SWE573 Software Development Practice - Final Project Report

Name: Zohreh Razavi

Course: SWE573 Software Development Practice, Spring 2025

Date: 18 May 2025

Project Name: Connect The Dots

Deployment URL: https://connectthedots.up.railway.app/

GitHub Repo URL: https://github.com/zohrehrazavi/SWE-573-SoftwareDevPractice

Git Tag Version URL:
https://github.com/zohrehrazavi/SWE-573-SoftwareDevPractice/releases/tag/v0.9

Demo Video:
https://drive.google.com/file/d/1Iymv8TuBVN6wqZJIQcPFhsJKNFGGpmdv/view?usp=sharing

HONOR CODE

Related to the submission of all the project deliverables for the Swe573 Spring 2025 semester project reported in this report, I Zohreh Razavi declare that:

- I am a student in the Software Engineering MS program at Bogazici University and am registered for SWE573 during the Spring 2025 semester.
- All the material that I am submitting related to my project (including but not limited to the project repository, the final project report, and supplementary documents) has been exclusively prepared by myself.
- I have prepared this material individually without the assistance of anyone else with the exception of permitted peer assistance which I have explicitly disclosed in this report.

Zohreh Razavi

Usernames for Testing

Test account credentials:

- *Zohreh - pnh4vTML74vYZAd*
- *Claire.thorne - 5xj4waUfAhvrc4c*
- *nina.brown - iso9001*

Third-party Software & License Declarations

This project uses the following libraries and services under valid open-source licenses:

- Django (BSD License)
- PostgreSQL (PostgreSQL License)
- Docker & Docker Compose (Apache 2.0)
- Node.js and Jest (MIT)
- Wikidata API (CC0)

Table of Contents

1. Overview

Connect The Dots is a visual knowledge-mapping tool that helps researchers and investigators structure complex information by creating nodes (entities) and linking them semantically. Users can:

- Create boards and nodes
- Enrich nodes via Wikidata
- Add properties manually
- View all connections in an interactive graph-like UI

2. Software Requirements Specification

Functional Requirements

- **User Authentication:**
  - Register and log in securely via Django Auth.
  - Session-based login with CSRF protection.

- **Board Management:**
  - Create boards with unique names and optional descriptions.
  - Add up to 5 descriptive tags per board (validated).
  - Allow users to view their own boards on login.

- **Node Management:**
  - Add nodes under a board with a name and optional description.
  - View node detail pages and see attached properties.
  - Automatically enrich nodes using data fetched from **Wikidata** (SPARQL queries).
  - Support **manual property addition** (e.g., gender, country, occupation) if Wikidata info is missing.

- **Semantic Linking:**
  - Users can **create edges (connections)** between nodes with a label (like "works with", "born in").
  - Each edge is saved per board and visible in node detail views.

- **Contribution & Collaboration:**
  - Users can **send edit requests** to suggest changes on nodes they don't own.
  - **Board owners can approve or ignore** edit requests.
  - **Board editors** (collaborators) can be added using the `BoardEditor` model.
  - A **Contribution Message Panel** allows team members to chat on the sidebar of a board.
  - **Message storage** supports editing and deleting of own messages.

- **Graph Visualization:**
  - Nodes and connections are visualized on the board.
  - Clicking on a node opens its detail view.

Non-functional Requirements

- **Deployment & Accessibility:**
  - Deployed on **Railway** at connectthedots.up.railway.app
  - Publicly accessible and responsive.
- **Dockerized Architecture:**

    - Docker Compose runs both Django backend and PostgreSQL DB in isolated containers.
- **Database:**
    - Uses **PostgreSQL** for storing user data, boards, nodes, relationships, and messages.
- **Security & Privacy:**
    - Django middleware secures user sessions.
    - CSRF tokens protect all form submissions.
    - Only authorized users can edit their boards or contribute via requests.
- **Scalability & Maintainability:**
    - Modular app structure (separated `nodes`, `user_auth`, etc.).
    - Clean URL routing and model separation.

3. Design Documents

Architecture:

- Backend: Django
- Frontend: HTML, CSS, JavaScript
- Database: PostgreSQL
- Deployment: Railway + Docker Compose

Entity Structure:

- `User → Board → Node → Property`
- Node enrichment happens via SPARQL queries to Wikidata

4. Requirement Status

Requirement	Status	Notes
Create research boards with title & description	✅ Completed	Boards are created with title, description, and tags
Add nodes to boards (research entities)	✅ Completed	Node creation via form
Suggest relevant entries from Wikidata	✅ Completed	SPARQL integration via Wikidata
Allow editing/approval/removal of properties	✅ Completed	Manual property addition UI available

Connect and disconnect nodes (graph edges)	✅ Completed	ManualEdge model supports labeled connections
Graph visualization of boards and node relationships	✅ Completed	Nodes are displayed in a board structure
Support user-generated content	✅ Completed	Users can input properties manually
Search/filter boards by tags	✅ Completed	Tags stored as JSON, validated (max 5)
Register/login as a user	✅ Completed	Django auth integration
Guest view of boards (unauthenticated read access)	❌ Not Implemented	Requires view permissions setup
Allow collaboration (edit/add to boards by others)	✅ Completed	`BoardEditor` and `EditRequest` models implemented
Chat/discussion feature per board	✅ Completed	ContributionMessage model stores sidebar discussions
Persistent, threaded discussion (not live chat)	✅ Completed	Messages persist and support delete/edit
Auto-extract objects (e.g., name, place) from input	❌ Not Implemented	NLP/LLM parsing not included in MVP
Real-time update of contributions	❌ Not Implemented	Manual refresh required
Responsive web interface	✅ Completed	Works on web; manually styled UI
English-only interface	✅ Completed	No multilingual support implemented
Optional LLM/AI-based suggestion system	❌ Not Implemented	Not in current scope

5. Deployment Details

- **URL:** https://connectthedots.up.railway.app/
- **Dockerized:** Yes
- Docker containers run Django backend and PostgreSQL database

6. Installation Instructions

This section describes how to set up and run the project either using Docker or manually for local development.

Option 1: Docker Installation (Recommended)

1. **Install Docker**
   Download and install Docker Desktop:
   https://www.docker.com/products/docker-desktop
2. **Clone the Repository**
   git clone https://github.com/zohrehrazavi/SWE-573-SoftwareDevPractice.git

   cd SWE-573-SoftwareDevPractice

3. **Set Up Environment Variables**
   If `.env` does not exist, copy the example file:
   cp .env.example .env
4. **Build and Run the Containers**
   docker-compose up --build
5. The app will be accessible at:
   http://localhost:8001

Apply Migrations
In a new terminal:
docker-compose exec web python manage.py migrate

6. **Create a Superuser (Optional)**
   docker-compose exec web python manage.py createsuperuser
7. You can now access the admin panel at `/admin`.
   Note: All backend commands should be run from within the `web` container.

Option 2: Manual Setup (Development without Docker)

Clone the Repository
git clone https://github.com/zohrehrazavi/SWE-573-SoftwareDevPractice.git
cd SWE-573-SoftwareDevPractice

1. **Install Dependencies**
   It's recommended to use a virtual environment:
   python -m venv venv

source venv/bin/activate # On Windows: venv\Scripts\activate
pip install -r requirements.txt

2. **Apply Migrations**
   python manage.py migrate
3. **Run the Server**
   python manage.py runserver
4. Access the app at:
   http://127.0.0.1:8000

7. User Manual

Access the Platform

1. **Visit the deployment URL:**
   https://connectthedots.up.railway.app

Account Management

2. **Register or log in:**
   - New users can register with a username and password
   - Returning users can securely log in

Board Creation & Management

3. **Create a research board:**
   - Provide a board title, description, and up to 5 tags
   - After creation, boards are listed on your dashboard

4. **Edit your board:**
   - Currently limited to description and tag updates (UI dependent)
   - You are automatically the board's owner

Node Creation & Visualization

5. **Add nodes to a board:**
   - Click into your board and select "Create Node"

- Enter a name and optional description

6. **Fetch from Wikidata:**
   - After creating a node, click **“Fetch from Wikidata”**
   - The system attempts to enrich the node with structured data (e.g., “instance of”, “occupation”)
   - Properties are stored as part of the node detail

7. **Add properties manually:**
   - If Wikidata fetch fails or is incomplete, you can manually enter known fields (e.g., gender, country) using Wikidata IDs or human-readable labels

8. **View node details:**

   - Clicking a node from the board brings you to its **Node Detail Page**
   - This page displays:
     - Basic info (name, description)
     - Properties (fetched and manual)
     - Connected edges (relationships to other nodes)

Semantic Connections

9. **Connect nodes (manual edges):**
   - Use the “Create Connection” form to link two nodes under the same board
   - Assign a custom label like *"collaborates with"*, *"related to"*, or *"inspired by"*
   - These appear as **semantic edges** on the board view and node detail page

Collaboration Features

10. **Send edit requests:**
    - If you are not the owner of a node or board, you can submit an **edit request**
    - This suggests edits to the owner, who can review and apply them

11. **Invite collaborators:**
    - Board owners can add trusted users as **Board Editors**
    - Editors can create and edit content within the board

Discussion & Contributions

12. **Use the contribution message panel:**
    - On the right side of the board view, contributors can open the **chat sidebar**

- Post messages to discuss research ideas, findings, or open questions
- Messages persist and can be edited or deleted by their authors

Navigation & Tags

13. **Search boards by tag and Titles (WIP):**
    - On the home/dashboard, boards can be searched using defined tags
    - This helps organize research boards by topic

8. Test Results

Test Coverage Overview

Comprehensive unit and integration tests were implemented using Django's built-in test framework to validate the core functionality of the system, including user management, board/node operations, permissions, and Wikidata integration.

Test Coverage Summary

Module / Feature	Test Type	Status
`nodes/models.py`	Model creation	Covered
`nodes/views.py`	API endpoints	Covered
Board & Node Management	CRUD, permissions	Covered
Manual Edge Creation	Label and linking	Covered
Wikidata Integration	Search & enrichment	Covered
Property Management	Manual & auto	Covered
User Registration/Login	Auth flow	Covered
Password Reset + Security Questions	Form validation	Covered
Edit Request System	Workflow & access	Covered

Contribution Chat	Add/Edit/Delete	Covered

Test Files

- `backend/user_auth/tests/test_security.py`
  Covers:
  - Registration
  - Login
  - Password reset with security questions
- `backend/nodes/tests/test_board_node.py`
  Covers:
  - Board creation
  - Node creation
  - Edge creation
  - Permissions

- `backend/nodes/tests/test_api.py`
  Covers:
  - Node API list and create endpoints

How to Run Tests

From the project root directory:

```
# Run all tests
python manage.py test backend.user_auth.tests backend.nodes.tests -v 2

# Run specific test modules
python manage.py test backend.user_auth.tests.test_security
python manage.py test backend.nodes.tests.test_board_node
```

Test Statistics

- Total number of test cases: 32
- All tests pass successfully with no errors or warnings

Breakdown

- User Authentication (2 tests)
  - Valid registration and login flows
- Security Questions (6 tests)

    - Validation, password reset flows, case-insensitive answer handling
- Board & Node Management (4 tests)
    - Create/edit nodes and boards
    - Permission logic for collaborators

- API (2 tests)
    - Listing and creating nodes through REST API

Test Categories

1. **Unit Tests**
    - Model and form validations
    - Tag constraints and edge relationships
2. **Integration Tests**
    - Full workflows such as board creation with editors, password reset flows, and edge linking
3. **API Tests**
    - REST endpoint coverage including authentication and expected output formatting

9. References

- Django Docs: https://docs.djangoproject.com/
- SPARQL & Wikidata Query Service: https://query.wikidata.org/
- Railway Hosting: https://railway.app/