

# **<Project Name: Binghamton Housing>**

## **Software Requirements Specification**

**Version <3.0>**

**Repository:**

**<<https://github.com/bucs445spring2025/portfolio-team-13>>**

### **Revision History**

Date	Version	Description	Author
<dd/mmm/yy>	<x.x>	<details>	<name>
02/21/2025	0.0	Complete the introduction of all members of Team 13.	<ul style="list-style-type: none"><li>• Andy Luna</li><li>• Sergio Soria</li><li>• Spencer Mines</li></ul>
02/26/2025	0.0	Recognize the purposes and assign first requirements for the project.	<ul style="list-style-type: none"><li>• Andy Luna</li></ul>
03/03/2025	0.0	First requirement (Prepare a database) fulfilled, assign the next requirement: create a web server with a printed message.	<ul style="list-style-type: none"><li>• Andy Luna</li><li>• Sergio Soria</li></ul>
03/15/2025	0.0	Add and construct the diagrams of the class architecture and app architecture.	<ul style="list-style-type: none"><li>• Spencer Mines</li></ul>
03/19/2025	0.0	Define the roles of every member of the team 13.	<ul style="list-style-type: none"><li>• Andy Luna</li><li>• Sergio Soria</li><li>• Spencer Mines</li></ul>
03/24/2025	0.0	Store recollected data of properties in the database in PostgreSQL	<ul style="list-style-type: none"><li>• Andy Luna</li></ul>
04/02/2025	0.1	Assign and establish a first version of the User Interface	<ul style="list-style-type: none"><li>• Sergio Soria</li></ul>
04/16/2025	1.0	Integrate student registration feature	<ul style="list-style-type: none"><li>• Spencer Mines</li></ul>

# **Table of Contents**

1. Introduction
  - a. Team
  - b. Purpose
  - c. Definitions, Acronyms and Abbreviations
  - d. References
2. Requirements
  - a. Functional Requirements
  - b. Non-Functional Requirements
3. Software Architecture Diagram
  - a. Application Architecture Diagram
  - b. Class Architecture Diagram
4. Testing
5. User Interface
6. Project Status

# Introduction

## a. Team

Andy Luna - Software Lead

- Responsibilities:
  - Overall code quality
  - meets user requirements
  - all documentation: SRS, README, Docstrings, etc.
  - Acceptance Test Procedure
  - Integration tests
  - Testing Coverage
- Skills:
  - Pre-Project:
    - C/C++, Python, Java
  - Post-Project:
    - SQL, Docker, Flask,

Sergio Soria - Front-End Lead

- Responsibilities:
  - GUI and Controller
  - UI Design
  - UX
  - Front end functionality
  - Functional Tests
- Skills:
  - Pre-Project:
    - Python, C / C++, Java
  - Post-Project:
    - TBD

Spencer Mines - Back-End Lead

- Responsibilities
  - DB
  - Models
  - all data and functional API classes
  - Back-End Logic
  - Unit Tests
- Skills:
  - Pre-project:
    - Python, Java, Flask, SQL
  - Post-project
    - TBD

**b. Purpose**

- Binghamton students often struggle to find housing and compatible roommates due to the scattered nature of available resources—ranging from various websites and apps to informal word of mouth. This process can be time-consuming, unreliable, and overwhelming. Students have expressed a need for a centralized, easy-to-use platform that streamlines the search for off-campus housing and connects them with potential roommates. Our goal is to develop a website that addresses these challenges and simplifies the entire experience.

**c. Definitions, Acronyms, and Abbreviations**

- N/A

**d. References**

- N/A

# Requirements

## a. Functional Requirements

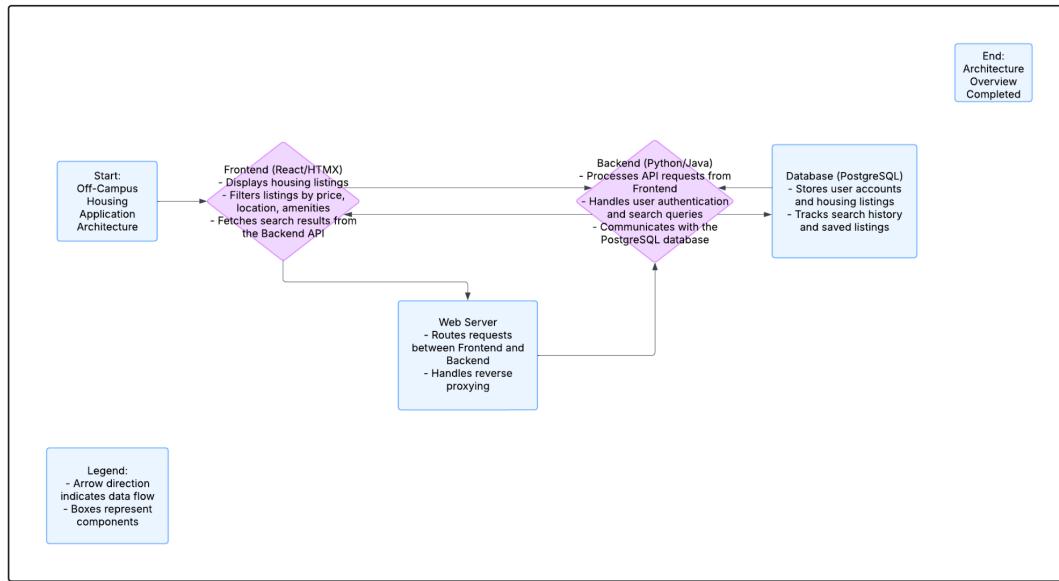
1. Read and Write into storage
  - **Associated User Story:** housing\_search.md
  - **Description:** We need to be able to read and write to storage
  - **Acceptance Test:** Manually read/write to ensure our read/write functions properly
2. Working web server that shows hello world
  - **Associated User Story:** housing\_search.md
  - **Description:** We need a running web page
  - **Acceptance Test:** opening the page on web browser
3. A means of data scraping
  - **Associated User Story:** housing\_search.md
  - **Description:** Find a way to gather listings from different sources to import into our database
  - **Acceptance Test:** checking for correct listing imported
4. A means of displaying one listing
  - **Associated User Story:** first\_time\_renter.md
  - **Description:** Find a way to show database info onto the web page
  - **Acceptance Test:** One listing information displayed on the webpage
5. A means of connecting web scraper to database
  - **Associated User Story:** housing\_search.md
  - **Description:** Find a way to transfer data from web scraping to our PostgreSQL database
  - **Acceptance Test:** Listing information shows up on admin table
6. Multiple web page tabs
  - **Associated User Story:** housing\_search.md
  - **Description:** We need several web page tabs
  - **Acceptance Test:** Localhost and localhost/properties
7. Proper data format in database
  - **Associated User Story:** housing\_search.md
  - **Description:** We need to make sure when data is imported, it is in the proper format
  - **Acceptance Test:** Checking for formatting issues
8. Homepage for users
  - **Associated User Story:** housing\_search.md
  - **Description:** We need a professional looking home page for users. Shows our goal, what we do, and general information
  - **Acceptance Test:** Load homepage and check for information

## b. Non-Functional Requirements

1. Testing Stability
  - **Associated User Story:** housing\_search.md
  - **Description:** Multiple users are able to reliably access the web server
  - **Acceptance Test:** Multiple browsers open
2. Data permanence
  - **Associated User Story:** housing\_search.md
  - **Description:** We need to make sure listing information stays even when we log out
  - **Acceptance Test:** Listing information displayed even after logging out
3. Usability
  - **Associated User Story:** first\_time\_renter.md
  - **Description:** The interface should be usable by students
  - **Acceptance Test:** Testing with 5 users results in 80% or more task completion without guidance

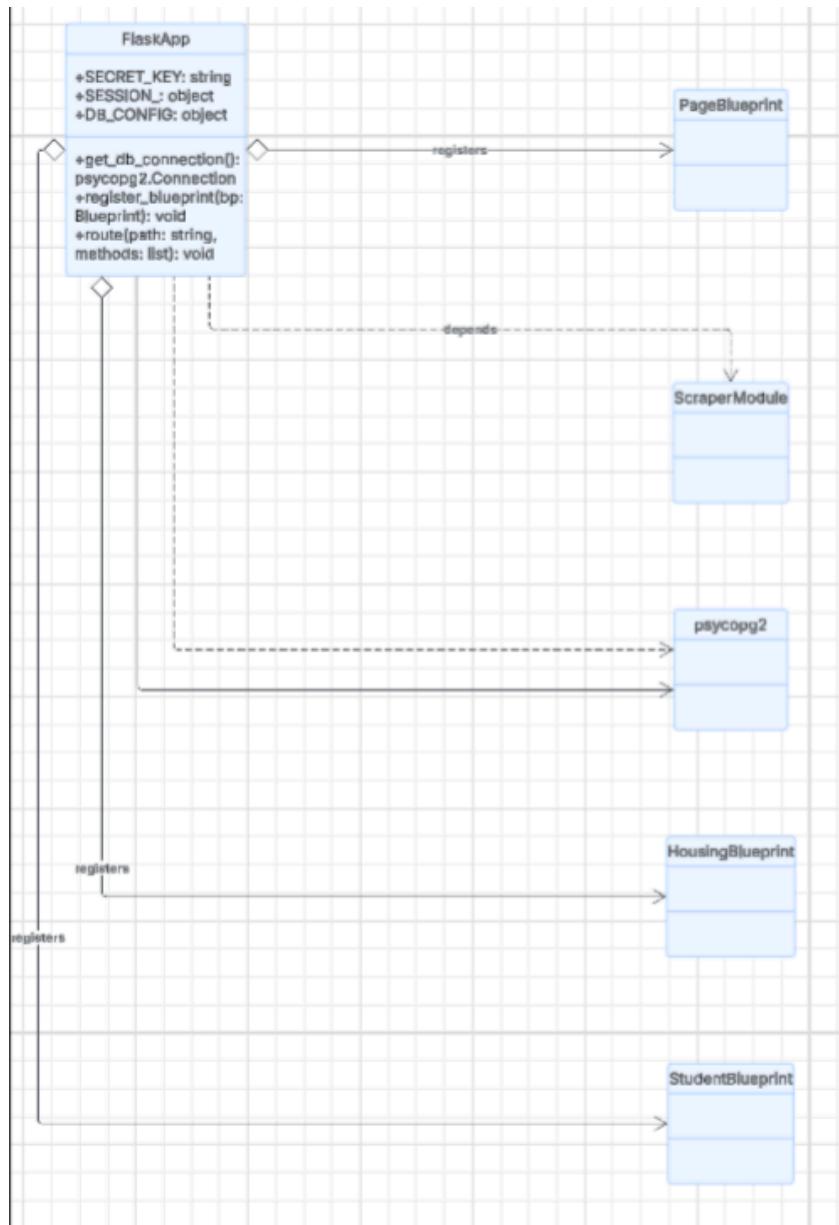
# Software Architecture Diagram

## a. Application Architecture Diagram

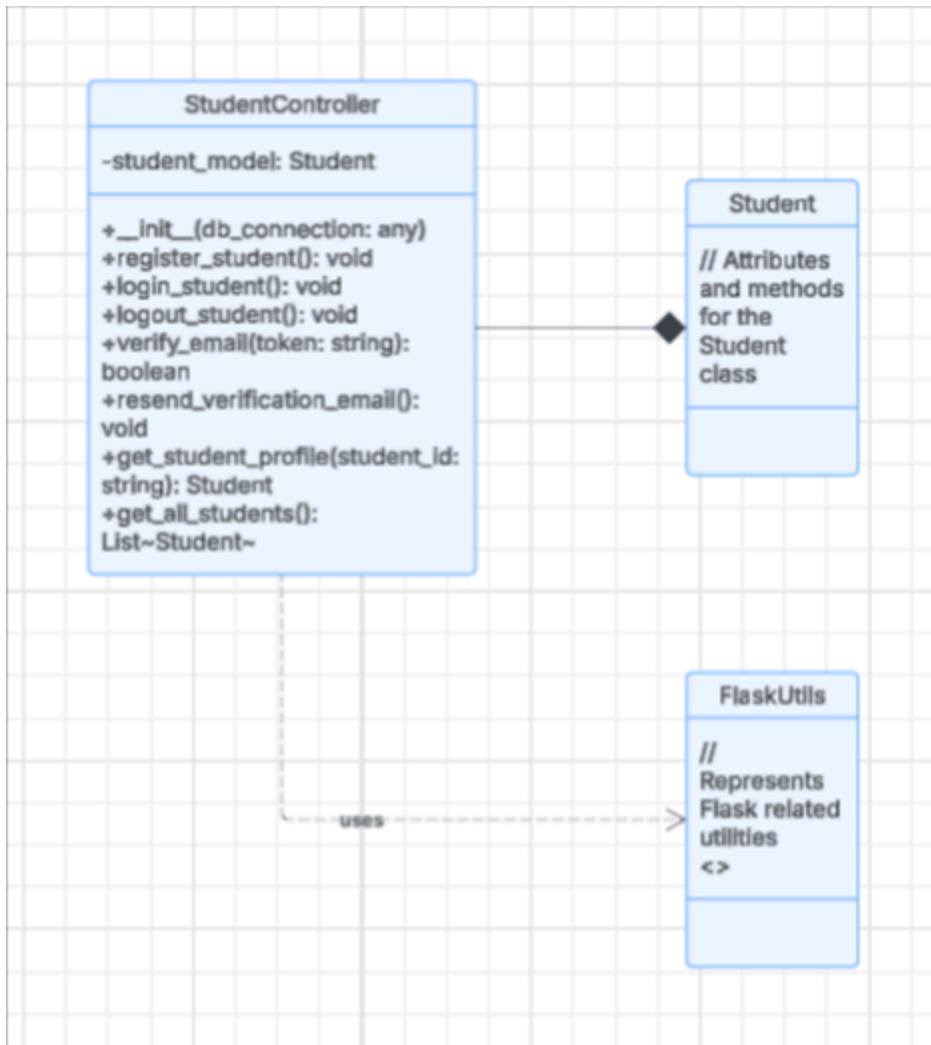


## b. Class Architecture Diagram

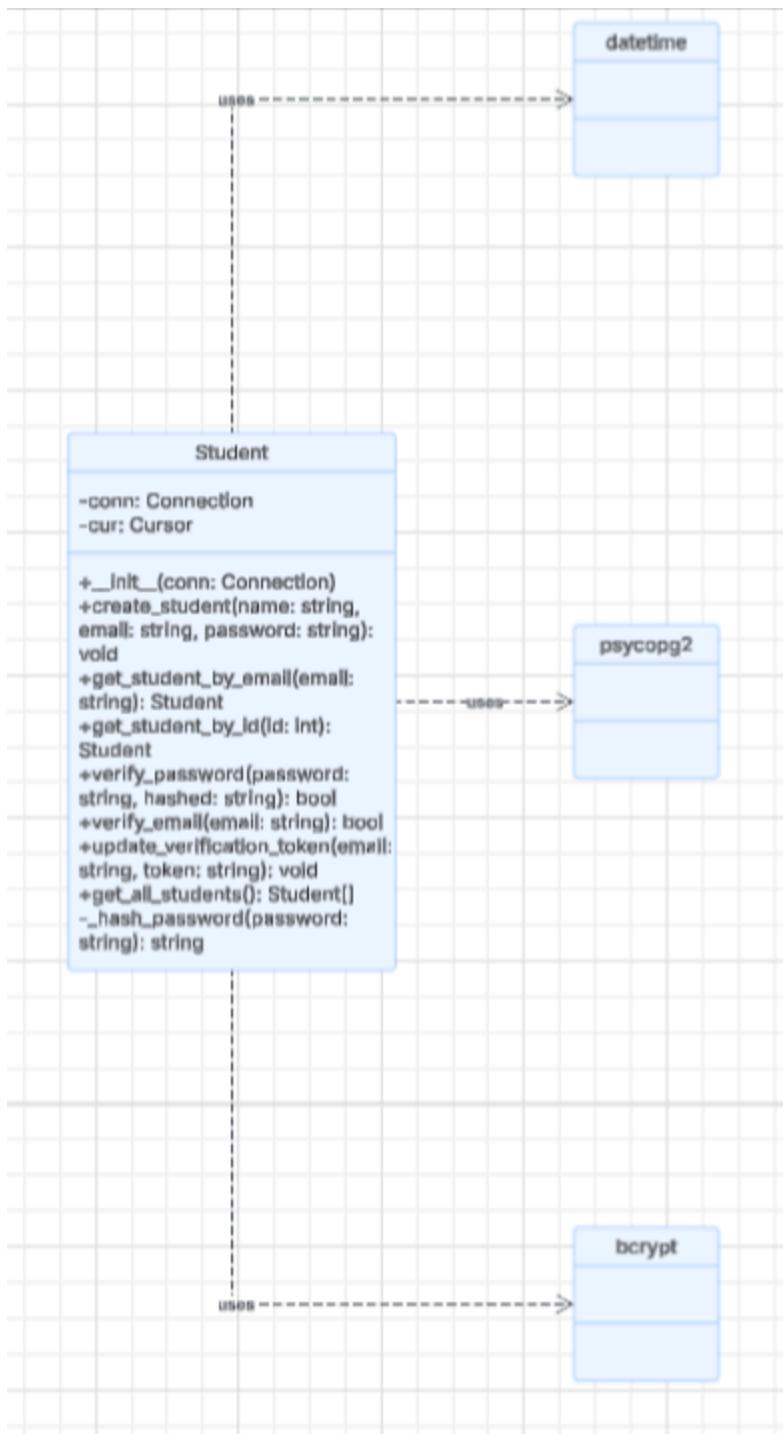
### Flask Application



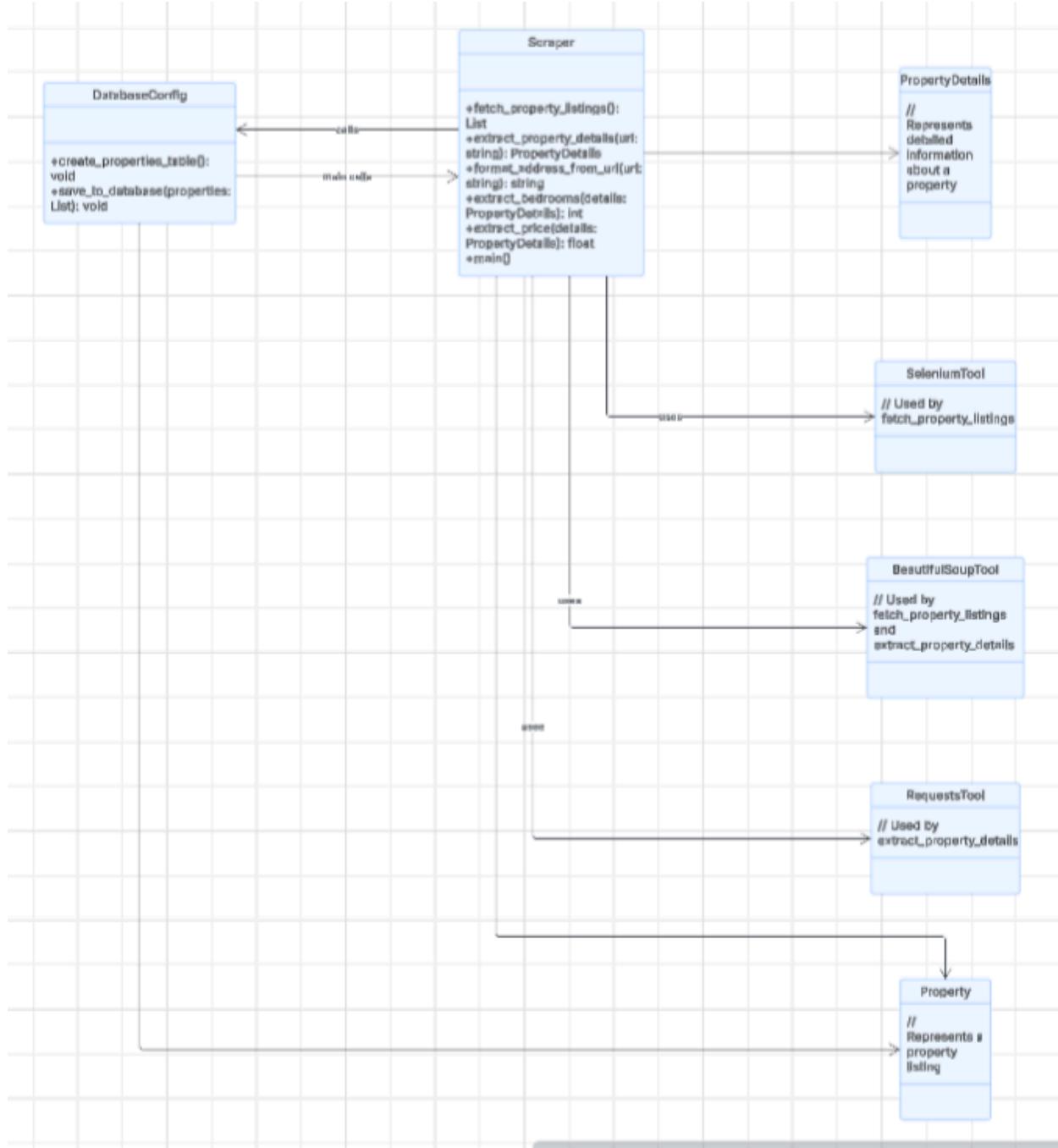
## Student Controller



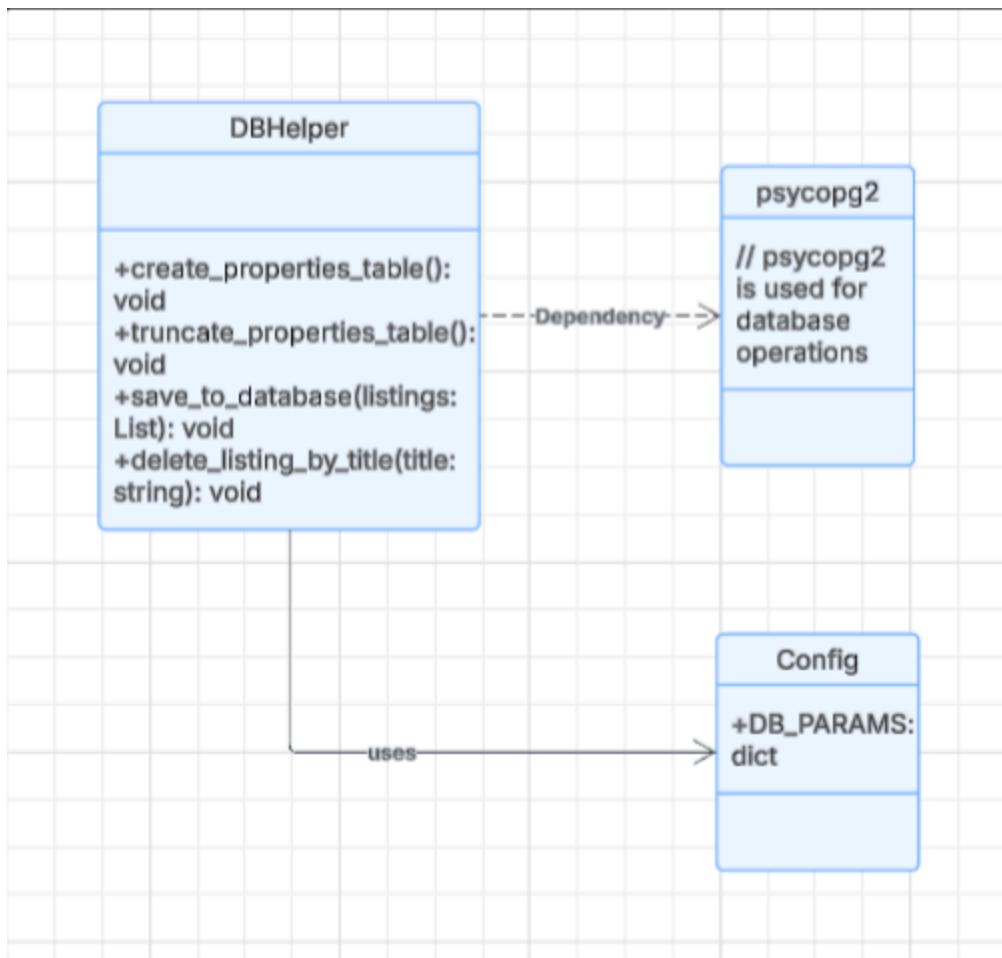
## Student Model



## Scraper



## Listings Database



# Testing

- a) Methodology:
  - i) Manually Test:
    - Check all main parts (Database and Web Server) to see how they work, to find a way to link both components with the arrangements of Docker.
- b) Frameworks/Models:
  - Flask
  - PostgreSQL
  - Psycopg2
  - Docker
  - Requests
- c) How to run Tests:
  - Current Run Test:
    - Start building the application:
      - With: "docker-compose build"
      - And then, "docker-compose up -d"
    - Continue with accessing:
      - <http://localhost:8000/>
    - If you want to visit the database, it can be accessed through:
      - <http://localhost:8080/>
    - Stop running:
      - With: "docker-compose down"
  - Previous Tests:
    - Our first test started with Docker (build & compose), allowing to add information about properties in the database in PostgreSQL (Adminer), but separated by the web server used to run as "python web\_server.py", and going to <http://localhost:5000/>. In other words, all modules were run independently of each other.
- d) List of tests per class/module:

This section outlines the unit tests designed for the core backend components of the Binghamton Housing [Portal.src/scrapers.py](#)

File: src/scrapers/utils.py

- test\_format\_address\_from\_url  
Method: format\_address\_from\_url(url\_path)

Description: Tests formatting URL slugs into readable addresses.

- `test_extract_bedrooms`  
Method: `extract_bedrooms(title)`  
Description: Tests extracting bedroom counts from property titles using various patterns.
- `test_extract_price`  
Method: `extract_price(text_or_element)`  
Description: Tests extracting price strings from text or HTML, including handling missing prices.
- `test_extract_property_details`  
Method: `extract_property_details(url)`  
Description: Mocks HTTP requests. Tests detailed extraction (title, price, amenities, description, bedrooms) and amenity standardization.

File: `src/scrapers/scrapers.py`

- `test_fetch_property_listings`  
Method: `fetch_property_listings()`  
Description: Mocks Selenium/HTTP requests. Tests main scraping workflow, URL processing, data extraction, and error handling.

File: `src/config/db.py`

- `test_create_properties_table`  
Method: `create_properties_table()`  
Description: Mocks DB connection. Verifies CREATE TABLE IF NOT EXISTS properties SQL execution.
- `test_save_to_database`  
Method: `save_to_database(listings)`  
Description: Mocks DB connection. Tests saving properties using INSERT ... ON CONFLICT ... DO UPDATE. Verifies commit and rollback.

File: src/init\_db.py

- test\_init\_db  
Method: init\_db()  
Description: Mocks DB connection. Verifies CREATE TABLE IF NOT EXISTS for core tables such as properties, students, and saved\_listings.

File: src/server.py (Flask app and blueprints)

- test\_db\_connection\_setup  
Method: get\_db\_connection()  
Description: Mocks DB connection. Verifies connection using proper config.
- test\_get\_properties\_api  
Route: /properties  
Description: Uses Flask test client and mocked DB. Tests fetching all properties and filtering by bedroom count. Verifies SQL and JSON response.
- test\_get\_property\_api  
Route: /properties/<int:property\_id>  
Description: Tests fetching a specific property (found and not found). Verifies SQL, JSON response, and 404 handling.
- test\_refresh\_data\_api  
Route: /refresh  
Description: Mocks run\_scraper. Tests triggering the scraper and verifies success/error responses.

File: src/server\_ui/students/controllers/student\_controller.py (Class: StudentController)

- test\_register\_student  
Method: register\_student()  
Description: Mocks request, model, and email utilities. Tests success, Binghamton email validation, duplicate check, and missing fields. Verifies 201, 400, and 409 responses.
- test\_login\_student

Method: login\_student()

Description: Mocks request and model. Tests successful login, invalid credentials, and unverified users. Verifies session and 200, 401, and 403 responses.

- test\_logout\_student

Method: logout\_student()

Description: Mocks session. Verifies clearing session and success response.

- test\_verify\_email

Method: verify\_email(token)

Description: Mocks model. Tests both success and failure token verification. Verifies 200 and 400 responses.

- test\_resend\_verification\_email

Method: resend\_verification\_email()

Description: Mocks request, model, and email utilities. Tests resending for unverified users, already verified cases, and non-existent emails. Verifies token update and email sending.

- test\_get\_student\_profile

Method: get\_student\_profile(student\_id)

Description: Mocks session and model. Tests authorized profile access, unauthorized access, and user not found. Verifies 200, 401, and 404 responses.

File: src/server\_ui/students/models/student\_model.py (Class: Student)

- Tests:

- test\_create\_student
- test\_get\_student\_by\_email
- test\_get\_student\_by\_id
- test\_verify\_password

- test\_verify\_email
  - test\_update\_verification\_token
- Description: Mocks DB connection. Verifies SQL correctness for INSERT, SELECT, and UPDATE statements related to student management and authentication.

File: src/server\_ui/routes/housing.py

- test\_get\_listings\_api  
Route: /api/listings  
Description: Uses Flask test client and mocked DB. Tests listing retrieval with filters (bedrooms, price, distance) and sorting. Verifies SQL and Python filtering logic.
- test\_get\_property\_api  
Route: /api/properties/<int:property\_id>  
Description: Re-listed for clarity. Mocks DB and extract\_property\_details. Tests fetching and merging detailed property data. Verifies response format.
- test\_save\_listing\_api  
Route: /api/saved-listings/save  
Description: Mocks session and DB. Tests saving listings (successful, already saved, property not found, unauthorized). Verifies SQL and status codes.
- test\_unsave\_listing\_api  
Route: /api/saved-listings/unsave  
Description: Mocks session and DB. Tests unsaving listings (successful, unauthorized). Verifies SQL and responses.
- test\_check\_saved\_listing\_api  
Route: /api/saved-listings/check  
Description: Mocks session and DB. Tests checking if a listing is saved. Verifies boolean response and SQL execution.
- test\_get\_saved\_listings\_api  
Route: /api/saved-listings

Description: Mocks session and DB. Tests retrieving saved listings for the logged-in user. Verifies SQL JOIN and response structure. Tests unauthorized access.

e) Coverage Report:

- The implemented unit tests provide comprehensive coverage for the application's core backend functionality. This includes the web scraping logic responsible for fetching and parsing property data from external sources, ensuring accurate extraction of details like addresses, pricing, and bedroom counts. Database interactions are thoroughly tested, covering the creation and manipulation of essential tables (properties, students, saved\_listings), including operations like saving scraped data, managing student accounts (registration, password verification, email verification flows), and handling saved property listings for users. Furthermore, the tests validate the primary API endpoints exposed by the Flask server, verifying the correct handling of requests for retrieving property listings (both all and specific), triggering data refreshes, managing student authentication (login, logout, registration), and enabling users to save and retrieve their preferred listings. The controller and model layers are specifically targeted to ensure business logic, data validation, and database query execution behave as expected under various scenarios, including success paths and common error conditions.

## User Interface

### 1. Interface Version 1.0

**Description:** Once the user opens the website, he will have access to all tabs distributed in the order of what they are planning to look for. In this case, they will register a student, then in the other tab see a list of students looking for roommates and in the last tab see available properties.

**Languages: Python - Flask, HTML & HTMX, CSS**

- **Versions:**

- **Python 3.10.12**
- **Htmx 2.0**
- **CSS**
- **HTML**
- **FLASK library**



Main User interface, page that appears when the website is open - Draft



How the data will be presented when interacting with the tabs (Drafts/Concepts)

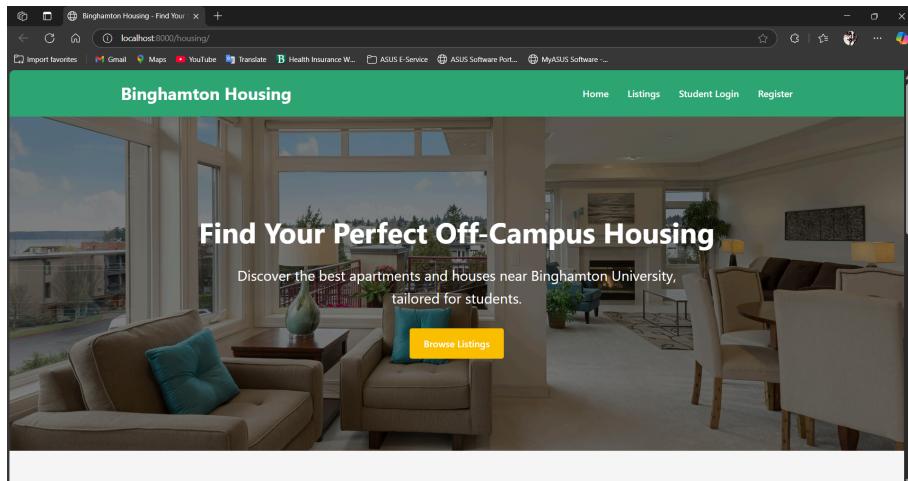
## 2. Interface Version 2.0 (Final Version)

**Description:** Final UI version. When the user opens the page they will be directed to the home page where they can browse listings, login, and register. Under the login window, Binghamton students can register using email, password, BU ID, major, and year. Under the listings, the user will be able to filter through different options like price range, beds, and distance to campus.

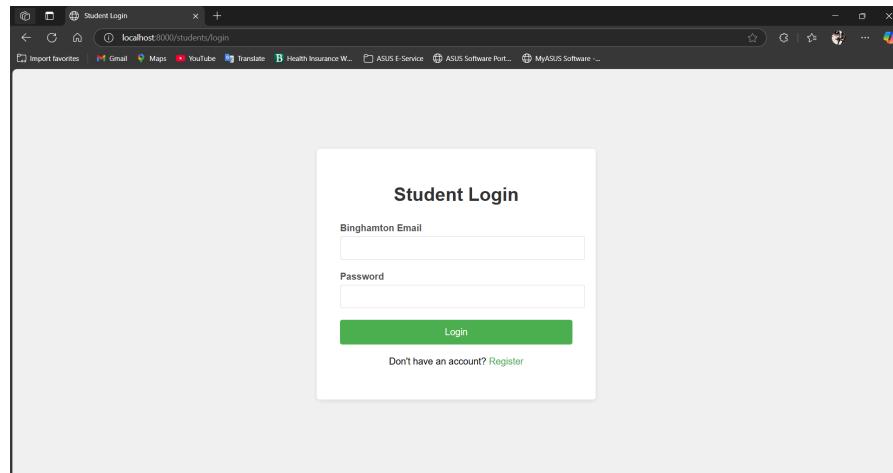
**Languages: Python, Flask, PostgreSQL, HTML, CSS, JavaScript, Selenium, BeautifulSoup**

- **Versions:**

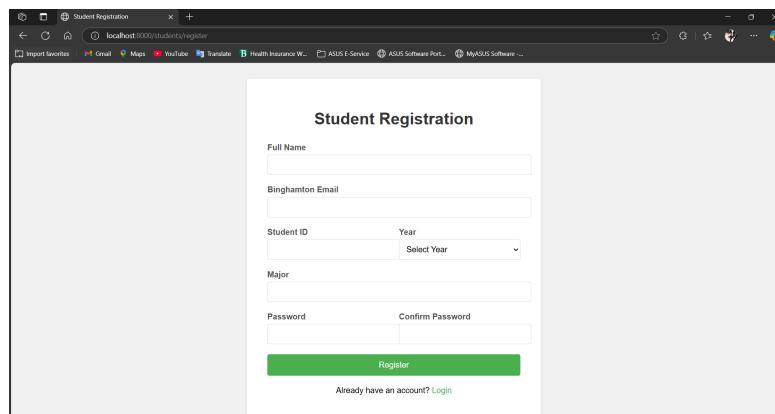
- Python 3.9
- Flask 2.0.1
- PostgreSQL 15
- HTML5
- CSS3
- Selenium 4.18.1
- BeautifulSoup4 4.10.0



*Home Page*



*Login Page*



*Student Registration*

A screenshot of a web page for "Binghamton Housing". The header has links for "Home", "Listings", "Student Login", and "Register". The main section is titled "Binghamton Housing Listings" and says "Showing all available properties". Below this are several filter options: "All", "1 Bedroom", "2 Bedrooms", "3 Bedrooms", and "4+ Bedrooms". A large "Filter Options" box contains fields for "Min Price (\$)", "Max Price (\$)", "Max Distance (miles)", and "Sort By" (set to "Price (Low to High)"). There is also a dropdown for "Show listings" set to "All listings". At the bottom of the filter box are buttons for "Reset Filters", "Apply Filters" (highlighted in yellow), and "View API Data".

*Listings Page*

# Project Status

```
1 ## Project Status
2
3 - Known Bugs
4   - API Request Timeout:
5     - description: API calls sometimes timeout, and there is no fallback, so the application gets an error
6     - nature: Performance
7     - severity: High-Severity
8     - priority: Low-Priority
9
10 - Critical Non-Implemented Features:
11   - Dice Roll:
12     - description: Virtual roll of any set of 4,6,8,10,12,20-dice with modifiers
13     - priority: high priority
14
15 - Phase 2 Features:
16   - Player Hud
17   - description: a HUD with quick access info for each player
```

## Known Bugs:

- Verification Error:
  - Since it is a local website the verification doesn't 100% work
    - Nature: Security
    - Severity: High Severity
    - Priority: Low Priority

## Critical Non-Implemented Features:

### Phase 2 Features:

- Move from local website to open website for others
- Better way to get data rather than web scraping
- A rating system so previous renters can let others know about their experience