

**T.C. MALTEPE UNIVERSITY**

**FACULTY OF** [**ENGINEERING AND NATURAL SCIENCES**](http://mf.maltepe.edu.tr/en)

**DEPARTMENT OF INDUSTRIAL ENGINEERING**

SE 307 DATABASE MANAGEMENT SYSTEMS TERM PROJECT – 2025/26 Fall Term

Ayşe Su Uslu 230706023

Irem Özbilgin 220706015

Zeynep Lara Korkmaz 230706020

Irmak Çakır 230706024

Yağmur Gülse Özel 220706009

Table of Contents

[1.⁠ ⁠INTRODUCTION 4](#_Toc217409262)

[2. ER DIAGRAM OF OUR DESIGN IN CROW’S FOOT NOTATION 5](#_Toc217409263)

[3. TABLES 6](#_Toc217409264)

[3.1 Person Table 6](#_Toc217409265)

[3.2 University Table 7](#_Toc217409266)

[3.3 Institute Table 7](#_Toc217409267)

[3.4 Thesis Table 8](#_Toc217409268)

[3.5 Supervisor Table 8](#_Toc217409269)

[3.6 Cosupervisor Table 9](#_Toc217409270)

[3.7 Topic Table 9](#_Toc217409271)

[3.8 Keyword Table 9](#_Toc217409272)

[4. Relational Database Diagram 10](#_Toc217409273)

[5. ADDING SAMPLE DATA THAT IS MEANINGFUL TO THE DATABASE 11](#_Toc217409274)

[5.1 Person Sample Data Table 11](#_Toc217409275)

[5.2 University Sample Data Table 11](#_Toc217409276)

[5.3 Institute Sample Data Table 11](#_Toc217409277)

[5.4 Thesis Sample Data Table 12](#_Toc217409278)

[5.5 Supervisor Sample Data Table 12](#_Toc217409279)

[5.6 Cosupervisor Sample Data Table 12](#_Toc217409280)

[5.7 Topic Sample Data Table 13](#_Toc217409281)

[5.8 Keyword Sample Data Table 13](#_Toc217409282)

[6. SQL COMMANDS 14](#_Toc217409283)

[6.1 Commands That Were Used To Create Entities, Indexes and Relationships 14](#_Toc217409284)

[6.1.1 Person Table 14](#_Toc217409285)

[6.1.2 University Table 14](#_Toc217409286)

[6.1.3 Institute Table 14](#_Toc217409287)

[6.1.4 Thesis Table 15](#_Toc217409288)

[6.1.5 Supervisor Table 15](#_Toc217409289)

[6.1.6 Cosupervisor Table 16](#_Toc217409290)

[6.1.7 Topic Table 16](#_Toc217409291)

[6.1.8 Keyword Table 16](#_Toc217409292)

[6.1.9 Indexes 17](#_Toc217409293)

[6.2 Commands Used To Create Sample Data 17](#_Toc217409294)

[6.1.1 Person Sample Data 17](#_Toc217409295)

[6.1.2 University Sample Data 17](#_Toc217409296)

[6.1.3 Institute Sample Data 18](#_Toc217409297)

[6.1.4 Thesis Sample Data 18](#_Toc217409298)

[6.1.5 Supervisor Sample Data 19](#_Toc217409299)

[6.1.6 Cosupervisor Sample Data 19](#_Toc217409300)

[6.1.7 Topic Sample Data 19](#_Toc217409301)

[6.1.8 Keyword Sample Data 20](#_Toc217409302)

[6.1.9 Test Queries to Show that it works 20](#_Toc217409303)

[7. Implementation Phase Of Graduate Thesis System 21](#_Toc217409304)

[7.1 The Overview of The Web Application 21](#_Toc217409305)

[7.2 Application Structure 22](#_Toc217409306)

[7.2.1 Implemented Frontend Routes 22](#_Toc217409307)

[7.2.2 Implemented API Endpoints 23](#_Toc217409308)

[7.2.3 Database Connection 24](#_Toc217409309)

[7.3 Functionality Of The Application 25](#_Toc217409310)

[7.3.1 Dashboard (Home) 25](#_Toc217409311)

[7.3.2 Search 26](#_Toc217409312)

[7.3.3 Add 27](#_Toc217409313)

[7.3.4 Edit 28](#_Toc217409314)

[7.3.4.1 Edit Thesis 29](#_Toc217409315)

[7.3.4.2 Edit Person 30](#_Toc217409316)

[7.3.4.3 Edit University 31](#_Toc217409317)

[7.3.4.4 Edit Institutes 32](#_Toc217409318)

[7.3.5 Details 33](#_Toc217409319)

[7.3.5.1 Thesis Details 34](#_Toc217409320)

[7.3.5.2 Person Details 35](#_Toc217409321)

[7.3.5.3 University Details 36](#_Toc217409322)

[7.3.5.4 Institute Details 37](#_Toc217409323)

[7.3.6 Delete 38](#_Toc217409324)

[7.3.6.1 Delete Thesis 39](#_Toc217409325)

[7.3.6.2 Delete Person 39](#_Toc217409326)

[7.3.6.3 Delete University 39](#_Toc217409327)

[7.3.6.4 Delete Institute 39](#_Toc217409328)

[7.3.7 Error Handling and Feedback 40](#_Toc217409329)

[7.3.8 Technology Summary 41](#_Toc217409330)

[7.4 Interaction with the Database 42](#_Toc217409331)

[7.4.1 Database Connection Lifecycle 42](#_Toc217409332)

[7.4.2 Dashboard Data Retrieval (Home) 43](#_Toc217409333)

[7.4.3 Search Queries and Filtering 45](#_Toc217409334)

[7.4.4 Create Operations (Add / POST) 47](#_Toc217409335)

[7.4.5 Read Operations (Details / GET) 50](#_Toc217409336)

[7.4.6 Update Operations (Edit / PUT) 53](#_Toc217409337)

[7.4.7 Delete Operations (DELETE) 57](#_Toc217409338)

[7.4.8 Referential Integrity and Constraint Handling 59](#_Toc217409339)

[7.4.9 Transaction Management (Commit/Rollback) 59](#_Toc217409340)

[7.4.10 Input Validation and Parameterized Queries 59](#_Toc217409341)

[8. Conclusion 60](#_Toc217409342)

[9. Appendix 61](#_Toc217409343)

1.⁠ ⁠INTRODUCTION

The aim of this project is to design a relational database for the Graduate Thesis System (GTS), which will store, organize, and manage detailed information about graduate theses. The system is required to maintain essential attributes such as thesis number, title, abstract, author, year, thesis type, university, institute, number of pages, language, and submission date. The database must also represent academic relationships accurately, including supervisors, optional co-supervisors, subject topics selected from a predefined list, and user-defined keywords.

This phase of the project focuses solely on database design. The tasks include creating the Entity–Relationship (ER) diagram using Crow’s Foot notation, transforming the conceptual model into relational tables, defining primary and foreign keys, applying constraints according to the provided business rules, and generating SQL commands for creating and populating all tables. Meaningful sample data is inserted into each table to demonstrate the accuracy and usability of the schema.

Our project group uses both macOS and Windows. To ensure a consistent and compatible development environment, we built the database using PostgreSQL and pgAdmin 4. PostgreSQL is explicitly accepted as a relational DBMS in the project requirements, and pgAdmin 4 provides a reliable, platform-independent management tool that allows all team members to work with the same schema and SQL commands.

This design document serves as the midterm submission. The resulting schema forms the foundation for the next phase of the project, where we will develop a functional application that interacts with the database to support thesis submission, modification, and retrieval.

# 2. ER DIAGRAM OF OUR DESIGN IN CROW’S FOOT NOTATION

A diagram of a company

AI-generated content may be incorrect.

# 3. TABLES

## 3.1 Person Table

A screenshot of a computer

AI-generated content may be incorrect.

## 3.2 University Table

A screenshot of a computer

AI-generated content may be incorrect.

## 3.3 Institute Table

A screenshot of a computer

AI-generated content may be incorrect.

## 3.4 Thesis Table

A screenshot of a computer

AI-generated content may be incorrect.

## 3.5 Supervisor Table

A screenshot of a computer

AI-generated content may be incorrect.

## 3.6 Cosupervisor Table

A screenshot of a computer

AI-generated content may be incorrect.

## 3.7 Topic Table

A screenshot of a computer

AI-generated content may be incorrect.

## 3.8 Keyword Table

A screenshot of a computer

AI-generated content may be incorrect.

# 4. Relational Database Diagram

A screenshot of a computer

AI-generated content may be incorrect.

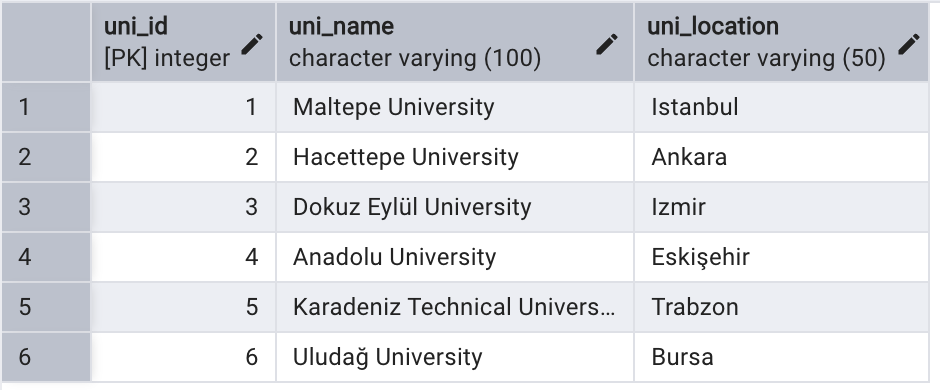
# 5. ADDING SAMPLE DATA THAT IS MEANINGFUL TO THE DATABASE

## 5.1 Person Sample Data Table

A screenshot of a computer

AI-generated content may be incorrect.

## 5.2 University Sample Data Table



## 5.3 Institute Sample Data Table

A screenshot of a computer

AI-generated content may be incorrect.

## 5.4 Thesis Sample Data Table

A screenshot of a computer

AI-generated content may be incorrect.

## 5.5 Supervisor Sample Data Table

A screenshot of a computer

AI-generated content may be incorrect.

## 5.6 Cosupervisor Sample Data Table

A screenshot of a computer

AI-generated content may be incorrect.

## 5.7 Topic Sample Data Table

A screenshot of a computer

AI-generated content may be incorrect.

## 5.8 Keyword Sample Data Table

A screenshot of a computer

AI-generated content may be incorrect.

# 6. SQL COMMANDS

6.1 Commands That Were Used To Create Entities, Indexes and Relationships

### 6.1.1 Person Table

CREATE TABLE person (

    per\_id      SERIAL PRIMARY KEY,

    first\_name  VARCHAR(25) NOT NULL,

    second\_name VARCHAR(25) NOT NULL,

    phone\_num   CHAR(11)

);

6.1.2 University Table

CREATE TABLE university (

    uni\_id       INTEGER PRIMARY KEY,

    uni\_name     VARCHAR(100) NOT NULL,

    uni\_location VARCHAR(50) NOT NULL

);

6.1.3 Institute Table

CREATE TABLE institute (

    ins\_id   INTEGER PRIMARY KEY,

    ins\_name VARCHAR(75) NOT NULL,

    uni\_id   INTEGER NOT NULL,

    FOREIGN KEY (uni\_id) REFERENCES university(uni\_id)

);

6.1.4 Thesis Table

CREATE TABLE thesis (

    th\_num          INTEGER PRIMARY KEY,

    title           VARCHAR(500) NOT NULL,

    abstract        VARCHAR(5000) NOT NULL,

    author\_id       INTEGER NOT NULL,

    th\_year         DATE NOT NULL,

    th\_type         VARCHAR(30) NOT NULL,

    uni\_id          INTEGER NOT NULL,

    ins\_id          INTEGER NOT NULL,

    page\_num        INTEGER NOT NULL,

    th\_language     VARCHAR(20) NOT NULL,

    submission\_date DATE NOT NULL,

    FOREIGN KEY (author\_id) REFERENCES person(per\_id),

    FOREIGN KEY (uni\_id)    REFERENCES university(uni\_id),

    FOREIGN KEY (ins\_id)    REFERENCES institute(ins\_id)

);

6.1.5 Supervisor Table

CREATE TABLE supervisor (

    per\_id INTEGER PRIMARY KEY,

    th\_num INTEGER NOT NULL,

    FOREIGN KEY (per\_id) REFERENCES person(per\_id),

    FOREIGN KEY (th\_num) REFERENCES thesis(th\_num)

);

6.1.6 Cosupervisor Table

CREATE TABLE cousupervisor (

    per\_id INTEGER NOT NULL,

    th\_num INTEGER NOT NULL,

    PRIMARY KEY (per\_id, th\_num),

    FOREIGN KEY (per\_id) REFERENCES person(per\_id),

    FOREIGN KEY (th\_num) REFERENCES thesis(th\_num)

);

6.1.7 Topic Table

CREATE TABLE topic (

    topic\_id   INTEGER PRIMARY KEY,

    th\_num     INTEGER NOT NULL,

    topic\_name VARCHAR(100) NOT NULL,

    FOREIGN KEY (th\_num) REFERENCES thesis(th\_num)

);

6.1.8 Keyword Table

CREATE TABLE keyword (

    keyword\_id INTEGER PRIMARY KEY,

    th\_num     INTEGER NOT NULL,

    keyword    VARCHAR(500) NOT NULL,

    FOREIGN KEY (th\_num) REFERENCES thesis(th\_num)

);

6.1.9 Indexes

CREATE INDEX idx\_thesis\_author   ON thesis(author\_id);

CREATE INDEX idx\_thesis\_uni      ON thesis(uni\_id);

CREATE INDEX idx\_thesis\_ins      ON thesis(ins\_id);

CREATE INDEX idx\_topic\_thesis    ON topic(th\_num);

CREATE INDEX idx\_keyword\_thesis  ON keyword(th\_num);

CREATE INDEX idx\_supervisor\_th   ON supervisor(th\_num);

CREATE INDEX idx\_cosup\_th        ON cousupervisor(th\_num);

## 6.2 Commands Used To Create Sample Data

6.1.1 Person Sample Data

INSERT INTO person (first\_name, second\_name, phone\_num) VALUES

('Aylin', 'Koral', '05437651289'),

('Burak', 'Temel', '05382994456'),

('Selin', 'Uslu', '05419873265'),

('Eren', 'Dumlu', '05468392177'),

('Naz',  'Ersoy', '05348229944'),

('Koray','Aktaş', '05458900312');

6.1.2 University Sample Data

INSERT INTO university (uni\_id, uni\_name, uni\_location) VALUES

(1, 'Maltepe University', 'Istanbul'),

(2, 'Hacettepe University', 'Ankara'),

(3, 'Dokuz Eylül University', 'Izmir'),

(4, 'Anadolu University', 'Eskişehir'),

(5, 'Karadeniz Technical University', 'Trabzon'),

(6, 'Uludağ University', 'Bursa');

6.1.3 Institute Sample Data

INSERT INTO institute (ins\_id, ins\_name, uni\_id) VALUES

(1, 'Graduate School of Engineering', 1),

(2, 'Institute of Social Sciences', 2),

(3, 'Institute of Health Sciences', 3),

(4, 'Institute of Fine Arts', 4),

(5, 'Institute of Marine Sciences', 5),

(6, 'Graduate School of Data Science', 1);

6.1.4 Thesis Sample Data

INSERT INTO thesis

(th\_num, title, abstract, author\_id, th\_year, th\_type, uni\_id, ins\_id, page\_num, th\_language, submission\_date)

VALUES

(1, 'AI-Based Debate Evaluation', 'Speech scoring model design', 1, '2023-01-01', 'Master', 1, 1, 82, 'English', '2023-06-12'),

(2, 'Renewable Energy Storage', 'Battery efficiency study', 2, '2022-01-01', 'PhD', 2, 2, 155, 'Turkish', '2022-11-30'),

(3, 'Marine Ecosystems & Climate', 'Plankton decline analysis', 3, '2021-01-01', 'Master', 5, 5, 93, 'English', '2021-05-18'),

(4, 'Data-Driven Health Diagnosis', 'Predictive model design', 4, '2024-01-01', 'Master', 3, 3, 120, 'English', '2024-02-01'),

(5, 'Art Therapy & Creativity', 'Visual cognition study', 5, '2020-01-01', 'PhD', 4, 4, 210, 'Turkish', '2020-09-22'),

(6, 'Machine Learning for Urban Mobility', 'Traffic flow prediction', 6, '2023-01-01', 'Master', 1, 6, 140, 'English', '2023-07-10');

6.1.5 Supervisor Sample Data

INSERT INTO supervisor (per\_id, th\_num) VALUES

(1, 2),

(2, 1),

(3, 4),

(4, 3),

(5, 6),

(6, 5);

6.1.6 Cosupervisor Sample Data

INSERT INTO cousupervisor (per\_id, th\_num) VALUES

(1, 3),

(2, 5),

(3, 1),

(4, 6),

(5, 2),

(6, 4);

6.1.7 Topic Sample Data

INSERT INTO topic (topic\_id, th\_num, topic\_name) VALUES

(1, 1, 'Artificial Intelligence'),

(2, 2, 'Energy Systems'),

(3, 3, 'Climate Change'),

(4, 4, 'Medical Informatics'),

(5, 5, 'Art & Psychology'),

(6, 6, 'Smart Mobility');

6.1.8 Keyword Sample Data

INSERT INTO keyword (keyword\_id, th\_num, keyword) VALUES

(1, 1, 'machine learning, scoring'),

(2, 2, 'battery, storage, efficiency'),

(3, 3, 'ocean, plankton, climate'),

(4, 4, 'diagnosis, prediction, health'),

(5, 5, 'creativity, therapy, cognition'),

(6, 6, 'traffic, mobility, forecasting');

6.1.9 Test Queries to Show that it works

A screenshot of a computer

AI-generated content may be incorrect.

7. Implementation Phase Of Graduate Thesis System

7.1 The Overview of The Web Application

The Graduate Thesis System web application is an online platform designed to manage and access graduate thesis data stored in our relational database. The system focuses on organizing core thesis metadata (thesis number, title, abstract, author, year, type, university, institute, number of pages, language, submission date) and representing the required academic relationships including supervisors, optional co-supervisors, thesis topics, and user-defined keywords.

The application provides a user-friendly interface to perform common database operations such as viewing records, searching by different fields and adding or updating thesis-related information. The web layer communicates with the PostgreSQL database managed and tested via pgAdmin4 to ensure consistent and structured data storage and reliable querying. The overall goal is to streamline thesis information management and improve accessibility for users by presenting database results through a clear navigable website structure.

7.2 Application Structure

The Graduate Thesis System is implemented as a web-based application with a Flask REST API backend and a React single-page application (SPA) frontend. The frontend uses route-based navigation with React Router and communicates with the backend over JSON endpoints.

7.2.1 Implemented Frontend Routes

Route: /

Purpose: Dashboard that lists theses, persons, universities, and institutes.

Route: /search

Purpose: Search and filter across entities. Thesis searches (Thesis/Topic/Keyword) call the API, while Person/University/Institute filters use cached lists from the API.

Route: /add

Purpose: Create new Person, University, Institute, and Thesis records.

Route: /details/<entity>/<id>

Purpose: Detail pages for thesis, person, university, and institute records.

Route: /edit/<entity>/<id>

Purpose: Edit pages for thesis, person, university, and institute records.

7.2.2 Implemented API Endpoints

The backend exposes REST endpoints under /api.

GET /api/theses, GET /api/theses/<id>, POST /api/theses, PUT /api/theses/<id>, DELETE /api/theses/<id>

GET /api/persons, GET /api/persons/<id>, POST /api/persons, PUT /api/persons/<id>, DELETE /api/persons/<id>

GET /api/universities, GET /api/universities/<id>, POST /api/universities, PUT /api/universities/<id>, DELETE /api/universities/<id>

GET /api/institutes, GET /api/institutes/<id>, POST /api/institutes, PUT /api/institutes/<id>, DELETE /api/institutes/<id>

POST /api/search (searches theses by title/abstract, topic, or keyword based on the request type)

7.2.3 Database Connection

The Graduate Thesis System is implemented to interact with a PostgreSQL relational database that stores all thesis-related data defined in the database schema. The web application communicates with the database through a backend layer developed using the Flask framework.

Database connectivity is handled using the psycopg2 library, which enables the application to establish and manage connections to the PostgreSQL database. A centralized helper function is used to create the database connection, allowing different routes of the application to perform database operations such as querying records, inserting new data, updating existing entries, and deleting records.

The database connection is opened when a request requires interaction with the database and is closed after the operation is completed to ensure efficient use of system resources and data consistency across the application.

An example of the database connection function used in the application is shown below:

import psycopg2

from flask import Flask

app = Flask(\_\_name\_\_)

def get\_db\_connection():

conn = psycopg2.connect(

host="localhost",

database="gtsdb",

user="postgres",

password=""

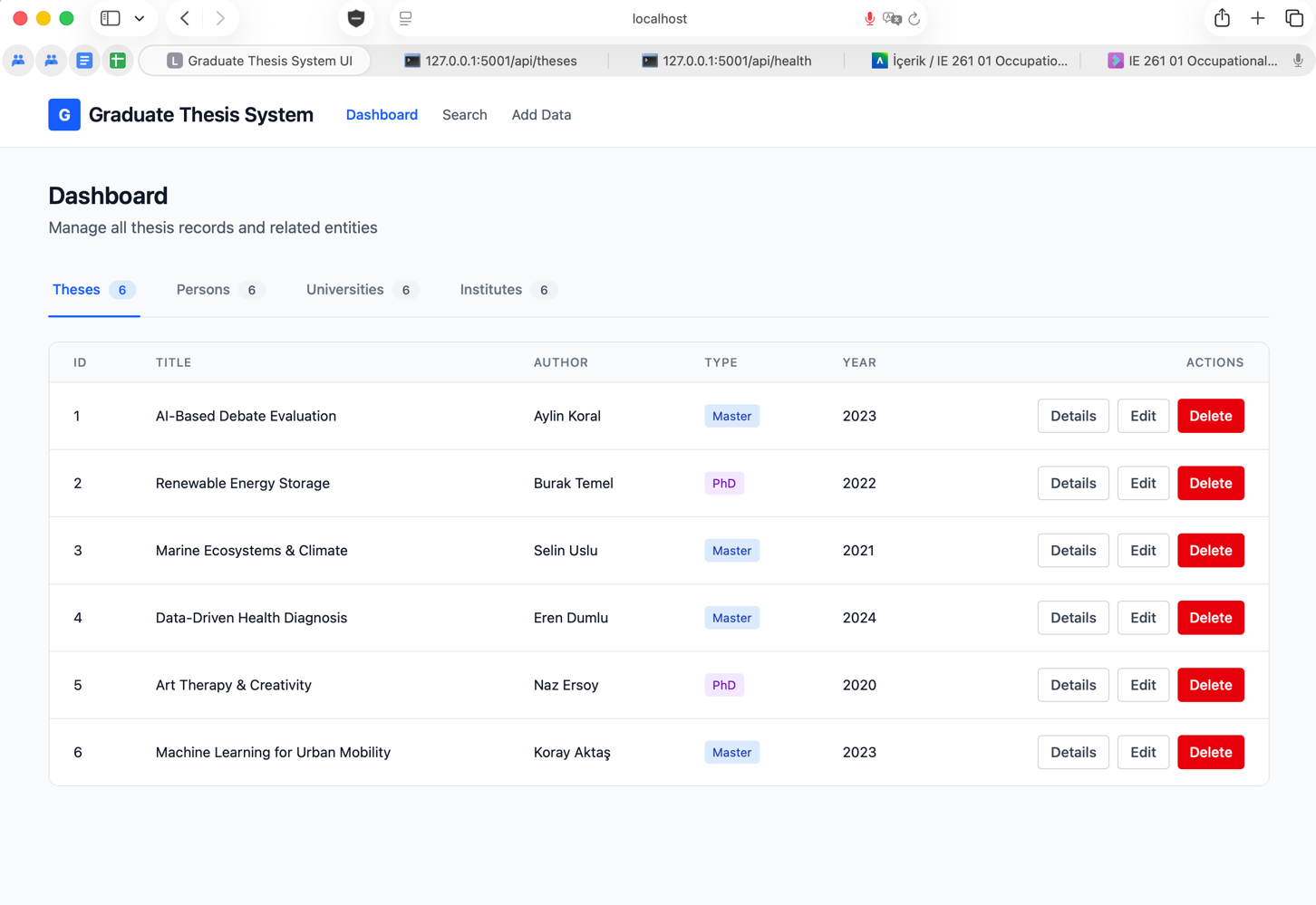
)

return conn

7.3 Functionality Of The Application

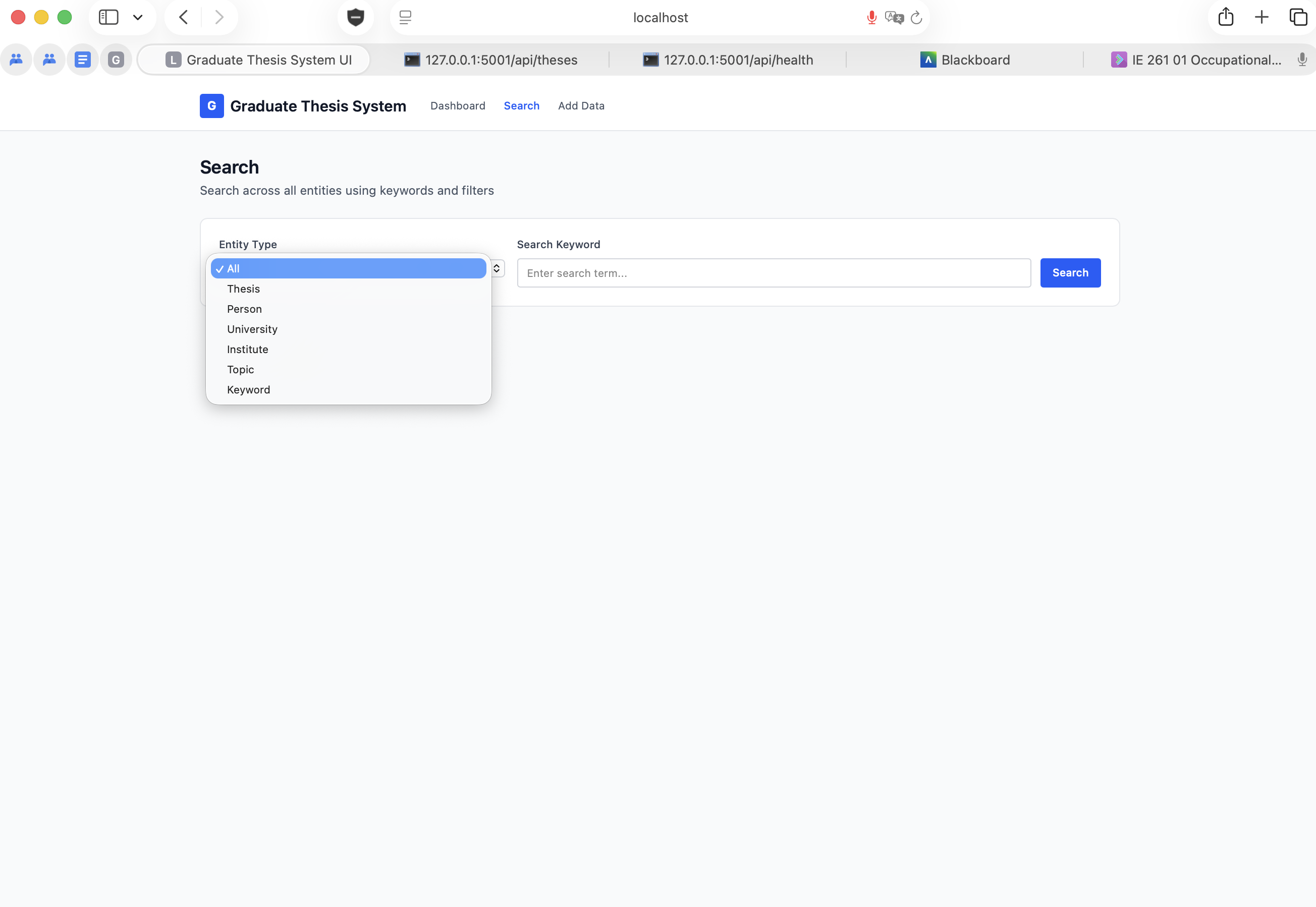
7.3.1 Dashboard (Home)

The home page lists theses, persons, universities, and institutes in separate tabs. Each tab uses data loaded from the API and supports basic actions such as opening details, editing, and deleting records.



7.3.2 Search

The search page allows filtering by entity type. Thesis searches (Thesis/Topic/Keyword) query the backend search endpoint and return theses based on title/abstract, topic, or keyword. Person, University, and Institute searches filter the already loaded lists on the client side. This provides fast filtering without extra requests for non-thesis entities.



7.3.3 Add

The add page provides separate forms for Person, University, Institute, and Thesis. Each form validates required fields and sends a POST request to the backend. Thesis creation also accepts multiple topics and keywords, which are stored in related tables.

A screenshot of a computer

AI-generated content may be incorrect.

7.3.4 Edit

Edit pages allow users to update existing records within the system. The Thesis edit page loads the complete set of thesis information, including metadata, topics, and keywords, and presents it in an editable form. When the user submits the changes, the frontend sends an HTTP PUT request to the corresponding API endpoint, and the backend updates the thesis record along with its associated topics and keywords in the database.

Edit pages for Person, University, and Institute entities enable modification of their respective attributes. All updates are processed through the API, ensuring validation and consistency at the backend level before changes are committed to the PostgreSQL database. This design maintains data integrity while allowing efficient management of academic records.

7.3.4.1 Edit Thesis

A screenshot of a computer

AI-generated content may be incorrect.

7.3.4.2 Edit Person

A screenshot of a computer

AI-generated content may be incorrect.

7.3.4.3 Edit University

A screenshot of a computer

AI-generated content may be incorrect.

7.3.4.4 Edit Institutes

A screenshot of a computer

AI-generated content may be incorrect.

7.3.5 Details

Detail pages provide comprehensive information for each entity in the system. The Thesis details page presents the complete metadata of a thesis, including its title, abstract, year, type, language, page count, associated topics, keywords, and related academic information such as the author and institutional affiliations.

Detail pages for Person, University, and Institute entities display their core information along with the theses associated with them. These pages include cross-links that allow users to navigate easily between related entities, enabling efficient exploration of relationships within the database. This structured presentation supports better understanding of academic connections and enhances overall usability of the Graduate Thesis Management System.

7.3.5.1 Thesis Details

A screenshot of a computer

AI-generated content may be incorrect.

7.3.5.2 Person Details

A screenshot of a computer

AI-generated content may be incorrect.

7.3.5.3 University Details

A screenshot of a computer

AI-generated content may be incorrect.

7.3.5.4 Institute Details

A screenshot of a computer

AI-generated content may be incorrect.

7.3.6 Delete

Delete operations are available from both list views and search result pages within the application. When a user initiates a delete action, the frontend sends an HTTP DELETE request to the corresponding API endpoint, including the unique identifier of the selected record.

The backend processes the request by attempting to remove the record from the PostgreSQL database. Before deletion is finalized, referential integrity constraints are evaluated. If the record is referenced by related entities (for example, an institute associated with one or more theses), the database prevents the operation and returns a constraint violation error.

In such cases, the backend forwards an appropriate error response to the frontend. The user interface then displays a clear and informative message explaining why the deletion cannot be completed. This approach ensures data consistency while providing meaningful feedback to users during delete operations.

7.3.6.1 Delete Thesis

A screenshot of a confirmation

AI-generated content may be incorrect.

7.3.6.2 Delete Person

A screenshot of a confirmation

AI-generated content may be incorrect.

7.3.6.3 Delete University

A screenshot of a computer error

AI-generated content may be incorrect.

7.3.6.4 Delete Institute

A screenshot of a confirmation

AI-generated content may be incorrect.

7.3.7 Error Handling and Feedback

API errors (400/404/500) are surfaced in the UI with short messages so users can understand why an action failed. This includes validation errors, foreign key constraint failures, and missing records.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a chat

AI-generated content may be incorrect.

7.3.8 Technology Summary

Frontend: React (Vite) with route-based navigation.

Backend: Flask REST API with psycopg2 for PostgreSQL access.

Database: PostgreSQL managed through pgAdmin 4.

7.4 Interaction with the Database

In this project, database access is handled through a Flask REST API. The React frontend translates user actions into API requests, and the backend uses psycopg2 to connect to PostgreSQL, execute SQL queries, and return JSON responses.

7.4.1 Database Connection Lifecycle

Database connectivity is handled using the psycopg2 library. For each incoming API request that requires data access, the backend opens a new connection using a centralized helper function (e.g., get\_db\_connection()), creates a cursor, performs the required SQL operations, commits changes when needed, and then closes the connection. This request-scoped pattern prevents stale connections and keeps transactions isolated

import os

import psycopg2

from psycopg2.extras import RealDictCursor

from flask import Flask, render\_template, jsonify, request

app = Flask(\_\_name\_\_)

# DB CONFIG

DB\_HOST = os.getenv("DB\_HOST", "localhost")

DB\_NAME = os.getenv("DB\_NAME", "gtsdb")

DB\_PORT = int(os.getenv("DB\_PORT", "5432"))

DB\_USER = os.getenv("DB\_USER", "postgres")

DB\_PASSWORD = os.getenv("DB\_PASSWORD", "")

def get\_db\_connection():

return psycopg2.connect(

host=DB\_HOST,

database=DB\_NAME,

port=DB\_PORT,

user=DB\_USER,

password=DB\_PASSWORD

)

7.4.2 Dashboard Data Retrieval (Home)

The dashboard loads its data by calling the list endpoints in the API. Theses, persons, universities, and institutes are fetched separately and displayed in tabbed tables. Each list query returns a collection of records ordered by ID, and the frontend maps these results into the table views used on the Home page.

@app.get("/api/theses")

def api\_theses():

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

T.th\_num,

T.title,

EXTRACT(YEAR FROM T.th\_year)::int AS th\_year,

T.th\_type,

P.first\_name || ' ' || P.second\_name AS author

FROM thesis T

JOIN person P ON T.author\_id = P.per\_id

ORDER BY T.th\_num

""")

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

@app.get("/api/persons")

def api\_persons():

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

per\_id AS id,

first\_name AS "firstName",

second\_name AS "secondName",

phone\_num AS "phoneNumber"

FROM person

ORDER BY per\_id

""")

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

@app.get("/api/universities")

def api\_universities():

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

uni\_id AS id,

uni\_name AS "universityName",

uni\_location AS "location"

FROM university

ORDER BY uni\_id

""")

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

@app.get("/api/institutes")

def api\_institutes():

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

I.ins\_id AS id,

I.ins\_name AS "instituteName",

I.uni\_id AS "universityId",

U.uni\_name AS "universityName"

FROM institute I

JOIN university U ON I.uni\_id = U.uni\_id

ORDER BY I.ins\_id

""")

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

7.4.3 Search Queries and Filtering

The search feature supports filtering by entity type. Thesis searches are executed on the backend via the /api/search/ endpoint, which looks up matches in title/abstract, topic, or keyword depending on the selected filter. Person, University, and Institute searches are filtered client‑side using the already loaded lists. Results are displayed in a unified table with the entity type shown for clarity.

@app.post("/api/search")

def api\_search():

body = request.get\_json(silent=True) or {}

keyword = (body.get("keyword") or "").strip()

search\_type = (body.get("type") or "").strip().lower()

param = f"%{keyword}%"

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

if search\_type == "topic":

cur.execute("""

SELECT DISTINCT

T.th\_num,

T.title,

EXTRACT(YEAR FROM T.th\_year)::int AS th\_year,

T.th\_type,

P.first\_name || ' ' || P.second\_name AS author

FROM thesis T

JOIN person P ON T.author\_id = P.per\_id

JOIN topic TP ON TP.th\_num = T.th\_num

WHERE TP.topic\_name ILIKE %s

ORDER BY T.th\_num

""", (param,))

elif search\_type == "keyword":

cur.execute("""

SELECT DISTINCT

T.th\_num,

T.title,

EXTRACT(YEAR FROM T.th\_year)::int AS th\_year,

T.th\_type,

P.first\_name || ' ' || P.second\_name AS author

FROM thesis T

JOIN person P ON T.author\_id = P.per\_id

JOIN keyword K ON K.th\_num = T.th\_num

WHERE K.keyword ILIKE %s

ORDER BY T.th\_num

""", (param,))

else:

cur.execute("""

SELECT

T.th\_num,

T.title,

EXTRACT(YEAR FROM T.th\_year)::int AS th\_year,

T.th\_type,

P.first\_name || ' ' || P.second\_name AS author

FROM thesis T

JOIN person P ON T.author\_id = P.per\_id

WHERE T.title ILIKE %s OR T.abstract ILIKE %s

ORDER BY T.th\_num

""", (param, param))

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

7.4.4 Create Operations (Add / POST)

Create operations are performed through the Add page, which provides separate forms for Person, University, Institute, and Thesis. When a form is submitted, the frontend sends a POST request to the appropriate API endpoint. The backend validates required fields, inserts the new record into the corresponding table, and returns the new ID. For theses, the request also includes topics and keywords, which are stored in their respective tables after the thesis record is created.

@app.post("/api/persons")

def api\_person\_create():

body = request.get\_json(silent=True) or {}

first\_name = (body.get("firstName") or "").strip()

second\_name = (body.get("secondName") or "").strip()

phone\_num = (body.get("phoneNumber") or "").strip()

if not first\_name or not second\_name:

return jsonify({"error": "Missing required fields"}), 400

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

try:

cur.execute("""

INSERT INTO person (first\_name, second\_name, phone\_num)

VALUES (%s, %s, %s)

RETURNING per\_id AS id

""", (first\_name, second\_name, phone\_num or None))

new\_id = cur.fetchone()["id"]

conn.commit()

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"id": new\_id}), 201

@app.post("/api/universities")

def api\_university\_create():

body = request.get\_json(silent=True) or {}

uni\_name = (body.get("universityName") or "").strip()

uni\_location = (body.get("location") or "").strip()

if not uni\_name or not uni\_location:

return jsonify({"error": "Missing required fields"}), 400

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

try:

cur.execute("""

INSERT INTO university (uni\_name, uni\_location)

VALUES (%s, %s)

RETURNING uni\_id AS id

""", (uni\_name, uni\_location))

new\_id = cur.fetchone()["id"]

conn.commit()

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"id": new\_id}), 201

@app.post("/api/institutes")

def api\_institute\_create():

body = request.get\_json(silent=True) or {}

ins\_name = (body.get("instituteName") or "").strip()

uni\_id = body.get("universityId")

if not ins\_name or not uni\_id:

return jsonify({"error": "Missing required fields"}), 400

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

try:

cur.execute("""

INSERT INTO institute (ins\_name, uni\_id)

VALUES (%s, %s)

RETURNING ins\_id AS id

""", (ins\_name, uni\_id))

new\_id = cur.fetchone()["id"]

conn.commit()

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"id": new\_id}), 201

@app.post("/api/theses")

def api\_thesis\_create():

body = request.get\_json(silent=True) or {}

title = (body.get("title") or "").strip()

abstract = (body.get("abstract") or "").strip()

author\_id = body.get("authorId")

thesis\_year = body.get("thesisYear")

thesis\_type = (body.get("thesisType") or "").strip()

university\_id = body.get("universityId")

institute\_id = body.get("instituteId")

page\_count = body.get("pageCount")

language = (body.get("language") or "").strip()

submission\_date = body.get("submissionDate")

topics = body.get("topics") or []

keywords = body.get("keywords") or []

if not title or not abstract or not author\_id or not thesis\_type or not university\_id or not institute\_id or not language:

return jsonify({"error": "Missing required fields"}), 400

th\_year\_date = parse\_date\_from\_year(thesis\_year)

if thesis\_year and not th\_year\_date:

return jsonify({"error": "Invalid thesisYear"}), 400

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

try:

cur.execute("""

INSERT INTO thesis (

title, abstract, author\_id, th\_year, th\_type, uni\_id, ins\_id,

page\_num, th\_language, submission\_date

)

VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)

RETURNING th\_num AS id

""", (

title,

abstract,

author\_id,

th\_year\_date,

thesis\_type,

university\_id,

institute\_id,

page\_count,

language,

submission\_date or None,

))

new\_id = cur.fetchone()["id"]

for topic in [t.strip() for t in topics if str(t).strip()]:

cur.execute("INSERT INTO topic (th\_num, topic\_name) VALUES (%s, %s)", (new\_id, topic))

for keyword in [k.strip() for k in keywords if str(k).strip()]:

cur.execute("INSERT INTO keyword (th\_num, keyword) VALUES (%s, %s)", (new\_id, keyword))

conn.commit()

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"id": new\_id}), 201

7.4.5 Read Operations (Details / GET)

Detail views retrieve a single record by its primary key. The backend executes SELECT queries with JOINs to return full information for the selected entity. For theses, the detail endpoint also aggregates related topics and keywords. The frontend displays the returned object in the corresponding details page.

@app.get("/api/theses/<int:th\_num>")

def api\_thesis\_detail(th\_num: int):

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

T.th\_num AS id,

T.title,

T.abstract,

T.author\_id AS "authorId",

P.first\_name || ' ' || P.second\_name AS "authorName",

EXTRACT(YEAR FROM T.th\_year)::int AS "thesisYear",

T.th\_type AS "thesisType",

T.uni\_id AS "universityId",

U.uni\_name AS "universityName",

T.ins\_id AS "instituteId",

I.ins\_name AS "instituteName",

T.page\_num AS "pageCount",

T.th\_language AS "language",

T.submission\_date AS "submissionDate",

COALESCE(

array\_agg(DISTINCT TP.topic\_name) FILTER (WHERE TP.topic\_name IS NOT NULL),

ARRAY[]::text[]

) AS topics,

COALESCE(

array\_agg(DISTINCT K.keyword) FILTER (WHERE K.keyword IS NOT NULL),

ARRAY[]::text[]

) AS keywords

FROM thesis T

JOIN person P ON T.author\_id = P.per\_id

JOIN university U ON T.uni\_id = U.uni\_id

JOIN institute I ON T.ins\_id = I.ins\_id

LEFT JOIN topic TP ON TP.th\_num = T.th\_num

LEFT JOIN keyword K ON K.th\_num = T.th\_num

WHERE T.th\_num = %s

GROUP BY

T.th\_num,

P.first\_name,

P.second\_name,

U.uni\_name,

I.ins\_name

""", (th\_num,))

thesis = cur.fetchone()

cur.close()

conn.close()

if thesis is None:

return jsonify({"error": "Thesis not found"}), 404

return jsonify(thesis)

@app.get("/api/persons/<int:per\_id>")

def api\_person\_detail(per\_id: int):

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

per\_id AS id,

first\_name AS "firstName",

second\_name AS "secondName",

phone\_num AS "phoneNumber"

FROM person

WHERE per\_id = %s

""", (per\_id,))

person = cur.fetchone()

cur.close()

conn.close()

if person is None:

return jsonify({"error": "Person not found"}), 404

return jsonify(person)

@app.get("/api/universities/<int:uni\_id>")

def api\_university\_detail(uni\_id: int):

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

uni\_id AS id,

uni\_name AS "universityName",

uni\_location AS "location"

FROM university

WHERE uni\_id = %s

""", (uni\_id,))

university = cur.fetchone()

cur.close()

conn.close()

if university is None:

return jsonify({"error": "University not found"}), 404

return jsonify(university)

@app.get("/api/institutes/<int:ins\_id>")

def api\_institute\_detail(ins\_id: int):

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

I.ins\_id AS id,

I.ins\_name AS "instituteName",

I.uni\_id AS "universityId",

U.uni\_name AS "universityName"

FROM institute I

JOIN university U ON I.uni\_id = U.uni\_id

WHERE I.ins\_id = %s

""", (ins\_id,))

institute = cur.fetchone()

cur.close()

conn.close()

if institute is None:

return jsonify({"error": "Institute not found"}), 404

return jsonify(institute)

7.4.6 Update Operations (Edit / PUT)

Update operations are triggered from the Edit pages. The frontend sends a PUT request with the updated fields to the relevant API endpoint. The backend validates the input, updates the record in the database, and returns a success response. For theses, associated topics and keywords are refreshed by removing old entries and inserting the updated lists.

@app.put("/api/persons/<int:per\_id>")

def api\_person\_update(per\_id: int):

body = request.get\_json(silent=True) or {}

first\_name = (body.get("firstName") or "").strip()

second\_name = (body.get("secondName") or "").strip()

phone\_num = (body.get("phoneNumber") or "").strip()

if not first\_name or not second\_name:

return jsonify({"error": "Missing required fields"}), 400

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("""

UPDATE person

SET first\_name = %s,

second\_name = %s,

phone\_num = %s

WHERE per\_id = %s

""", (first\_name, second\_name, phone\_num or None, per\_id))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "Person not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

@app.put("/api/universities/<int:uni\_id>")

def api\_university\_update(uni\_id: int):

body = request.get\_json(silent=True) or {}

uni\_name = (body.get("universityName") or "").strip()

uni\_location = (body.get("location") or "").strip()

if not uni\_name or not uni\_location:

return jsonify({"error": "Missing required fields"}), 400

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("""

UPDATE university

SET uni\_name = %s,

uni\_location = %s

WHERE uni\_id = %s

""", (uni\_name, uni\_location, uni\_id))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "University not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

@app.put("/api/institutes/<int:ins\_id>")

def api\_institute\_update(ins\_id: int):

body = request.get\_json(silent=True) or {}

ins\_name = (body.get("instituteName") or "").strip()

uni\_id = body.get("universityId")

if not ins\_name or not uni\_id:

return jsonify({"error": "Missing required fields"}), 400

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("""

UPDATE institute

SET ins\_name = %s,

uni\_id = %s

WHERE ins\_id = %s

""", (ins\_name, uni\_id, ins\_id))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "Institute not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

@app.put("/api/theses/<int:th\_num>")

def api\_thesis\_update(th\_num: int):

body = request.get\_json(silent=True) or {}

title = (body.get("title") or "").strip()

abstract = (body.get("abstract") or "").strip()

author\_id = body.get("authorId")

thesis\_year = body.get("thesisYear")

thesis\_type = (body.get("thesisType") or "").strip()

university\_id = body.get("universityId")

institute\_id = body.get("instituteId")

page\_count = body.get("pageCount")

language = (body.get("language") or "").strip()

submission\_date = body.get("submissionDate")

topics = body.get("topics") or []

keywords = body.get("keywords") or []

if not title or not abstract or not author\_id or not thesis\_type or not university\_id or not institute\_id or not language:

return jsonify({"error": "Missing required fields"}), 400

th\_year\_date = parse\_date\_from\_year(thesis\_year)

if thesis\_year and not th\_year\_date:

return jsonify({"error": "Invalid thesisYear"}), 400

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("""

UPDATE thesis

SET title = %s,

abstract = %s,

author\_id = %s,

th\_year = %s,

th\_type = %s,

uni\_id = %s,

ins\_id = %s,

page\_num = %s,

th\_language = %s,

submission\_date = %s

WHERE th\_num = %s

""", (

title,

abstract,

author\_id,

th\_year\_date,

thesis\_type,

university\_id,

institute\_id,

page\_count,

language,

submission\_date or None,

th\_num,

))

if cur.rowcount == 0:

conn.rollback()

return jsonify({"error": "Thesis not found"}), 404

cur.execute("DELETE FROM topic WHERE th\_num = %s", (th\_num,))

cur.execute("DELETE FROM keyword WHERE th\_num = %s", (th\_num,))

for topic in [t.strip() for t in topics if str(t).strip()]:

cur.execute("INSERT INTO topic (th\_num, topic\_name) VALUES (%s, %s)", (th\_num, topic))

for keyword in [k.strip() for k in keywords if str(k).strip()]:

cur.execute("INSERT INTO keyword (th\_num, keyword) VALUES (%s, %s)", (th\_num, keyword))

conn.commit()

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

7.4.7 Delete Operations (DELETE)

Delete operations are triggered from list and search views. The frontend sends a DELETE request to the appropriate API endpoint. The backend removes the record and returns a success response; if the record is referenced by other tables, a foreign key constraint error is returned and displayed in the UI.

@app.delete("/api/persons/<int:per\_id>")

def api\_person\_delete(per\_id: int):

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("DELETE FROM person WHERE per\_id = %s", (per\_id,))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "Person not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

@app.delete("/api/universities/<int:uni\_id>")

def api\_university\_delete(uni\_id: int):

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("DELETE FROM university WHERE uni\_id = %s", (uni\_id,))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "University not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

@app.delete("/api/institutes/<int:ins\_id>")

def api\_institute\_delete(ins\_id: int):

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("DELETE FROM institute WHERE ins\_id = %s", (ins\_id,))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "Institute not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

@app.delete("/api/theses/<int:th\_num>")

def api\_thesis\_delete(th\_num: int):

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("DELETE FROM keyword WHERE th\_num = %s", (th\_num,))

cur.execute("DELETE FROM topic WHERE th\_num = %s", (th\_num,))

cur.execute("DELETE FROM thesis WHERE th\_num = %s", (th\_num,))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "Thesis not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

7.4.8 Referential Integrity and Constraint Handling

The database schema uses foreign key constraints to protect relationships between core entities such as Thesis–Institute, Thesis–University, and Thesis–Person. These constraints prevent operations that would leave related data in an inconsistent state. For example, if a user tries to delete an institute that is still referenced by a thesis, PostgreSQL rejects the request because it would break the existing relationship.

The backend handles such cases by wrapping database write operations in transaction blocks. If an error occurs, the transaction is rolled back so that no partial changes are saved. The error message returned by PostgreSQL is captured and sent back to the client as a structured response. The frontend displays this message to the user, explaining why the action failed.

This design guarantees that data integrity is always enforced at the database level while still providing clear feedback to the user. It prevents orphaned records, preserves valid links between tables, and ensures that the system remains consistent even under invalid user actions.

7.4.9 Transaction Management (Commit/Rollback)

All write operations are executed within explicit transaction boundaries. When a create, update, or delete request is processed, the backend starts a transaction, performs the SQL statements, and commits the changes only if the entire operation succeeds. If any error occurs—such as invalid input or a constraint violation—the transaction is rolled back to prevent partial updates. This ensures atomicity and keeps the database consistent, especially for multi‑step operations like thesis creation (which also inserts topics and keywords) or thesis updates (which replace related topics/keywords).

7.4.10 Input Validation and Parameterized Queries

Input validation is applied on the backend to ensure required fields are present and values are well‑formed before any database operation is executed. For example, missing mandatory fields or invalid year values are rejected with clear error messages.

All SQL statements use parameterized queries rather than string concatenation. This prevents SQL injection and guarantees that user input is treated as data, not executable SQL. Combined with validation, this approach improves both data quality and application security.

8. Conclusion

This project delivers a functional Graduate Thesis System that integrates a PostgreSQL database with a Flask REST API and a React single‑page application. The implemented modules cover the core workflows of listing, searching, adding, editing, and deleting records, with a clear and consistent user interface for navigating thesis‑related data.

Data integrity is preserved through database constraints, transaction management, and input validation. These safeguards ensure that relationships remain consistent, invalid operations are rejected safely, and multi‑step operations complete atomically without partial updates.

Overall, the implementation aligns with the designed schema and fulfills the essential requirements of the system. The architecture is organized and maintainable, providing a stable foundation.

9. Appendix

app.py:

import os

import psycopg2

from psycopg2.extras import RealDictCursor

from flask import Flask, render\_template, jsonify, request

app = Flask(\_\_name\_\_)

# DB CONFIG

DB\_HOST = os.getenv("DB\_HOST", "localhost")

DB\_NAME = os.getenv("DB\_NAME", "gtsdb")

DB\_PORT = int(os.getenv("DB\_PORT", "5432"))

DB\_USER = os.getenv("DB\_USER", "postgres")

DB\_PASSWORD = os.getenv("DB\_PASSWORD", "")

def get\_db\_connection():

return psycopg2.connect(

host=DB\_HOST,

database=DB\_NAME,

port=DB\_PORT,

user=DB\_USER,

password=DB\_PASSWORD

)

# --- HEALTH CHECK (DB bağlı mı diye) ---

@app.get("/api/health")

def api\_health():

conn = get\_db\_connection()

cur = conn.cursor()

cur.execute("SELECT 1;")

v = cur.fetchone()[0]

cur.close()

conn.close()

return jsonify({"ok": True, "db": DB\_NAME, "test": v})

def parse\_date\_from\_year(year\_value):

try:

year\_int = int(year\_value)

except (TypeError, ValueError):

return None

return f"{year\_int}-01-01"

# --- HTML HOME (Jinja) ---

@app.route("/")

def home():

conn = get\_db\_connection()

cur = conn.cursor()

# Theses

cur.execute("""

SELECT T.th\_num, T.title, T.th\_type, T.th\_year, T.th\_language,

T.page\_num, P.first\_name || ' ' || P.second\_name AS author\_name

FROM thesis T

JOIN person P ON T.author\_id = P.per\_id

ORDER BY T.th\_num

""")

theses = cur.fetchall()

cur.close()

conn.close()

return render\_template("home.html", theses=theses)

# --- API: LIST THESES (React için) ---

@app.get("/api/theses")

def api\_theses():

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

T.th\_num,

T.title,

EXTRACT(YEAR FROM T.th\_year)::int AS th\_year,

T.th\_type,

P.first\_name || ' ' || P.second\_name AS author

FROM thesis T

JOIN person P ON T.author\_id = P.per\_id

ORDER BY T.th\_num

""")

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

# --- API: LIST PERSONS ---

@app.get("/api/persons")

def api\_persons():

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

per\_id AS id,

first\_name AS "firstName",

second\_name AS "secondName",

phone\_num AS "phoneNumber"

FROM person

ORDER BY per\_id

""")

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

# --- API: LIST UNIVERSITIES ---

@app.get("/api/universities")

def api\_universities():

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

uni\_id AS id,

uni\_name AS "universityName",

uni\_location AS "location"

FROM university

ORDER BY uni\_id

""")

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

# --- API: LIST INSTITUTES ---

@app.get("/api/institutes")

def api\_institutes():

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

I.ins\_id AS id,

I.ins\_name AS "instituteName",

I.uni\_id AS "universityId",

U.uni\_name AS "universityName"

FROM institute I

JOIN university U ON I.uni\_id = U.uni\_id

ORDER BY I.ins\_id

""")

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

# --- API: CREATE PERSON ---

@app.post("/api/persons")

def api\_person\_create():

body = request.get\_json(silent=True) or {}

first\_name = (body.get("firstName") or "").strip()

second\_name = (body.get("secondName") or "").strip()

phone\_num = (body.get("phoneNumber") or "").strip()

if not first\_name or not second\_name:

return jsonify({"error": "Missing required fields"}), 400

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

try:

cur.execute("""

INSERT INTO person (first\_name, second\_name, phone\_num)

VALUES (%s, %s, %s)

RETURNING per\_id AS id

""", (first\_name, second\_name, phone\_num or None))

new\_id = cur.fetchone()["id"]

conn.commit()

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"id": new\_id}), 201

# --- API: UPDATE PERSON ---

@app.put("/api/persons/<int:per\_id>")

def api\_person\_update(per\_id: int):

body = request.get\_json(silent=True) or {}

first\_name = (body.get("firstName") or "").strip()

second\_name = (body.get("secondName") or "").strip()

phone\_num = (body.get("phoneNumber") or "").strip()

if not first\_name or not second\_name:

return jsonify({"error": "Missing required fields"}), 400

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("""

UPDATE person

SET first\_name = %s,

second\_name = %s,

phone\_num = %s

WHERE per\_id = %s

""", (first\_name, second\_name, phone\_num or None, per\_id))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "Person not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

# --- API: DELETE PERSON ---

@app.delete("/api/persons/<int:per\_id>")

def api\_person\_delete(per\_id: int):

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("DELETE FROM person WHERE per\_id = %s", (per\_id,))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "Person not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

# --- API: CREATE UNIVERSITY ---

@app.post("/api/universities")

def api\_university\_create():

body = request.get\_json(silent=True) or {}

uni\_name = (body.get("universityName") or "").strip()

uni\_location = (body.get("location") or "").strip()

if not uni\_name or not uni\_location:

return jsonify({"error": "Missing required fields"}), 400

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

try:

cur.execute("""

INSERT INTO university (uni\_name, uni\_location)

VALUES (%s, %s)

RETURNING uni\_id AS id

""", (uni\_name, uni\_location))

new\_id = cur.fetchone()["id"]

conn.commit()

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"id": new\_id}), 201

# --- API: UPDATE UNIVERSITY ---

@app.put("/api/universities/<int:uni\_id>")

def api\_university\_update(uni\_id: int):

body = request.get\_json(silent=True) or {}

uni\_name = (body.get("universityName") or "").strip()

uni\_location = (body.get("location") or "").strip()

if not uni\_name or not uni\_location:

return jsonify({"error": "Missing required fields"}), 400

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("""

UPDATE university

SET uni\_name = %s,

uni\_location = %s

WHERE uni\_id = %s

""", (uni\_name, uni\_location, uni\_id))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "University not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

# --- API: DELETE UNIVERSITY ---

@app.delete("/api/universities/<int:uni\_id>")

def api\_university\_delete(uni\_id: int):

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("DELETE FROM university WHERE uni\_id = %s", (uni\_id,))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "University not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

# --- API: CREATE INSTITUTE ---

@app.post("/api/institutes")

def api\_institute\_create():

body = request.get\_json(silent=True) or {}

ins\_name = (body.get("instituteName") or "").strip()

uni\_id = body.get("universityId")

if not ins\_name or not uni\_id:

return jsonify({"error": "Missing required fields"}), 400

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

try:

cur.execute("""

INSERT INTO institute (ins\_name, uni\_id)

VALUES (%s, %s)

RETURNING ins\_id AS id

""", (ins\_name, uni\_id))

new\_id = cur.fetchone()["id"]

conn.commit()

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"id": new\_id}), 201

# --- API: UPDATE INSTITUTE ---

@app.put("/api/institutes/<int:ins\_id>")

def api\_institute\_update(ins\_id: int):

body = request.get\_json(silent=True) or {}

ins\_name = (body.get("instituteName") or "").strip()

uni\_id = body.get("universityId")

if not ins\_name or not uni\_id:

return jsonify({"error": "Missing required fields"}), 400

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("""

UPDATE institute

SET ins\_name = %s,

uni\_id = %s

WHERE ins\_id = %s

""", (ins\_name, uni\_id, ins\_id))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "Institute not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

# --- API: DELETE INSTITUTE ---

@app.delete("/api/institutes/<int:ins\_id>")

def api\_institute\_delete(ins\_id: int):

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("DELETE FROM institute WHERE ins\_id = %s", (ins\_id,))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "Institute not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

# --- API: CREATE THESIS ---

@app.post("/api/theses")

def api\_thesis\_create():

body = request.get\_json(silent=True) or {}

title = (body.get("title") or "").strip()

abstract = (body.get("abstract") or "").strip()

author\_id = body.get("authorId")

thesis\_year = body.get("thesisYear")

thesis\_type = (body.get("thesisType") or "").strip()

university\_id = body.get("universityId")

institute\_id = body.get("instituteId")

page\_count = body.get("pageCount")

language = (body.get("language") or "").strip()

submission\_date = body.get("submissionDate")

topics = body.get("topics") or []

keywords = body.get("keywords") or []

if not title or not abstract or not author\_id or not thesis\_type or not university\_id or not institute\_id or not language:

return jsonify({"error": "Missing required fields"}), 400

th\_year\_date = parse\_date\_from\_year(thesis\_year)

if thesis\_year and not th\_year\_date:

return jsonify({"error": "Invalid thesisYear"}), 400

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

try:

cur.execute("""

INSERT INTO thesis (

title, abstract, author\_id, th\_year, th\_type, uni\_id, ins\_id,

page\_num, th\_language, submission\_date

)

VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)

RETURNING th\_num AS id

""", (

title,

abstract,

author\_id,

th\_year\_date,

thesis\_type,

university\_id,

institute\_id,

page\_count,

language,

submission\_date or None,

))

new\_id = cur.fetchone()["id"]

for topic in [t.strip() for t in topics if str(t).strip()]:

cur.execute("INSERT INTO topic (th\_num, topic\_name) VALUES (%s, %s)", (new\_id, topic))

for keyword in [k.strip() for k in keywords if str(k).strip()]:

cur.execute("INSERT INTO keyword (th\_num, keyword) VALUES (%s, %s)", (new\_id, keyword))

conn.commit()

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"id": new\_id}), 201

# --- API: UPDATE THESIS ---

@app.put("/api/theses/<int:th\_num>")

def api\_thesis\_update(th\_num: int):

body = request.get\_json(silent=True) or {}

title = (body.get("title") or "").strip()

abstract = (body.get("abstract") or "").strip()

author\_id = body.get("authorId")

thesis\_year = body.get("thesisYear")

thesis\_type = (body.get("thesisType") or "").strip()

university\_id = body.get("universityId")

institute\_id = body.get("instituteId")

page\_count = body.get("pageCount")

language = (body.get("language") or "").strip()

submission\_date = body.get("submissionDate")

topics = body.get("topics") or []

keywords = body.get("keywords") or []

if not title or not abstract or not author\_id or not thesis\_type or not university\_id or not institute\_id or not language:

return jsonify({"error": "Missing required fields"}), 400

th\_year\_date = parse\_date\_from\_year(thesis\_year)

if thesis\_year and not th\_year\_date:

return jsonify({"error": "Invalid thesisYear"}), 400

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("""

UPDATE thesis

SET title = %s,

abstract = %s,

author\_id = %s,

th\_year = %s,

th\_type = %s,

uni\_id = %s,

ins\_id = %s,

page\_num = %s,

th\_language = %s,

submission\_date = %s

WHERE th\_num = %s

""", (

title,

abstract,

author\_id,

th\_year\_date,

thesis\_type,

university\_id,

institute\_id,

page\_count,

language,

submission\_date or None,

th\_num,

))

if cur.rowcount == 0:

conn.rollback()

return jsonify({"error": "Thesis not found"}), 404

cur.execute("DELETE FROM topic WHERE th\_num = %s", (th\_num,))

cur.execute("DELETE FROM keyword WHERE th\_num = %s", (th\_num,))

for topic in [t.strip() for t in topics if str(t).strip()]:

cur.execute("INSERT INTO topic (th\_num, topic\_name) VALUES (%s, %s)", (th\_num, topic))

for keyword in [k.strip() for k in keywords if str(k).strip()]:

cur.execute("INSERT INTO keyword (th\_num, keyword) VALUES (%s, %s)", (th\_num, keyword))

conn.commit()

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

# --- API: DELETE THESIS ---

@app.delete("/api/theses/<int:th\_num>")

def api\_thesis\_delete(th\_num: int):

conn = get\_db\_connection()

cur = conn.cursor()

try:

cur.execute("DELETE FROM keyword WHERE th\_num = %s", (th\_num,))

cur.execute("DELETE FROM topic WHERE th\_num = %s", (th\_num,))

cur.execute("DELETE FROM thesis WHERE th\_num = %s", (th\_num,))

conn.commit()

if cur.rowcount == 0:

return jsonify({"error": "Thesis not found"}), 404

except Exception as exc:

conn.rollback()

return jsonify({"error": str(exc)}), 400

finally:

cur.close()

conn.close()

return jsonify({"ok": True})

# --- API: THESIS DETAIL ---

@app.get("/api/theses/<int:th\_num>")

def api\_thesis\_detail(th\_num: int):

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

T.th\_num AS id,

T.title,

T.abstract,

T.author\_id AS "authorId",

P.first\_name || ' ' || P.second\_name AS "authorName",

EXTRACT(YEAR FROM T.th\_year)::int AS "thesisYear",

T.th\_type AS "thesisType",

T.uni\_id AS "universityId",

U.uni\_name AS "universityName",

T.ins\_id AS "instituteId",

I.ins\_name AS "instituteName",

T.page\_num AS "pageCount",

T.th\_language AS "language",

T.submission\_date AS "submissionDate",

COALESCE(

array\_agg(DISTINCT TP.topic\_name) FILTER (WHERE TP.topic\_name IS NOT NULL),

ARRAY[]::text[]

) AS topics,

COALESCE(

array\_agg(DISTINCT K.keyword) FILTER (WHERE K.keyword IS NOT NULL),

ARRAY[]::text[]

) AS keywords

FROM thesis T

JOIN person P ON T.author\_id = P.per\_id

JOIN university U ON T.uni\_id = U.uni\_id

JOIN institute I ON T.ins\_id = I.ins\_id

LEFT JOIN topic TP ON TP.th\_num = T.th\_num

LEFT JOIN keyword K ON K.th\_num = T.th\_num

WHERE T.th\_num = %s

GROUP BY

T.th\_num,

P.first\_name,

P.second\_name,

U.uni\_name,

I.ins\_name

""", (th\_num,))

thesis = cur.fetchone()

cur.close()

conn.close()

if thesis is None:

return jsonify({"error": "Thesis not found"}), 404

return jsonify(thesis)

# --- API: SEARCH (title/abstract) ---

@app.post("/api/search")

def api\_search():

body = request.get\_json(silent=True) or {}

keyword = (body.get("keyword") or "").strip()

search\_type = (body.get("type") or "").strip().lower()

param = f"%{keyword}%"

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

if search\_type == "topic":

cur.execute("""

SELECT DISTINCT

T.th\_num,

T.title,

EXTRACT(YEAR FROM T.th\_year)::int AS th\_year,

T.th\_type,

P.first\_name || ' ' || P.second\_name AS author

FROM thesis T

JOIN person P ON T.author\_id = P.per\_id

JOIN topic TP ON TP.th\_num = T.th\_num

WHERE TP.topic\_name ILIKE %s

ORDER BY T.th\_num

""", (param,))

elif search\_type == "keyword":

cur.execute("""

SELECT DISTINCT

T.th\_num,

T.title,

EXTRACT(YEAR FROM T.th\_year)::int AS th\_year,

T.th\_type,

P.first\_name || ' ' || P.second\_name AS author

FROM thesis T

JOIN person P ON T.author\_id = P.per\_id

JOIN keyword K ON K.th\_num = T.th\_num

WHERE K.keyword ILIKE %s

ORDER BY T.th\_num

""", (param,))

else:

cur.execute("""

SELECT

T.th\_num,

T.title,

EXTRACT(YEAR FROM T.th\_year)::int AS th\_year,

T.th\_type,

P.first\_name || ' ' || P.second\_name AS author

FROM thesis T

JOIN person P ON T.author\_id = P.per\_id

WHERE T.title ILIKE %s OR T.abstract ILIKE %s

ORDER BY T.th\_num

""", (param, param))

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

# --- API: PERSON DETAIL ---

@app.get("/api/persons/<int:per\_id>")

def api\_person\_detail(per\_id: int):

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

per\_id AS id,

first\_name AS "firstName",

second\_name AS "secondName",

phone\_num AS "phoneNumber"

FROM person

WHERE per\_id = %s

""", (per\_id,))

person = cur.fetchone()

cur.close()

conn.close()

if person is None:

return jsonify({"error": "Person not found"}), 404

return jsonify(person)

# --- API: PERSON THESES ---

@app.get("/api/persons/<int:per\_id>/theses")

def api\_person\_theses(per\_id: int):

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

T.th\_num AS id,

T.title,

EXTRACT(YEAR FROM T.th\_year)::int AS "thesisYear",

T.th\_type AS "thesisType",

T.page\_num AS "pageCount"

FROM thesis T

WHERE T.author\_id = %s

ORDER BY T.th\_num

""", (per\_id,))

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

# --- API: UNIVERSITY DETAIL ---

@app.get("/api/universities/<int:uni\_id>")

def api\_university\_detail(uni\_id: int):

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

uni\_id AS id,

uni\_name AS "universityName",

uni\_location AS "location"

FROM university

WHERE uni\_id = %s

""", (uni\_id,))

university = cur.fetchone()

cur.close()

conn.close()

if university is None:

return jsonify({"error": "University not found"}), 404

return jsonify(university)

# --- API: UNIVERSITY INSTITUTES ---

@app.get("/api/universities/<int:uni\_id>/institutes")

def api\_university\_institutes(uni\_id: int):

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

I.ins\_id AS id,

I.ins\_name AS "instituteName",

I.uni\_id AS "universityId"

FROM institute I

WHERE I.uni\_id = %s

ORDER BY I.ins\_id

""", (uni\_id,))

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

# --- API: UNIVERSITY THESES ---

@app.get("/api/universities/<int:uni\_id>/theses")

def api\_university\_theses(uni\_id: int):

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

T.th\_num AS id,

T.title,

EXTRACT(YEAR FROM T.th\_year)::int AS "thesisYear",

T.th\_type AS "thesisType"

FROM thesis T

WHERE T.uni\_id = %s

ORDER BY T.th\_num

""", (uni\_id,))

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

# --- API: INSTITUTE DETAIL ---

@app.get("/api/institutes/<int:ins\_id>")

def api\_institute\_detail(ins\_id: int):

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

I.ins\_id AS id,

I.ins\_name AS "instituteName",

I.uni\_id AS "universityId",

U.uni\_name AS "universityName"

FROM institute I

JOIN university U ON I.uni\_id = U.uni\_id

WHERE I.ins\_id = %s

""", (ins\_id,))

institute = cur.fetchone()

cur.close()

conn.close()

if institute is None:

return jsonify({"error": "Institute not found"}), 404

return jsonify(institute)

# --- API: INSTITUTE THESES ---

@app.get("/api/institutes/<int:ins\_id>/theses")

def api\_institute\_theses(ins\_id: int):

conn = get\_db\_connection()

cur = conn.cursor(cursor\_factory=RealDictCursor)

cur.execute("""

SELECT

T.th\_num AS id,

T.title,

EXTRACT(YEAR FROM T.th\_year)::int AS "thesisYear",

T.th\_type AS "thesisType",

T.page\_num AS "pageCount"

FROM thesis T

WHERE T.ins\_id = %s

ORDER BY T.th\_num

""", (ins\_id,))

rows = cur.fetchall()

cur.close()

conn.close()

return jsonify(rows)

if \_\_name\_\_ == "\_\_main\_\_":

# 5001 yaptım çakışıyor emre hocanınkiyle galiba

app.run(debug=True, port=5001)

search.tsx:

import { useEffect, useState } from 'react';

import Layout from '../components/Layout';

import Button from '../components/Button';

import DataTable, { Column } from '../components/DataTable';

import DeleteModal from '../components/DeleteModal';

import { EntityType } from '../types';

export default function Search() {

const [entityType, setEntityType] = useState<EntityType>('All');

const [keyword, setKeyword] = useState('');

const [searchResults, setSearchResults] = useState<any[]>([]);

const [hasSearched, setHasSearched] = useState(false);

const [searchLoading, setSearchLoading] = useState(false);

const [searchError, setSearchError] = useState<string | null>(null);

const [persons, setPersons] = useState<any[]>([]);

const [universities, setUniversities] = useState<any[]>([]);

const [institutes, setInstitutes] = useState<any[]>([]);

const [entitiesError, setEntitiesError] = useState<string | null>(null);

const [deleteModal, setDeleteModal] = useState<{ isOpen: boolean; id: number | null; title: string; entityType: string }>({

isOpen: false,

id: null,

title: '',

entityType: '',

});

useEffect(() => {

const controller = new AbortController();

const loadEntities = async () => {

setEntitiesError(null);

try {

const [personsRes, universitiesRes, institutesRes] = await Promise.all([

fetch('/api/persons', { signal: controller.signal }),

fetch('/api/universities', { signal: controller.signal }),

fetch('/api/institutes', { signal: controller.signal }),

]);

if (!personsRes.ok || !universitiesRes.ok || !institutesRes.ok) {

throw new Error('HTTP error');

}

const [personsData, universitiesData, institutesData] = await Promise.all([

personsRes.json(),

universitiesRes.json(),

institutesRes.json(),

]);

setPersons(Array.isArray(personsData) ? personsData : []);

setUniversities(Array.isArray(universitiesData) ? universitiesData : []);

setInstitutes(Array.isArray(institutesData) ? institutesData : []);

} catch (error) {

if ((error as Error).name !== 'AbortError') {

setEntitiesError('Failed to load people, universities, and institutes from the API.');

}

}

};

loadEntities();

return () => controller.abort();

}, []);

const fetchThesisResults = async (searchType: EntityType) => {

const response = await fetch('/api/search', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ keyword, type: searchType }),

});

if (!response.ok) {

let message = `HTTP ${response.status}`;

try {

const errorBody = await response.json();

if (errorBody?.error) message = errorBody.error;

} catch {

try {

const text = await response.text();

if (text) message = text;

} catch {

// ignore parse errors

}

}

throw new Error(message);

}

const data = await response.json();

return (Array.isArray(data) ? data : []).map((row) => ({

id: row.th\_num,

title: row.title,

thesisYear: row.th\_year,

thesisType: row.th\_type,

authorName: row.author,

\_entityType: 'Thesis',

}));

};

const handleSearch = async () => {

setHasSearched(true);

setSearchLoading(true);

setSearchError(null);

const trimmedKeyword = keyword.trim();

const lowerKeyword = trimmedKeyword.toLowerCase();

let results: any[] = [];

if (!trimmedKeyword) {

setSearchResults([]);

setSearchLoading(false);

setSearchError('Please enter a keyword to filter.');

return;

}

try {

if (entityType === 'All' || entityType === 'Thesis' || entityType === 'Topic' || entityType === 'Keyword') {

const thesisResults = await fetchThesisResults(entityType);

results = [...results, ...thesisResults];

}

} catch (error) {

setSearchError(error instanceof Error ? error.message : 'Failed to search theses from the API.');

}

if (entityType === 'All' || entityType === 'Person') {

const personResults = persons.filter(p =>

p.firstName.toLowerCase().includes(lowerKeyword) ||

p.secondName.toLowerCase().includes(lowerKeyword)

).map(p => ({ ...p, \_entityType: 'Person' }));

results = [...results, ...personResults];

}

if (entityType === 'All' || entityType === 'University') {

const universityResults = universities.filter(u =>

u.universityName.toLowerCase().includes(lowerKeyword) ||

u.location.toLowerCase().includes(lowerKeyword)

).map(u => ({ ...u, \_entityType: 'University' }));

results = [...results, ...universityResults];

}

if (entityType === 'All' || entityType === 'Institute') {

const instituteResults = institutes.filter(i =>

i.instituteName.toLowerCase().includes(lowerKeyword)

).map(i => ({ ...i, \_entityType: 'Institute' }));

results = [...results, ...instituteResults];

}

setSearchResults(results);

setSearchLoading(false);

};

const handleDelete = (id: number) => {

const item = searchResults.find(r => r.id === id);

if (!item) return;

let title = '';

if (item.\_entityType === 'Thesis') title = item.title;

else if (item.\_entityType === 'Person') title = `${item.firstName} ${item.secondName}`;

else if (item.\_entityType === 'University') title = item.universityName;

else if (item.\_entityType === 'Institute') title = item.instituteName;

setDeleteModal({ isOpen: true, id, title, entityType: item.\_entityType });

};

const confirmDelete = async () => {

if (!deleteModal.id || !deleteModal.entityType) {

setDeleteModal({ isOpen: false, id: null, title: '', entityType: '' });

return;

}

try {

setEntitiesError(null);

const type = deleteModal.entityType.toLowerCase();

const deletePath = type === 'thesis'

? `/api/theses/${deleteModal.id}`

: type === 'university'

? `/api/universities/${deleteModal.id}`

: `/api/${type}s/${deleteModal.id}`;

const response = await fetch(deletePath, { method: 'DELETE' });

if (!response.ok) {

let message = `HTTP ${response.status}`;

try {

const errorBody = await response.json();

if (errorBody?.error) message = errorBody.error;

} catch {

try {

const text = await response.text();

if (text) message = text;

} catch {

// ignore parse errors

}

}

setEntitiesError(message);

return;

}

if (type === 'person') {

setPersons((prev) => prev.filter((p) => p.id !== deleteModal.id));

} else if (type === 'university') {

setUniversities((prev) => prev.filter((u) => u.id !== deleteModal.id));

} else if (type === 'institute') {

setInstitutes((prev) => prev.filter((i) => i.id !== deleteModal.id));

}

} catch {

setEntitiesError('Failed to delete from the API.');

} finally {

setDeleteModal({ isOpen: false, id: null, title: '', entityType: '' });

// Re-run search to update results

handleSearch();

}

};

const getColumns = (): Column[] => {

if (entityType === 'All') {

return [

{ key: '\_entityType', label: 'Type', render: (value) => (

<span className="inline-flex px-2 py-1 text-xs rounded bg-gray-100 text-gray-800">

{value}

</span>

)},

{ key: 'id', label: 'ID' },

{ key: 'title', label: 'Information', render: (value, row) => {

if (row.\_entityType === 'Thesis') return <div className="max-w-sm truncate">{row.title}</div>;

if (row.\_entityType === 'Person') return `${row.firstName} ${row.secondName}`;

if (row.\_entityType === 'University') return row.universityName;

if (row.\_entityType === 'Institute') return row.instituteName;

return value;

}},

];

}

if (entityType === 'Thesis' || entityType === 'Topic' || entityType === 'Keyword') {

return [

{ key: 'id', label: 'ID' },

{ key: 'title', label: 'Title', render: (value) => <div className="max-w-md truncate">{value}</div> },

{ key: 'thesisType', label: 'Type', render: (value) => (

<span className={`inline-flex px-2 py-1 text-xs rounded ${

value === 'PhD' ? 'bg-purple-100 text-purple-800' : 'bg-blue-100 text-blue-800'

}`}>

{value}

</span>

)},

{ key: 'thesisYear', label: 'Year' },

];

}

if (entityType === 'Person') {

return [

{ key: 'id', label: 'ID' },

{ key: 'firstName', label: 'First Name' },

{ key: 'secondName', label: 'Last Name' },

{ key: 'phoneNumber', label: 'Phone' },

];

}

if (entityType === 'University') {

return [

{ key: 'id', label: 'ID' },

{ key: 'universityName', label: 'University Name' },

{ key: 'location', label: 'Location' },

];

}

if (entityType === 'Institute') {

return [

{ key: 'id', label: 'ID' },

{ key: 'instituteName', label: 'Institute Name' },

{ key: 'universityId', label: 'University', render: (value, row) => {

if (row.universityName) return row.universityName;

const university = universities.find((u) => u.id === value);

return university ? university.universityName : value;

}},

];

}

return [];

};

return (

<Layout>

<div className="space-y-6">

<div>

<h2 className="text-2xl font-semibold text-gray-900">Search</h2>

<p className="text-gray-600 mt-1">Search across all entities using keywords and filters</p>

</div>

<div className="bg-white p-6 rounded-lg border border-gray-200">

{entitiesError && (

<p className="text-sm text-red-600 mb-4">{entitiesError}</p>

)}

<div className="grid grid-cols-1 md:grid-cols-3 gap-4">

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Entity Type

</label>

<select

value={entityType}

onChange={(e) => setEntityType(e.target.value as EntityType)}

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

>

<option value="All">All</option>

<option value="Thesis">Thesis</option>

<option value="Person">Person</option>

<option value="University">University</option>

<option value="Institute">Institute</option>

<option value="Topic">Topic</option>

<option value="Keyword">Keyword</option>

</select>

</div>

<div className="md:col-span-2">

<label className="block text-sm font-medium text-gray-700 mb-2">

Search Keyword

</label>

<div className="flex gap-3">

<input

type="text"

value={keyword}

onChange={(e) => setKeyword(e.target.value)}

onKeyDown={(e) => e.key === 'Enter' && handleSearch()}

placeholder="Enter search term..."

className="flex-1 border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

<Button onClick={handleSearch}>

Search

</Button>

</div>

</div>

</div>

</div>

{hasSearched && (

<div>

<div className="flex items-center justify-between mb-4">

<h3 className="text-lg font-medium text-gray-900">

Search Results

<span className="ml-2 text-gray-500">({searchResults.length} found)</span>

</h3>

</div>

{searchError && (

<p className="text-sm text-red-600 mb-3">{searchError}</p>

)}

{searchLoading ? (

<div className="bg-white border border-gray-200 rounded-lg p-12 text-center">

<p className="text-gray-500">Searching...</p>

</div>

) : searchResults.length === 0 ? (

<div className="bg-white border border-gray-200 rounded-lg p-12 text-center">

<p className="text-gray-500">No results found for "{keyword}"</p>

</div>

) : (

<DataTable

columns={getColumns()}

data={searchResults}

entityType={searchResults[0]?.\_entityType?.toLowerCase() || 'thesis'}

onDelete={handleDelete}

/>

)}

</div>

)}

</div>

<DeleteModal

isOpen={deleteModal.isOpen}

onClose={() => setDeleteModal({ isOpen: false, id: null, title: '', entityType: '' })}

onConfirm={confirmDelete}

title={deleteModal.title}

/>

</Layout>

);

}

vite.config.ts:

import { defineConfig } from "vite";

import { fileURLToPath, URL } from "url";

import tailwindcss from "@tailwindcss/vite";

import react from "@vitejs/plugin-react";

export default defineConfig({

plugins: [

react(),

tailwindcss(),

],

resolve: {

alias: {

"@": fileURLToPath(new URL("./src", import.meta.url)),

},

},

server: {

proxy: {

"/api": {

target: "http://127.0.0.1:5001",

changeOrigin: true,

},

},

},

});

add.tsx:

import { useEffect, useState } from 'react';

import { useNavigate } from 'react-router-dom';

import Layout from '../components/Layout';

import Button from '../components/Button';

type AddEntityType = 'Person' | 'University' | 'Institute' | 'Thesis';

type PersonRow = {

id: number;

firstName: string;

secondName: string;

};

type UniversityRow = {

id: number;

universityName: string;

};

type InstituteRow = {

id: number;

instituteName: string;

universityId: number;

};

export default function Add() {

const navigate = useNavigate();

const [entityType, setEntityType] = useState<AddEntityType>('Person');

const [topics, setTopics] = useState<string[]>(['']);

const [keywords, setKeywords] = useState<string[]>(['']);

const [persons, setPersons] = useState<PersonRow[]>([]);

const [universities, setUniversities] = useState<UniversityRow[]>([]);

const [institutes, setInstitutes] = useState<InstituteRow[]>([]);

const [entitiesError, setEntitiesError] = useState<string | null>(null);

const [submitError, setSubmitError] = useState<string | null>(null);

useEffect(() => {

const controller = new AbortController();

const loadEntities = async () => {

setEntitiesError(null);

try {

const [personsRes, universitiesRes, institutesRes] = await Promise.all([

fetch('/api/persons', { signal: controller.signal }),

fetch('/api/universities', { signal: controller.signal }),

fetch('/api/institutes', { signal: controller.signal }),

]);

if (!personsRes.ok || !universitiesRes.ok || !institutesRes.ok) {

throw new Error('HTTP error');

}

const [personsData, universitiesData, institutesData] = await Promise.all([

personsRes.json(),

universitiesRes.json(),

institutesRes.json(),

]);

setPersons(Array.isArray(personsData) ? personsData : []);

setUniversities(Array.isArray(universitiesData) ? universitiesData : []);

setInstitutes(Array.isArray(institutesData) ? institutesData : []);

} catch (error) {

if ((error as Error).name !== 'AbortError') {

setEntitiesError('Failed to load dropdown data from the API.');

}

}

};

loadEntities();

return () => controller.abort();

}, []);

const handleSubmit = async (e: React.FormEvent) => {

e.preventDefault();

setSubmitError(null);

const form = e.currentTarget as HTMLFormElement;

const data = new FormData(form);

try {

let endpoint = '';

let payload: Record<string, any> = {};

if (entityType === 'Person') {

endpoint = '/api/persons';

payload = {

firstName: data.get('firstName'),

secondName: data.get('secondName'),

phoneNumber: data.get('phoneNumber'),

};

} else if (entityType === 'University') {

endpoint = '/api/universities';

payload = {

universityName: data.get('universityName'),

location: data.get('location'),

};

} else if (entityType === 'Institute') {

endpoint = '/api/institutes';

payload = {

instituteName: data.get('instituteName'),

universityId: Number(data.get('universityId')),

};

} else if (entityType === 'Thesis') {

endpoint = '/api/theses';

payload = {

title: data.get('title'),

abstract: data.get('abstract'),

authorId: Number(data.get('authorId')),

thesisYear: Number(data.get('thesisYear')),

thesisType: data.get('thesisType'),

universityId: Number(data.get('universityId')),

instituteId: Number(data.get('instituteId')),

pageCount: Number(data.get('pageCount')),

language: data.get('language'),

submissionDate: data.get('submissionDate'),

topics: topics.map((t) => t.trim()).filter(Boolean),

keywords: keywords.map((k) => k.trim()).filter(Boolean),

};

}

const response = await fetch(endpoint, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(payload),

});

if (!response.ok) {

let message = `HTTP ${response.status}`;

try {

const errorBody = await response.json();

if (errorBody?.error) {

message = errorBody.error;

}

} catch {

try {

const text = await response.text();

if (text) message = text;

} catch {

// ignore parse errors

}

}

throw new Error(message);

}

alert(`${entityType} added successfully!`);

navigate('/');

} catch (error) {

setSubmitError(error instanceof Error ? error.message : 'Failed to save. Check the form and try again.');

}

};

const addTopic = () => setTopics([...topics, '']);

const removeTopic = (index: number) => setTopics(topics.filter((\_, i) => i !== index));

const updateTopic = (index: number, value: string) => {

const newTopics = [...topics];

newTopics[index] = value;

setTopics(newTopics);

};

const addKeyword = () => setKeywords([...keywords, '']);

const removeKeyword = (index: number) => setKeywords(keywords.filter((\_, i) => i !== index));

const updateKeyword = (index: number, value: string) => {

const newKeywords = [...keywords];

newKeywords[index] = value;

setKeywords(newKeywords);

};

return (

<Layout>

<div className="space-y-6">

<div>

<h2 className="text-2xl font-semibold text-gray-900">Add New Data</h2>

<p className="text-gray-600 mt-1">Insert new records into the database</p>

</div>

<div className="bg-white p-6 rounded-lg border border-gray-200">

<div className="mb-6">

<label className="block text-sm font-medium text-gray-700 mb-2">

Select Entity Type

</label>

<select

value={entityType}

onChange={(e) => setEntityType(e.target.value as AddEntityType)}

className="w-full md:w-64 border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

>

<option value="Person">Person</option>

<option value="University">University</option>

<option value="Institute">Institute</option>

<option value="Thesis">Thesis</option>

</select>

</div>

<form onSubmit={handleSubmit}>

{entitiesError && (

<p className="text-sm text-red-600 mb-4">{entitiesError}</p>

)}

{submitError && (

<p className="text-sm text-red-600 mb-4">{submitError}</p>

)}

{/\* Person Form \*/}

{entityType === 'Person' && (

<div className="space-y-4">

<div className="grid grid-cols-1 md:grid-cols-2 gap-4">

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

First Name \*

</label>

<input

type="text"

required

name="firstName"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Second Name \*

</label>

<input

type="text"

required

name="secondName"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

</div>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Phone Number \*

</label>

<input

type="tel"

required

name="phoneNumber"

placeholder="+1-555-0100"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

</div>

</div>

)}

{/\* University Form \*/}

{entityType === 'University' && (

<div className="space-y-4">

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

University Name \*

</label>

<input

type="text"

required

name="universityName"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Location \*

</label>

<input

type="text"

required

name="location"

placeholder="City, State/Country"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

</div>

</div>

)}

{/\* Institute Form \*/}

{entityType === 'Institute' && (

<div className="space-y-4">

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Institute Name \*

</label>

<input

type="text"

required

name="instituteName"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

University \*

</label>

<select

required

name="universityId"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

>

<option value="">Select University</option>

{universities.map((u) => (

<option key={u.id} value={u.id}>{u.universityName}</option>

))}

</select>

</div>

</div>

)}

{/\* Thesis Form \*/}

{entityType === 'Thesis' && (

<div className="space-y-4">

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Title \*

</label>

<textarea

required

rows={3}

name="title"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Abstract \*

</label>

<textarea

required

rows={5}

name="abstract"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

</div>

<div className="grid grid-cols-1 md:grid-cols-2 gap-4">

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Author \*

</label>

<select

required

name="authorId"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

>

<option value="">Select Author</option>

{persons.map((p) => (

<option key={p.id} value={p.id}>

{p.firstName} {p.secondName}

</option>

))}

</select>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Thesis Year \*

</label>

<input

type="number"

required

min="1900"

max="2100"

defaultValue={new Date().getFullYear()}

name="thesisYear"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

</div>

</div>

<div className="grid grid-cols-1 md:grid-cols-2 gap-4">

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Thesis Type \*

</label>

<select

required

name="thesisType"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

>

<option value="">Select Type</option>

<option value="Master">Master</option>

<option value="PhD">PhD</option>

</select>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Language \*

</label>

<select

required

name="language"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

>

<option value="">Select Language</option>

<option value="English">English</option>

<option value="Turkish">Turkish</option>

<option value="German">German</option>

<option value="French">French</option>

</select>

</div>

</div>

<div className="grid grid-cols-1 md:grid-cols-2 gap-4">

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

University \*

</label>

<select

required

name="universityId"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

>

<option value="">Select University</option>

{universities.map((u) => (

<option key={u.id} value={u.id}>{u.universityName}</option>

))}

</select>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Institute \*

</label>

<select

required

name="instituteId"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

>

<option value="">Select Institute</option>

{institutes.map((i) => (

<option key={i.id} value={i.id}>{i.instituteName}</option>

))}

</select>

</div>

</div>

<div className="grid grid-cols-1 md:grid-cols-2 gap-4">

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Page Count \*

</label>

<input

type="number"

required

min="1"

name="pageCount"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Submission Date \*

</label>

<input

type="date"

required

name="submissionDate"

className="w-full border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

</div>

</div>

{/\* Topics \*/}

<div>

<div className="flex items-center justify-between mb-2">

<label className="block text-sm font-medium text-gray-700">

Topics

</label>

<Button type="button" size="sm" onClick={addTopic}>

+ Add Topic

</Button>

</div>

<div className="space-y-2">

{topics.map((topic, index) => (

<div key={index} className="flex gap-2">

<input

type="text"

value={topic}

onChange={(e) => updateTopic(index, e.target.value)}

placeholder="Enter topic"

className="flex-1 border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

{topics.length > 1 && (

<Button

type="button"

variant="secondary"

size="sm"

onClick={() => removeTopic(index)}

>

Remove

</Button>

)}

</div>

))}

</div>

</div>

{/\* Keywords \*/}

<div>

<div className="flex items-center justify-between mb-2">

<label className="block text-sm font-medium text-gray-700">

Keywords

</label>

<Button type="button" size="sm" onClick={addKeyword}>

+ Add Keyword

</Button>

</div>

<div className="space-y-2">

{keywords.map((keyword, index) => (

<div key={index} className="flex gap-2">

<input

type="text"

value={keyword}

onChange={(e) => updateKeyword(index, e.target.value)}

placeholder="Enter keyword"

className="flex-1 border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

{keywords.length > 1 && (

<Button

type="button"

variant="secondary"

size="sm"

onClick={() => removeKeyword(index)}

>

Remove

</Button>

)}

</div>

))}

</div>

</div>

</div>

)}

<div className="flex justify-end gap-3 mt-6 pt-6 border-t border-gray-200">

<Button type="button" variant="secondary" onClick={() => navigate('/')}>

Cancel

</Button>

<Button type="submit">

Save {entityType}

</Button>

</div>

</form>

</div>

</div>

</Layout>

);

}