods=['GET']): Provides an interface for users to search the thesis database based on various criteria.

... and more routes for adding, editing, deleting, and retrieving detailed information about theses and related entities.

Each route is designed to perform specific operations, such as retrieving data, displaying forms, or updating the database, based on the user's actions.

7.3.2. Templates:

Flask uses the **Jinja2** template engine to render **HTML** pages. Templates are HTML files that contain placeholders for dynamic content that can be injected by the Flask application. In the GTS application, templates serve as the structure for the user interface, displaying data and forms to the user.

search_result.html: Displays the results of a thesis search.

add_data.html: Contains forms for adding new theses, persons, universities, etc.

result.html: Shows success or error messages after data operations.

edit.html, thesis.html, etc.: Other templates for specific functionalities and detailed views.

Templates help maintain a clean separation between the application's logic and the presentation layer, making the code more manageable and the user interface more flexible.

7.3.3. Database Connection:

The application connects to a PostgreSQL database using the psycopg2 library, which is a PostgreSQL adapter for Python. The connection is established using credentials and parameters that specify the host, database name, user, and password.

Connection is typically established at the start of the application and reused for executing queries throughout the application's lifecycle.

Transactions are handled automatically or manually, depending on the operation. For instance, inserting new records or updating existing ones might involve transaction management to ensure that changes are committed or rolled back appropriately.

The application uses cursors obtained from the connection object to execute SQL queries, fetch results, and perform other database operations.

Proper database connection management ensures that the application can execute queries efficiently and handle multiple requests reliably, providing a robust backend for the Flask application.

7.4. App Functionality

The Flask application for the Graduate Thesis System is designed to provide a comprehensive suite of functionalities to manage the thesis database efficiently. Below are the key operations that users can perform:

7.4.1. Adding Data

- **Thesis Records:** Users can add new thesis records via the 'add_data' page, providing necessary details such as title, abstract, author, year, type, and associated university and institute. The application ensures that the data is consistent with the database schema before insertion.

  The **/add_thesis** route in the Flask application is responsible for adding new thesis records to the database. Here's a concise overview of its operation:

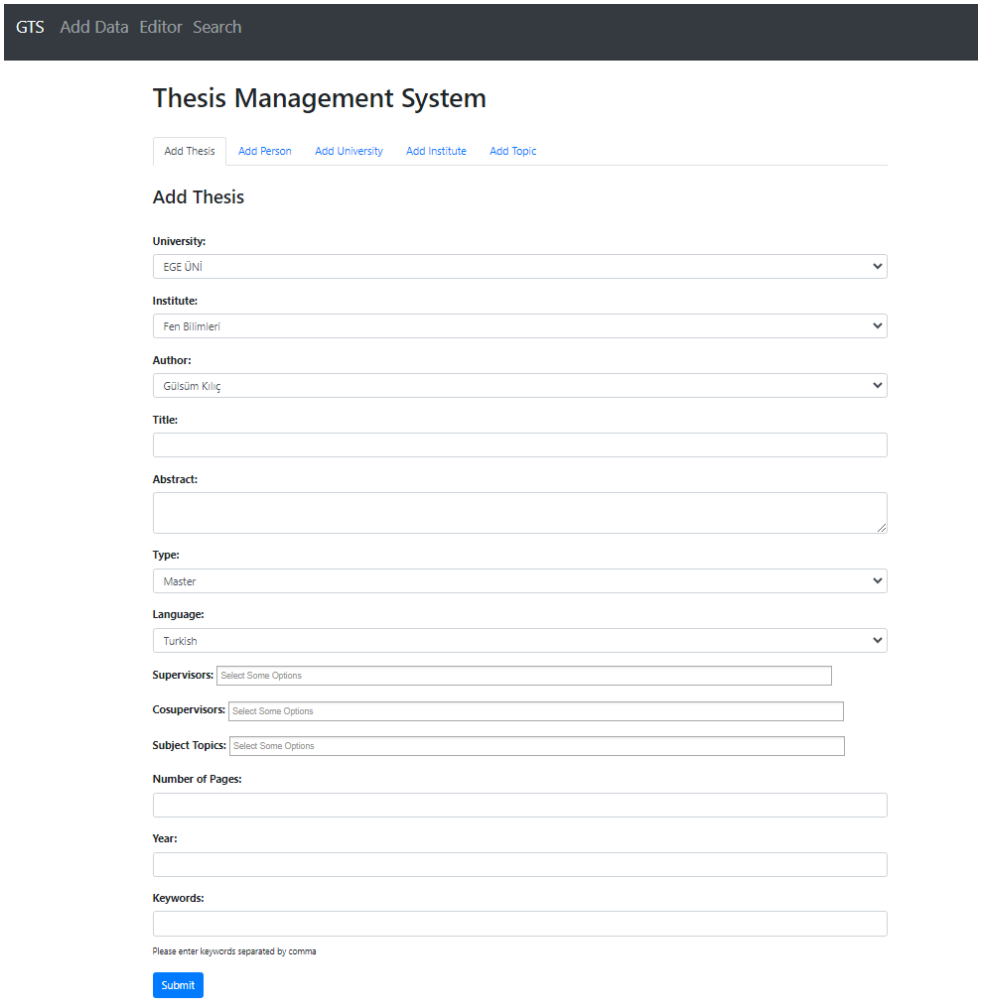
  Data Collection: It gathers all necessary details for a thesis from the user's form submission, including title, abstract, author, year, type, university, institute, number of pages, language, supervisors, co-supervisors, topics, and keywords.

  Validation: The function validates the input by ensuring there is at least one supervisor, at least one topic, all persons exist in the database, and roles are unique.

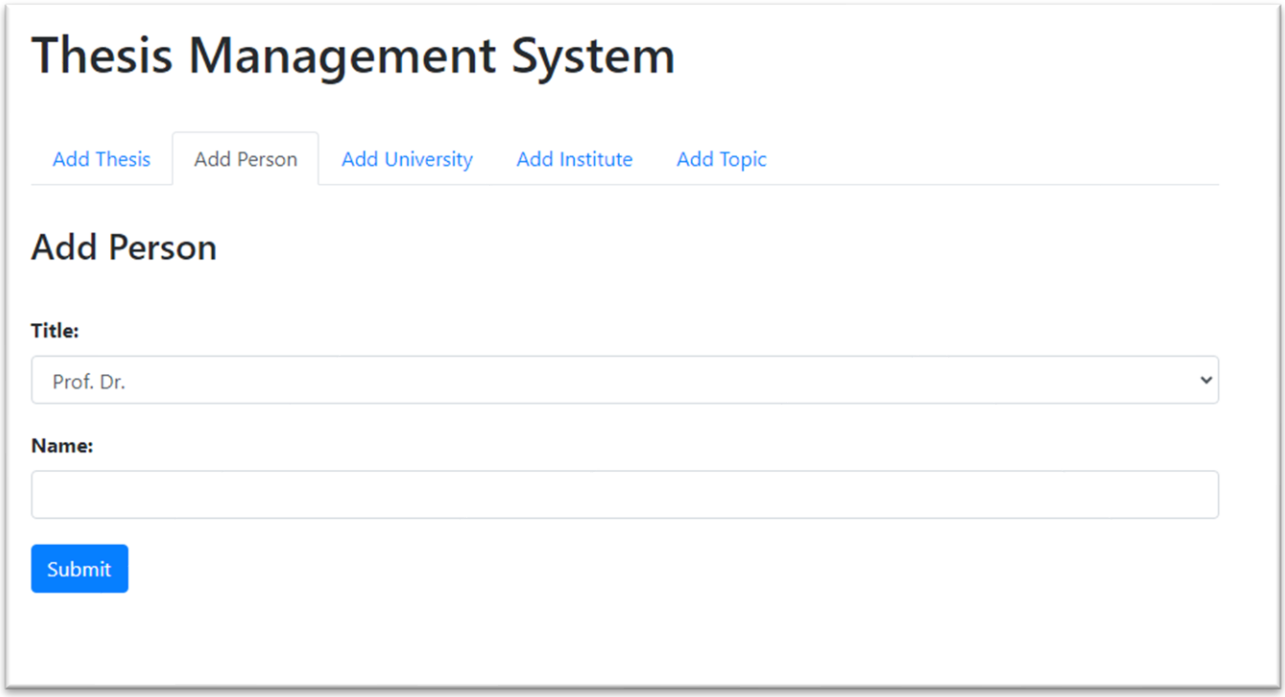
  Insertion: Upon successful validation, it inserts the thesis and related data (supervisors, co-supervisors, topics, and keywords) into their respective tables.

  Transaction Handling: The entire process is wrapped in a database transaction to maintain data integrity, rolling back all changes if an error occurs.

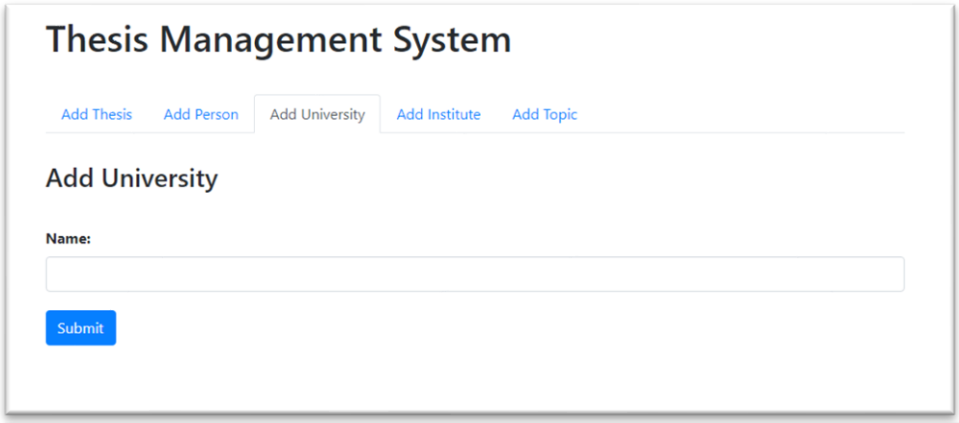
  Feedback: Finally, it returns a success message with the thesis number or an error message detailing any issues encountered.

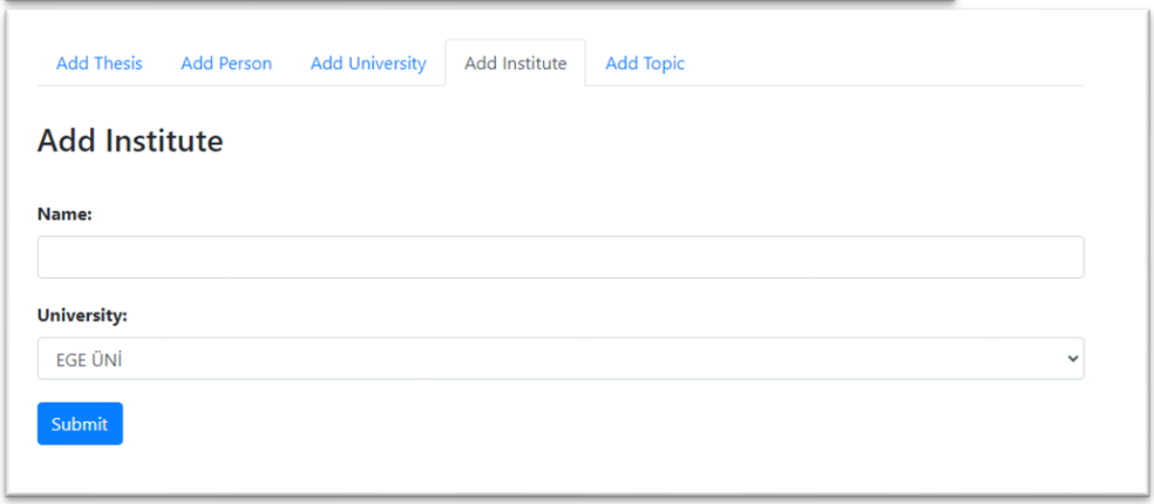


- **Persons**: The 'add_person' route allows users to add new individuals into the system, who can be authors, supervisors, or co-supervisors, by providing their title and name.

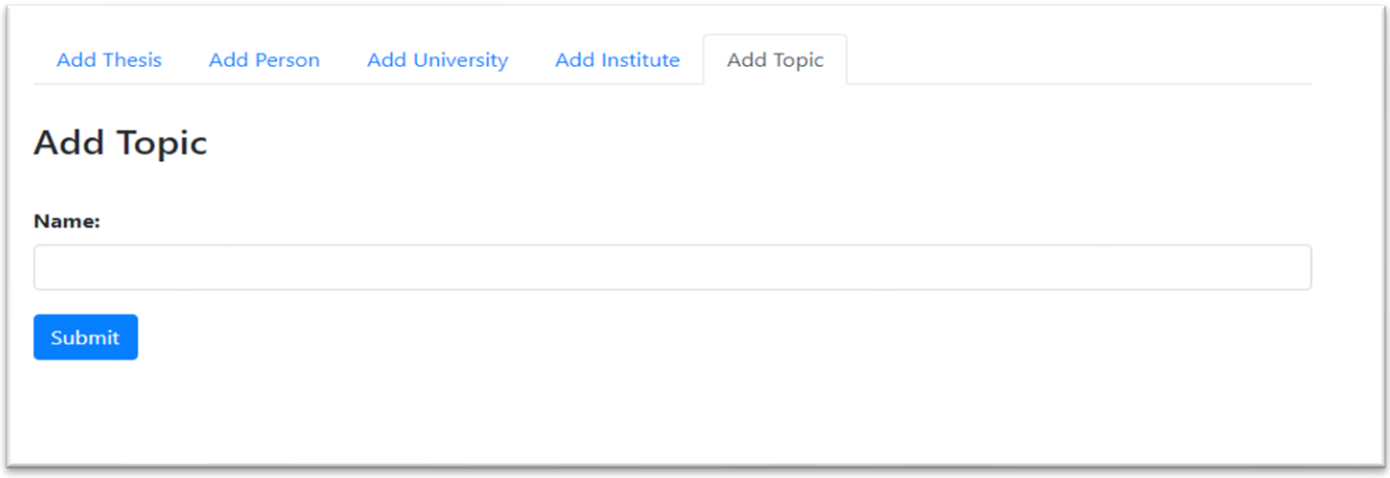


- **Universities & Institutes**: Users can add new universities and their associated institutes, facilitating the expansion of the database to include new academic institutions.



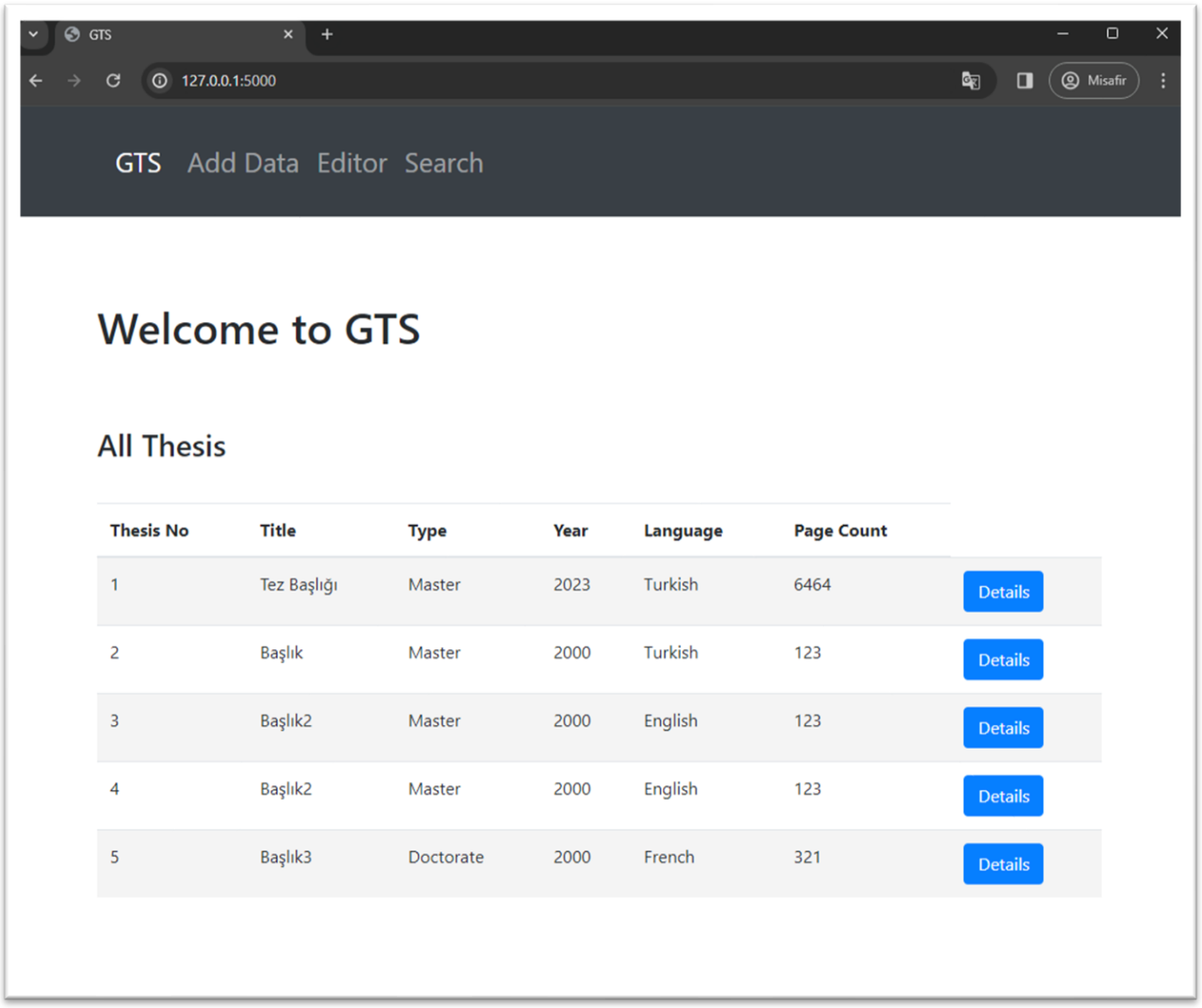


- **Topics**: Similar functionalities exist for adding topics.

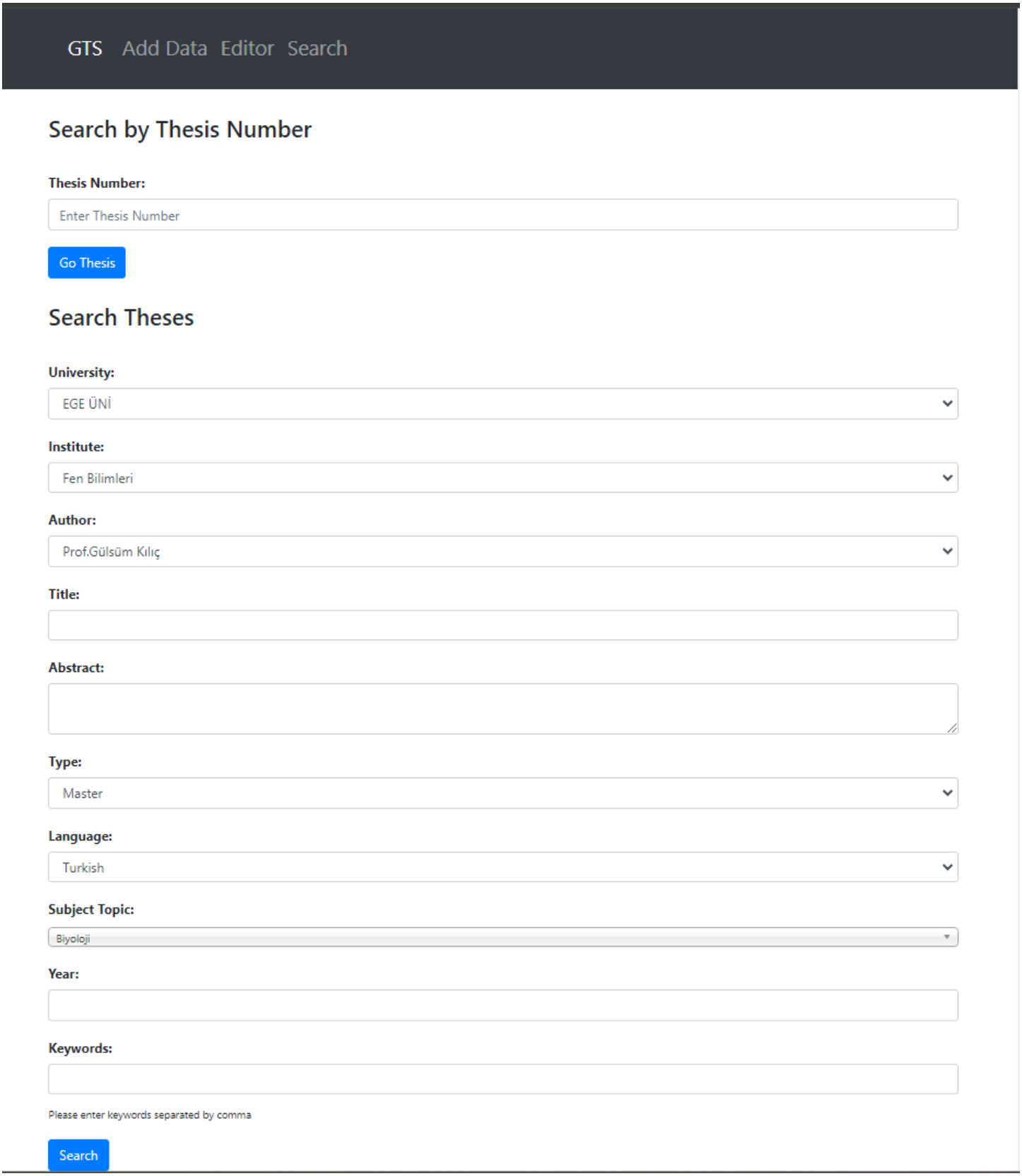


7.4.2. Viewing and Searching Data:

View All Theses: The **index page** presents a summary of all thesis records, allowing users to browse through the entire collection at a glance.

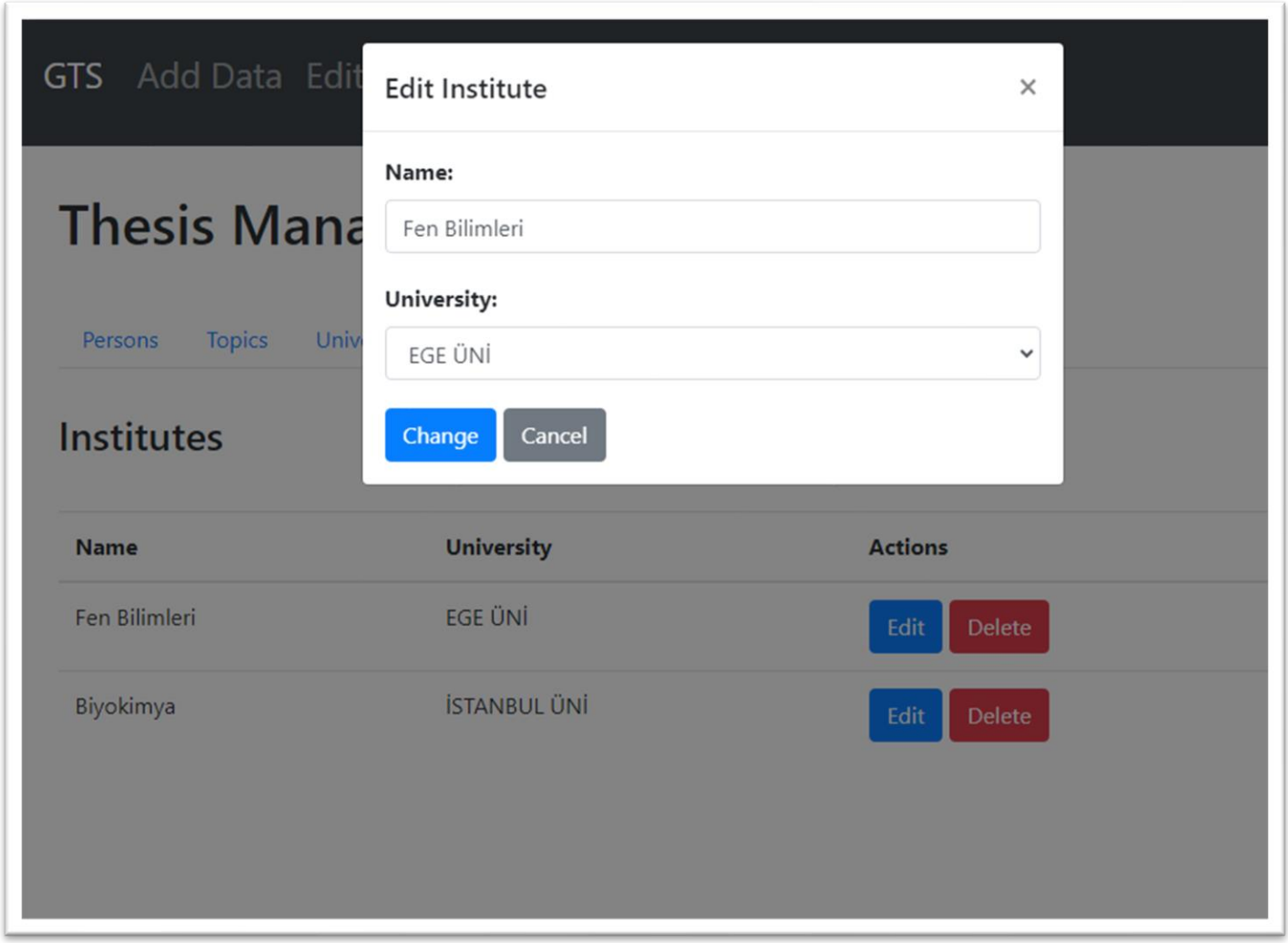


Detailed Search: Users can perform detailed searches based on various criteria such as author, university, keywords, and more. The 'search' functionality provides a form where users can specify their search parameters, and the application retrieves matching records from the database.



7.4.3. Updating Data:

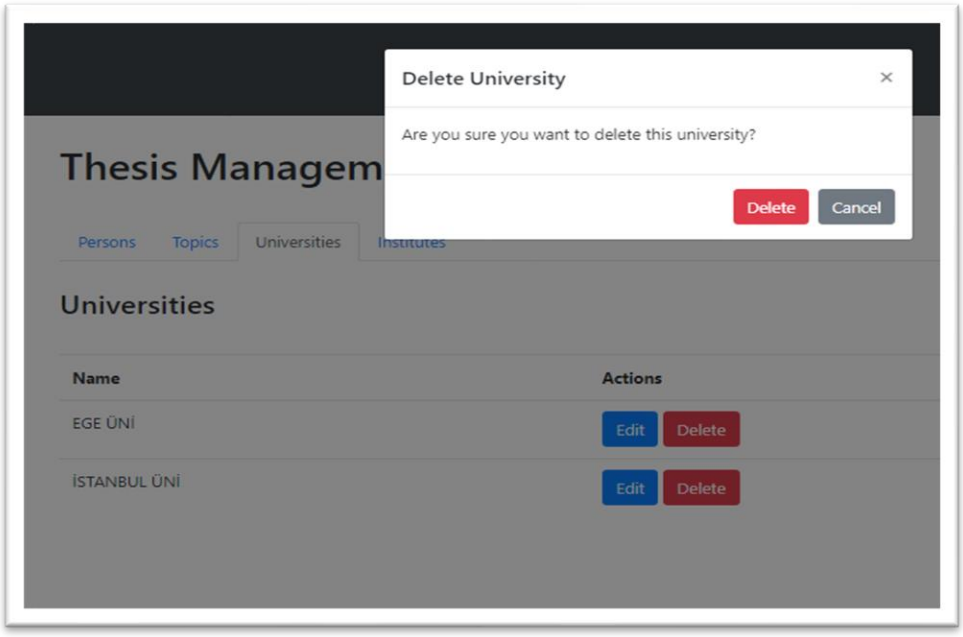
Edit Records: Users can update existing records, such as changing a thesis's details, editing an author's name, or modifying university information. Each entity type has a dedicated interface for updating its details, ensuring data integrity and ease of use.



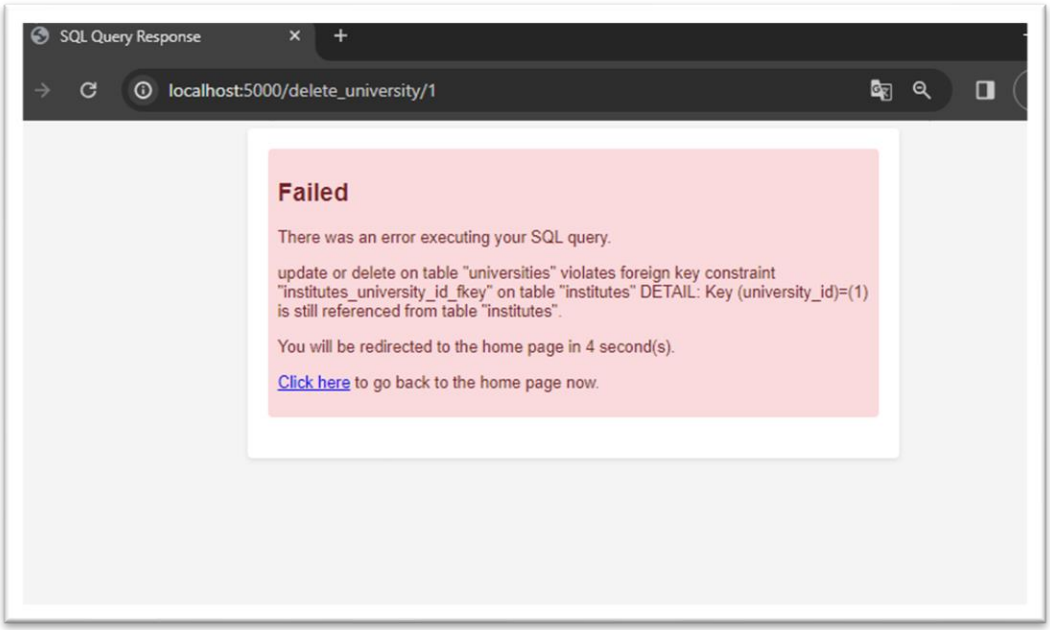
Maintain Relationships: When updating records, the application handles the relationships between different entities, such as the association between theses and authors or universities and institutes, ensuring that all changes are reflected accurately across the database.

7.4.4.Deleting Data

Remove Records: Users have the ability to delete records from the database, such as removing a thesis, a person, or an institute. The application provides confirmation prompts to prevent accidental deletions and handles the removal process, ensuring that related data is also appropriately managed (e.g., removing a thesis also removes its associated supervisor relationships).



Integrity Checks: The application checks for referential integrity, ensuring that deleting a record does not leave orphaned entries or violate the database's consistency. For instance, it prevents the deletion of a university if there are still institutes or theses associated with it.



Together, these functionalities provide a robust interface for managing all aspects of the Graduate Thesis System, from the detailed management of thesis records to the handling of related entities like authors and institutions, ensuring that users can efficiently and effectively interact with the database.

7.5 Interacting with the Database:

The Flask application interacts with the PostgreSQL database through the psycopg2 library, which provides functions and methods to execute PostgreSQL commands from Python. Here's how the application manages database interaction:

Database Connection: At the start of the application, a connection is established to the PostgreSQL database using credentials (host, database, user, password). This connection is used to create a cursor object, which is then used to execute SQL commands and queries.

```
conn = psycopg2.connect(
    host="localhost",
    database="gts",
    user="postgres",
    password="123"
)
# get a cursor from the database connection
conn.autocommit = True
conn.set_client_encoding('UTF8')
cur=conn.cursor()
```

SQL Queries: The application employs SQL queries to interact with the database. These queries include:

INSERT for adding new records to various tables like Theses, Persons, Universities, etc.

```
        # Add data to the Thesis table
        cur.execute(
            """
            INSERT INTO Theses(title, abstract, author_id, year, type, university_id, institute_id,
num_pages, language, submission_date)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s) RETURNING thesis_no;
            """,
            (title, abstract, author, year, type, uni, ins, num_pages, language, submission_date)
        )
```

SELECT for retrieving information, such as fetching all theses or searching based on specific criteria.

```
    # query that gets institute names and its university name
    cur.execute("SELECT institutes.institute_id, institutes.name, institutes.university_id, universities.name
FROM institutes INNER JOIN universities ON institutes.university_id = universities.university_id")
```

UPDATE for modifying existing records, like updating a thesis's details or a person's name.

```
     # query that updates person name and title
       cur.execute("UPDATE Persons SET title = %s, name = %s WHERE person_id = %s", (title, name, id))
```

DELETE for removing records from the database.

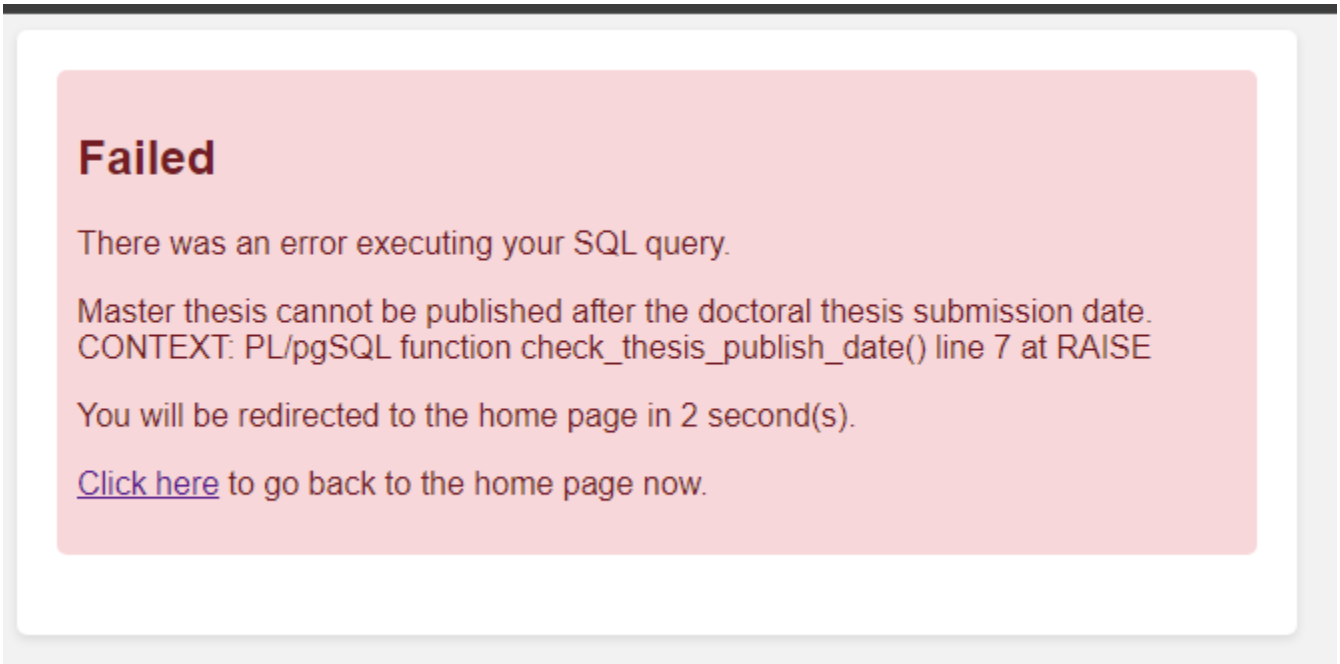
```
        # query that deletes topic for the given topic_id
       cur.execute("DELETE FROM SubjectTopics WHERE topic_id = %s", (id,))
```

Transaction Management: To ensure data integrity, the application uses transactions. When performing operations like adding a thesis, which involves inserting data into multiple tables, the application starts a transaction. If an error occurs at any step, the transaction is rolled back, and no changes are made to the database. If all operations are successful, the transaction is committed, and the changes are saved.

```
            # Commit the operations and close the connection
            conn.commit()
        except psycopg2.errors.RaiseException as e:
            # rollback the add thesis operation
            conn.rollback()
            return render_template('result.html', response=(e), error=True)
```

Managing Relationships: The application maintains the integrity of relationships between different entities. For example, when adding a thesis, it ensures that the specified author exists in the Persons table and that the selected university and institute are valid. It also manages many-to-many relationships, such as a thesis having multiple topics or keywords.

Error Handling: The application is designed to handle database errors gracefully. For instance, if an insert operation violates a constraint (like a duplicate entry or a foreign key constraint), the application catches the exception, rolls back the transaction if necessary, and provides an appropriate error message to the user.



Database Schema Compliance: The application respects the defined database schema, including data types, constraints, and relationships. For example, it ensures that the thesis number is within the allowed range or that the language is one of the predefined options.

7.6 Challenges and Solutions:

During the development of the Graduate Thesis System, several challenges were navigated to create a functional and user-friendly application. Here are some of the key hurdles and the strategies employed to overcome them:

Managing Data Across Multiple Tables:

Inserting related data across multiple tables simultaneously, such as adding a thesis, its author, supervisors, and topics, was complex and prone to errors.

Solution: Implemented transactional control in Python using psycopg2, where multiple insert operations were treated as a single atomic transaction. This ensured that either all changes were committed or none at all, maintaining database integrity.

User Interface Design:

Creating an intuitive and aesthetically pleasing user interface that would make navigation and data entry straightforward for users.

Solution: Utilized HTML, CSS, and JavaScript to build dynamic and responsive web pages. Bootstrap was used for styling and responsiveness, ensuring the application was accessible across various devices and screen sizes.

Logical Checks and Trigger Mistake:

Originally, a trigger was designed to prevent an author from being their own supervisor or co-supervisor. However, it was later realized that the trigger was not correctly accessing the necessary data due to the relational structure.

Solution: Recognized the error in the trigger, where 'NEW.supervisor' didn't exist as supervisors were stored in a different table. The trigger was removed using the DROP TRIGGER command, and logical checks were incorporated at the application layer, ensuring the validation before data insertion.

Optimizing Keyword Insertion:

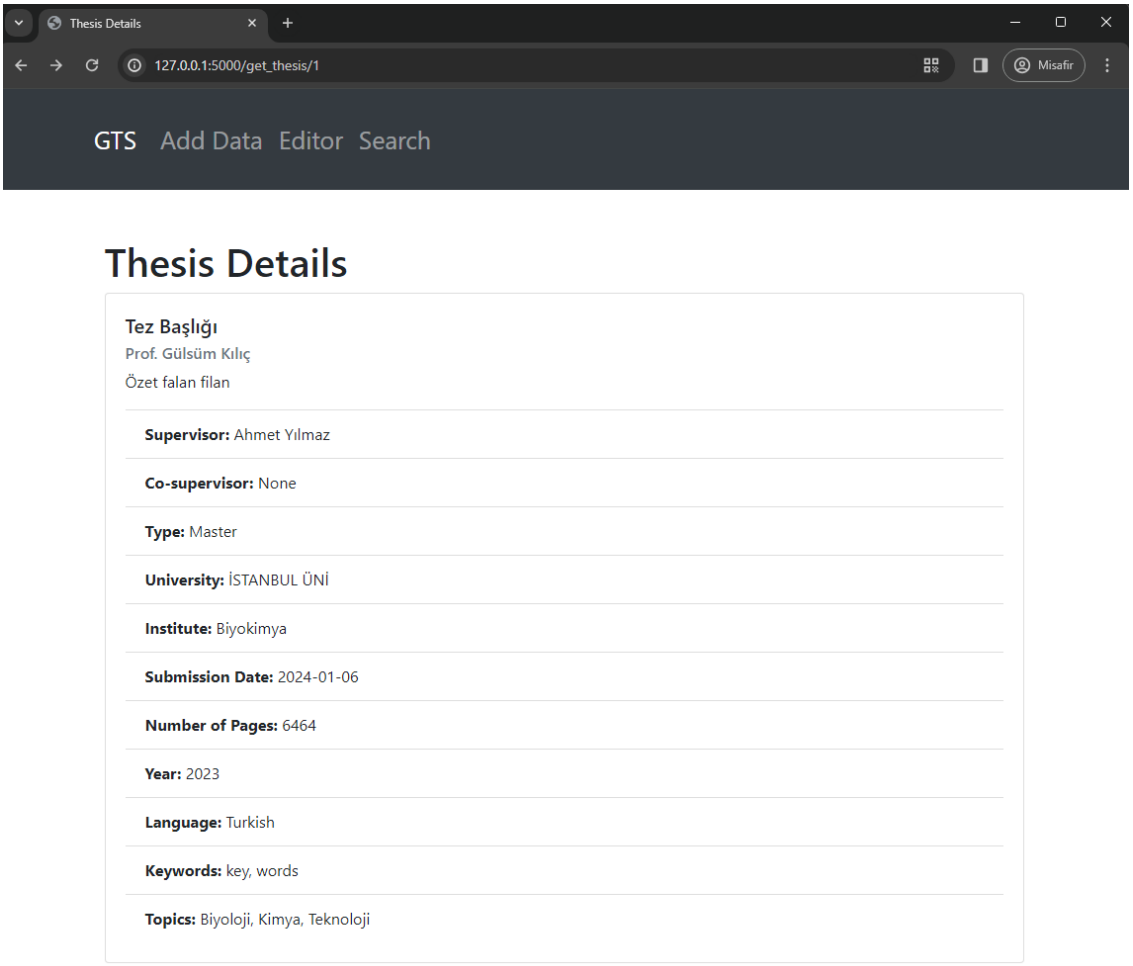
Handling the insertion of keywords in such a way to avoid duplicates and ensure efficiency.

Solution: Implemented the 'ON CONFLICT' clause in PostgreSQL insert queries for keywords. However, to use this feature, it was necessary to ensure that the keyword column was unique. Therefore, an alteration to the Keywords table was made to add a unique constraint with ALTER TABLE Keywords ADD CONSTRAINT unique_keyword UNIQUE(keyword);. This setup enabled the database to handle repeated keyword entries gracefully and avoid duplicate data.

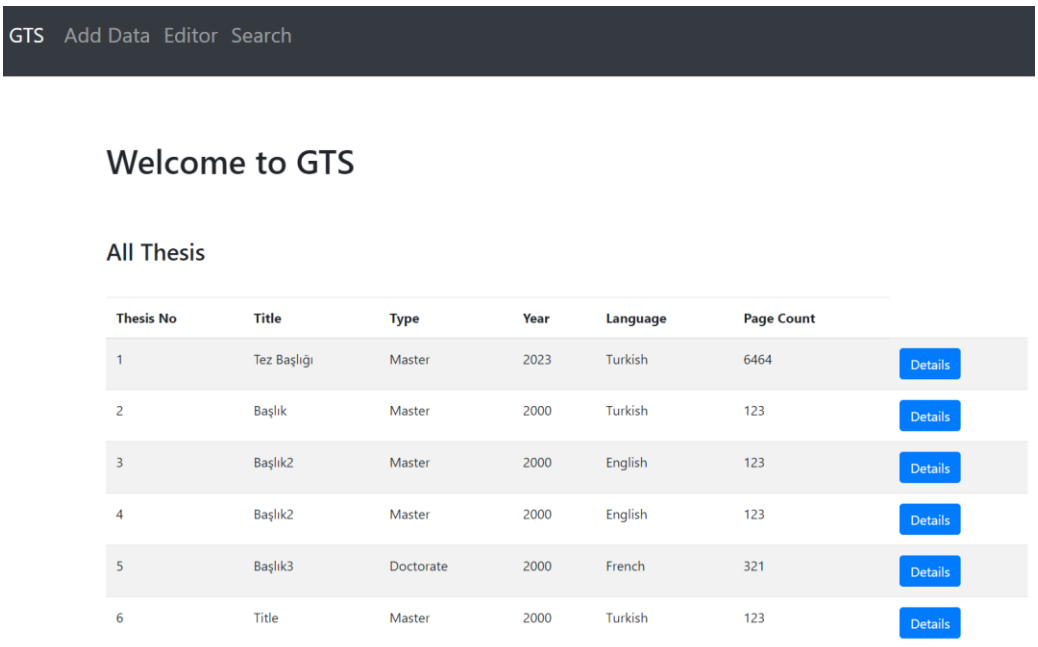
These challenges contributed to a learning curve, pushing for a deeper understanding of not just the technologies but also the importance of planning, testing, and iterating in software development. The experience has significantly contributed to improving problem-solving skills and technical acumen.

7.7 Screenshots From App

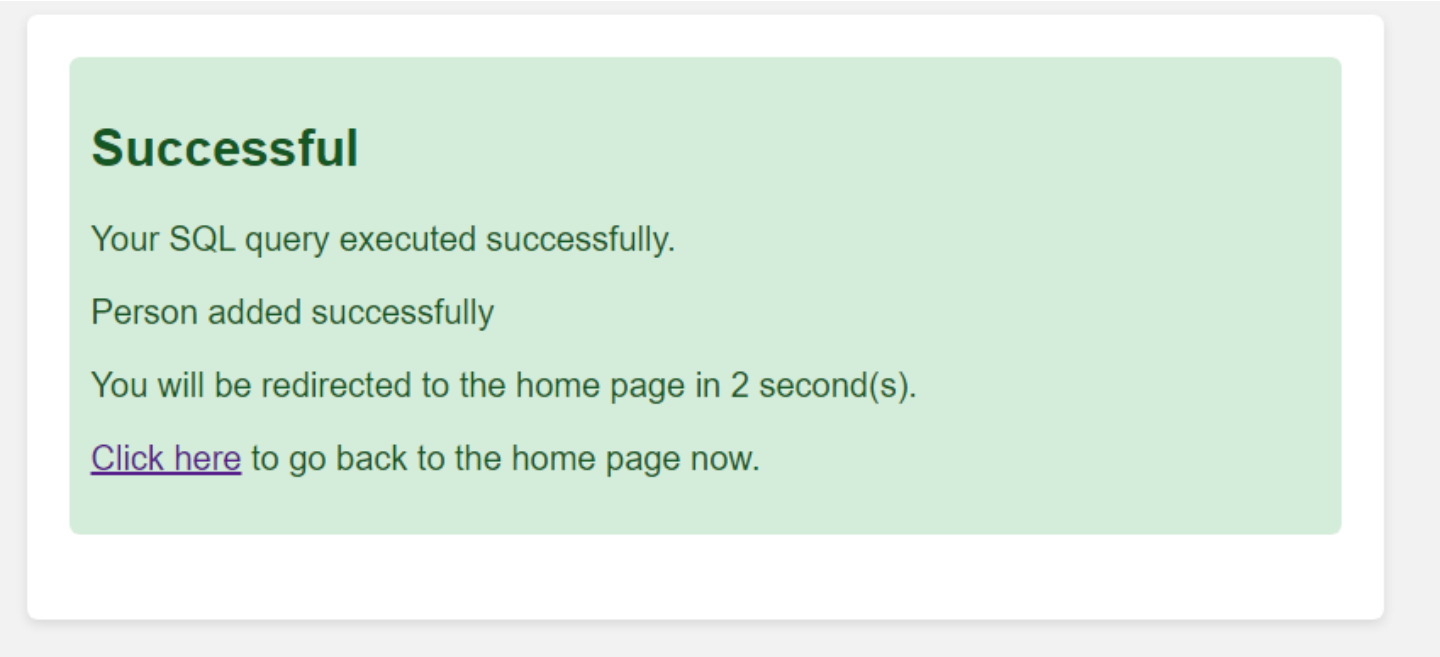
Thesis Details Page



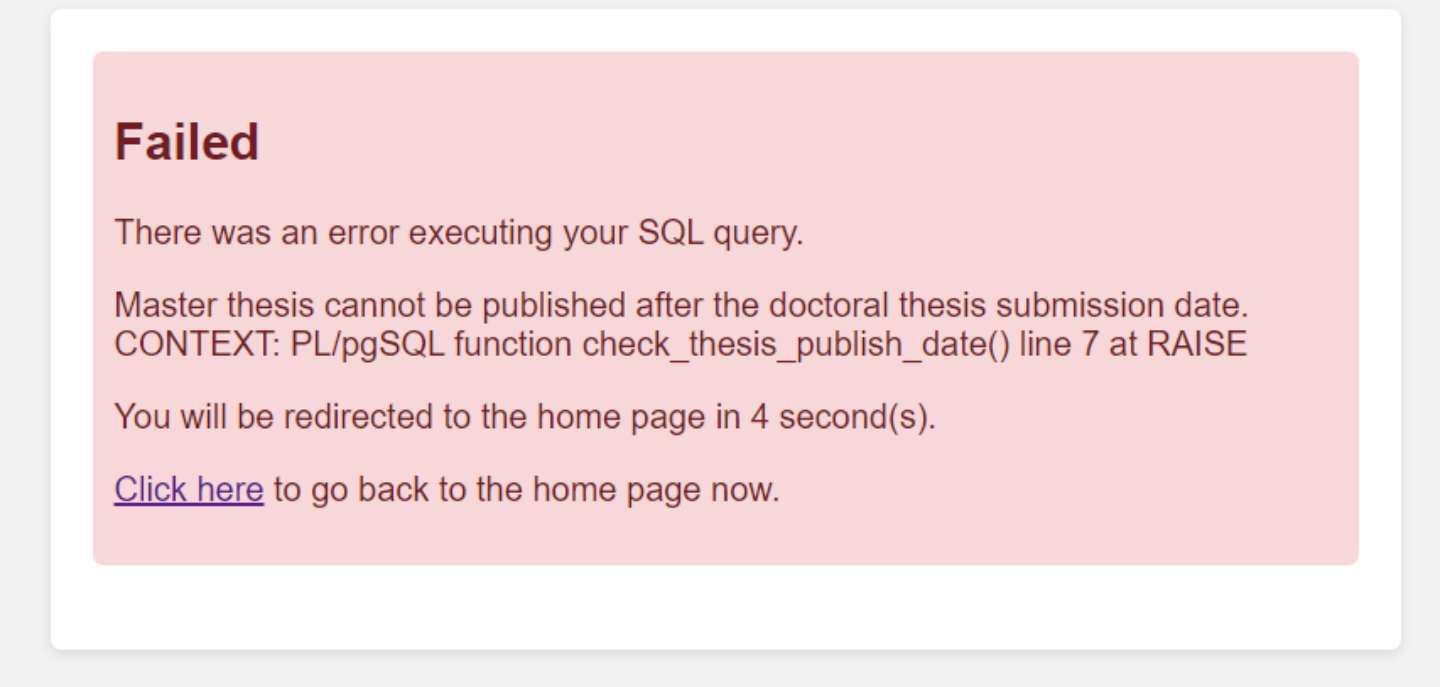
Index Page



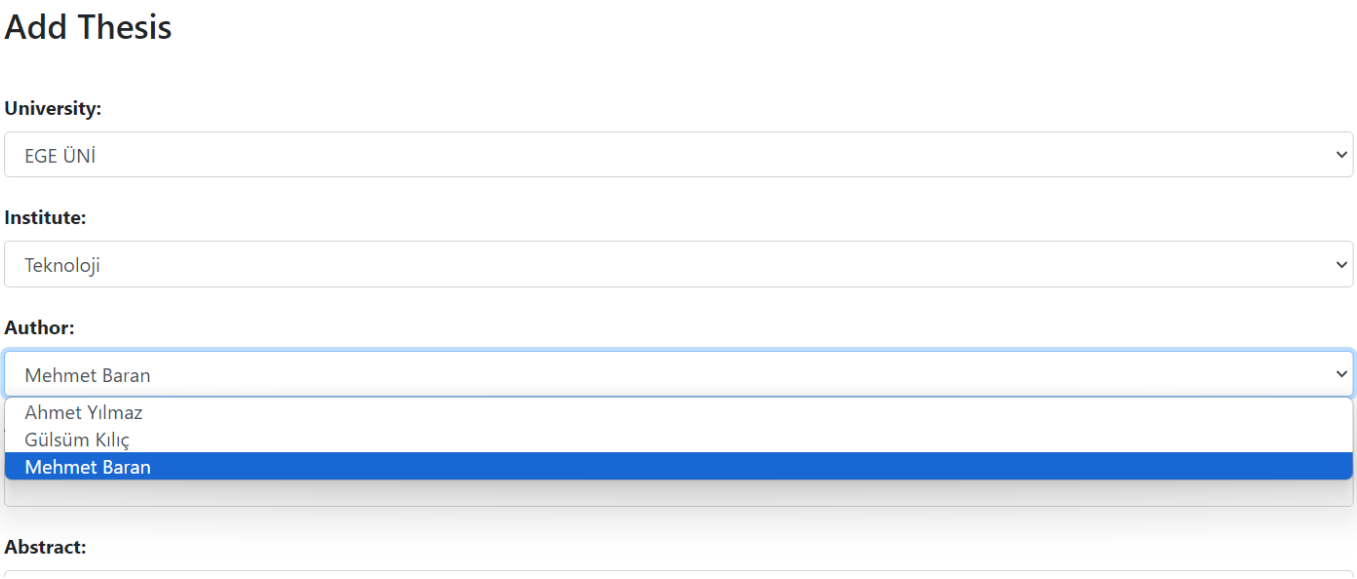
'Person Added Successfully' Feedback



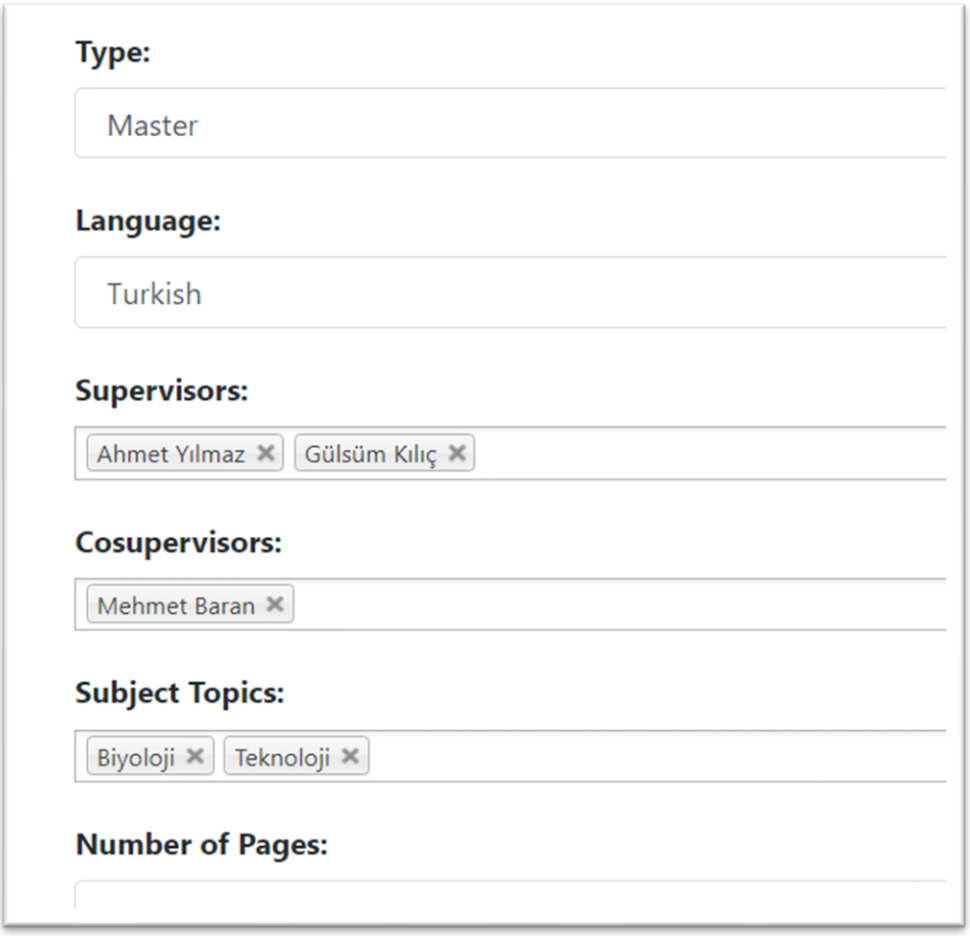
'Master thesis cannot be published after the doctoral thesis submission date' Error Feedback



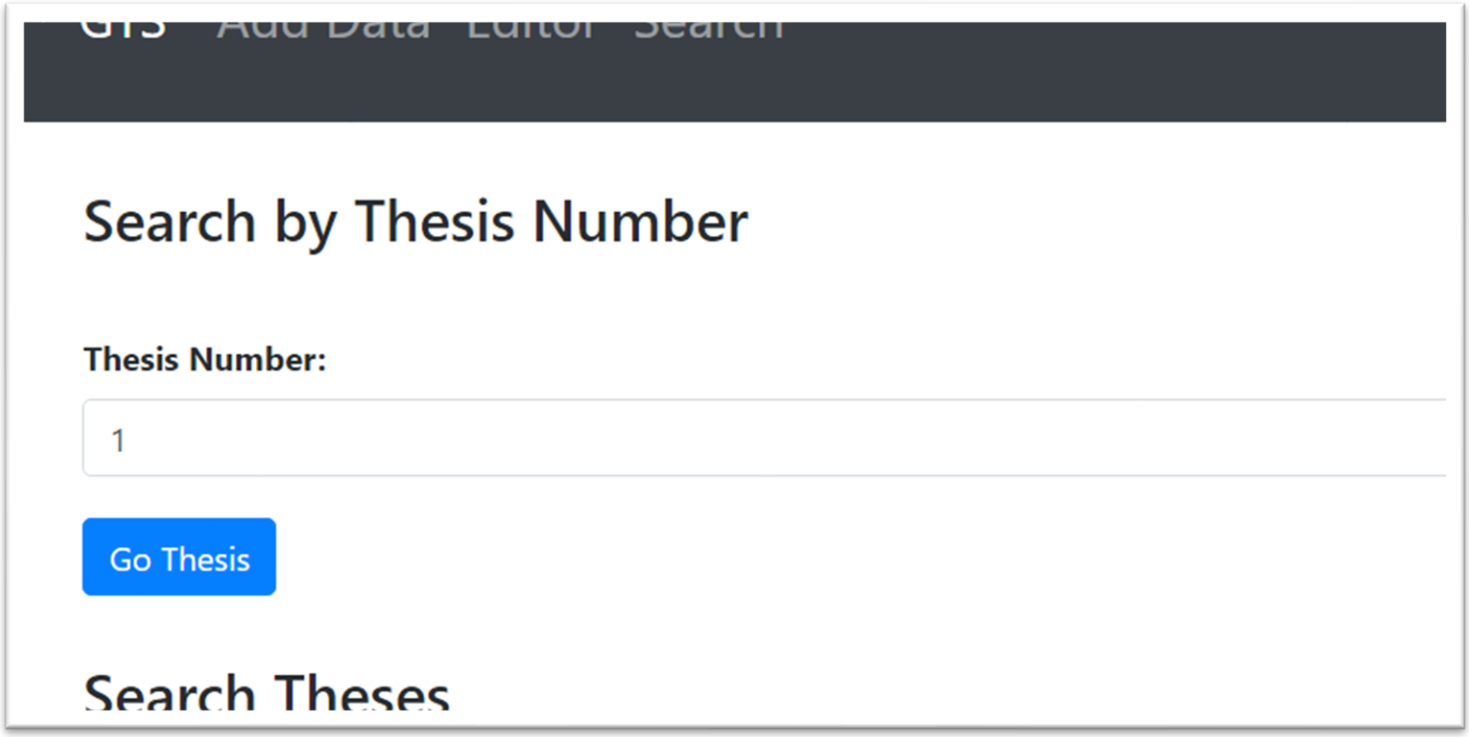
Data Can Be Selected From Options



Multiple Data Can Be Selected From Options



Thesis Can Be Accessible By Its Thesis No



You can access the working version of the application at https://gts-test.vercel.app/