🎓 Ethics Dashboard Student Side Developer Guide

April 2025

**Developers:**

* Keona Gagnier [Email](mailto:kohorstkeona@outlook.com?Subject=Ethics%20Dashboard%20Development)
* Fir Bolken
* Cody Jorgenson [cjorgenson39@gmail.com](mailto:cjorgenson39@gmail.com)

You can also find a very similar markdown file to this one in our Github in the “documentation” folder, it is a bit easier to read.

# Table of Contents

[**Table of Contents 2**](#_1c1j31dwfi48)

[**Project Description 5**](#_h3bneki9t2w9)

[**Install the Project 6**](#_1obzki6k5jiq)

[**Run the Project Development Environment 6**](#_6qfdoe1418mh)

[**How to populate DB with the DB dump 7**](#_jfteh81p33r9)

[**API Documentation 8**](#_kffbqal6m6ah)

[What API routes do we have? 8](#_tk59yr5tlews)

[Getting started guide 8](#_9t1bfdmnq91o)

[Api jwt token 9](#_foevqbskwv0q)

[Student 9](#_4nt2t4o4jehf)

[GET Get all students 9](#_m2zn3kierusi)

[PATCH Update consent 9](#_i3yzayw86s54)

[GET Get consent 9](#_ym4lr39d0bk)

[GET Get email 10](#_za4z9h9vkae)

[Authorization 10](#_n7ulj32h8446)

[POST Register student 10](#_ug3v0uyfgpwu)

[POST Login student 11](#_gjr853vz44fr)

[POST Validate token 11](#_t846w0sn1wij)

[POST Change student password 11](#_g6birvrzlxsh)

[POST Login professor 12](#_mh8gjgai4ft1)

[POST Register professor 12](#_5jhugdu9b0k0)

[POST Register TA 13](#_xj6awfac4ymh)

[POST Register as Guest 13](#_g0bxpn2mqwox)

[POST Change student email 13](#_ivm0qyboa2g5)

[DELETE Delete student 14](#_dqy1i4i0tcpn)

[Assignments 15](#_wv86mdq6e1c1)

[POST Save answers 15](#_3wka21l0n3vj)

[POST Submit answers 17](#_lpis0ujg44xt)

[GET Get answers 18](#_7hbb3nafip79)

[GET Is form submitted 19](#_h1v45667dg69)

[GET Get assignments for student 19](#_pbdbpip6ijyw)

[GET Get assignments for student class 19](#_sh08scetcit4)

[PATCH Set case study options 19](#_lo57cv9j1dcy)

[GET Is assignment submitted 20](#_wvq3zw9va771)

[Questions 20](#_t7ep9py2v4fe)

[GET Get questions 20](#_97z2ypi0lsm7)

[GET Get total questions 21](#_2h0xbwn475mw)

[Feedback 21](#_9jwi3pd5c08)

[GET Get feedback for form 21](#_8v4pulrtn2pg)

[Grade 21](#_mkahg3gvacex)

[GET Get grade 21](#_yvjlvonc2v0p)

[GET Get grades 21](#_6pxzps6y97m4)

[Enrollment 22](#_z84ybymobu55)

[GET Get enrollments 22](#_n4iov03vcfpe)

[POST Enroll student 22](#_shitumxoe0qf)

[Case Studies 22](#_h1hvi5ha559b)

[GET Get options 22](#_4iwjnmw7bw70)

[Form Description 23](#_xfvvnbiplmx7)

[GET Get description 23](#_i3k30xxaa6iq)

[Slider Questions 23](#_94u8kbybxtn4)

[GET Get Slider Questions 23](#_tny2tjjlk9ov)

[GET Get Total Slider Questions 23](#_1u641agvw0e9)

[Guest 24](#_ju4ohemp3ypd)

[POST Set up guest 24](#_q9dfec8o55qu)

[DELETE Delete all guests 24](#_ze6a54qh6olj)

[**Architecture 24**](#_s8ehk4o1wfyr)

[Model 24](#_c7bd25xlvfv0)

[View 25](#_9cryw2p0t9my)

[Controller 25](#_r1g3hbsiw8ob)

[**DB Diagram 26**](#_yo8vlbkgqo8)

[Tables Overview 27](#_xg6b1r83lake)

[1. Submissions 27](#_o0dc3bytj9v7)

[2. Grades 27](#_mhowa8cz19ux)

[3. Students 27](#_u67eyxe5mm3h)

[4. Assignments 27](#_qv7r13te3cx1)

[5. Answers 27](#_4vdu92y6a5qe)

[6. Enrollments 27](#_y9ucopiadigk)

[7. Classes 27](#_rk79qzb7bj0j)

[8. Forms 27](#_3ilwrhmuq3h2)

[9. Professors 28](#_dac2zbfhzi4j)

[10. TAs (Teaching Assistants) 28](#_ev10pud11c7x)

[11. Case Studies 28](#_aftlciojfp3u)

[12. Case Study Options 28](#_thsddwd3msh3)

[13. Slider Questions 28](#_qy55nwn4ywzk)

[14. Dynamic Questions 28](#_jq69jlyh6ot1)

[15. Form Descriptions 28](#_1lg1c2phx7lr)

[16. Feedbacks 28](#_wguxyhafja3x)

[Section Key Values 28](#_5yt8wun5d9yt)

[**Sequence Diagram 31**](#_x8u05jmqzpz8)

[**How to add a model 32**](#_8kdqtz1u41r)

[**How to add a controller/API route 34**](#_gzo6a2rzki9i)

[1. Navigate to the Controllers Directory 34](#_tmly8lyv06ts)

[2. Create a New Controller File 34](#_mkte3hfz7xx9)

[3. Import Necessary Modules 34](#_rhvw92x0yfpr)

[4. Register Your Controller with a Blueprint 34](#_h1g6dfhye1xe)

[5. Define API Routes 34](#_mazmfe7eiqm9)

[6. Register the Controller in app.py 35](#_y06j1ra1rwvi)

[7. Restart Your Flask Server 35](#_2fu2f1wfymtc)

[Summary of Steps 36](#_j0g6e8qtc4q8)

[**How to add a view 37**](#_wqbdhqw1apob)

[1. Creating a New Page 37](#_b7kzdt73xq0m)

[Example page.tsx: 37](#_u40ezr5ti79l)

[2. Adding UI Components 37](#_g3rlhw1fduz3)

[Example Component (/frontend/app/ui/profile/ProfileForm.tsx): 38](#_b39k29pqur05)

[3. Reusable Components & Utility Functions 39](#_2euyo5k2xyka)

[Example Utility Function (fetchUser.ts): 39](#_ydeqpwlxn0uk)

[4. Testing Your Page 39](#_tzfp295vka4f)

[5. Summary of Steps 39](#_ril06ckk465m)

[**How to create and run tests 41**](#_77w2e7eoubrm)

[1. Navigate to the Test Directory 41](#_kxybuhn7og3v)

[2. Create a New Test File 41](#_gtvc6c7sflf6)

[3. Import Necessary Modules 41](#_fecfgfhoq3ul)

[4. Write Your Test Function 42](#_fihtsjya5dls)

[5. Run Your Tests 42](#_n0kvsqwxinfh)

[**Using the Server and Deployment 43**](#_2wdnyksx9e63)

[Deployment on the Server 43](#_57kb2su16ahc)

[1. Connect to server via terminal window 43](#_nisthe2ha6ll)

[2. Navigate to the ethicdDashboard directory 43](#_b94eej2qnx6i)

[3. Build the Docker containers 43](#_1wgojjkfvzvp)

[4. Run the Containers 43](#_xqnkzvse1mvo)

[Production Testing Environment 43](#_rw5jxfhpbt07)

[1. Navigate to ethicdDashboard 43](#_3l3if8o9hfu2)

# 

# Project Description

The **Ethics Dashboard** is a responsive web application designed to facilitate ethical decision-making for students, instructors, and teaching assistants. It provides a structured framework to analyze ethical dilemmas through visual and interactive tools.

**Key Features:**

1. **Ethics Dashboard**: A central hub presenting a visual summary of ethical analyses and inputs in real-time.
2. **Ethical Analysis Framework**: Guides users through a structured Seven-Step Framework for decision-making.
3. **Global Ethical Lenses**: Includes both Western and non-Western ethical theories, emphasizing diverse perspectives.
4. **Self-Reflection Tools**: Encourages evaluation using universal ethical principles to build personal and global ethical awareness.
5. **Instructor Feedback**: Allows instructors to assess student progress, provide feedback, and assign grades.
6. **User-Friendly Interface**: Features guided prompts, step-by-step inputs, and a clean navigation system for seamless usability.
7. **Secure Accounts**: Supports user authentication and integration with learning platforms (e.g., Moodle).

**Scope:**

* **Student Workflow**: Input ethical dilemmas, analyze using various frameworks, and generate comprehensive reports.
* **Instructor Workflow**: Evaluate student work through dashboards and detailed reports.
* **Backend Logic**: Stores user data, executes framework logic, and ensures secure data handling.

**Project Timeline:**

* **Dec 1**: Ethics Dashboard prototype with partial functionality.
* **Mar 1**: Fully functional application ready for testing.
* **Apr 1**: Final revisions based on feedback.
* **End of Term**: Launch for use in applied ethics courses.

By combining interactive learning with comprehensive ethical analysis tools, the **Ethics Dashboard** aims to enhance ethical education through an engaging and globally inclusive approach.

# Install the Project

You must have the following installed as prerequisites:

* Docker
* Python

You can download the project files from Cody Jorgensons GitHub repo, titled ethicdDashboard and the professor side of the project titled Ethics-Dashboard-Prof

Clone the repository locally on your device.

The repo is private at the time of writing this, you may need to request access.

Alternatively the files are all stored on the kamatera server, you can copy the directories, delete the .git directory and re-initialize the github repo under your ownership.

If the link is broken, the GitHub username is c-jorg. You can also reachCody by his email at cjorgenson39@gmail.com, and he can help you get access.

I recommend installing the project in your root C:/ folder for easy access.

# Run the Project Development Environment

Navigate to the ethicdDashboardt directory via command line and run the following commands to build and start the Docker containers:

$ docker-compose -f compose.dev.yml up –build

then navigate to localhost in your browser to see the log in page.

The compose.dev.yml file builds the project with development variables, nginx container and without certbot, instead using a self signed ssl certificate.

This will start the next.js development environment, and you should now be able to log in on localhost

Since you just created your database, it will be empty, so you will need to create an account first.

* Remember that students have to be invited to create an account, so you have to manually put in the students email into the database Students table (or use the Professor side to invite a student).

As for the rest of your database, we have a database dump file full of dummy data that you can use to populate your database if you want. That will be explained in another section.

To access the postgres database, execute the following command in the ethicdDashboard directory:

$ docker exec -it db psql -U postgres

# How to populate DB with the DB dump

The file in the root of the project titled “dml\_dump.sql” contains dummy data.

To populate the database using this file, follow these steps:

1. Copy dml\_dump.sql from Host to the Database Container

Run this command on your host system to copy the SQL dump file into the container:

docker cp C:/ethicdDashboard/dml\_dump.sql db:/tmp/dml\_dump.sql

1. Access the Database Container Terminal

Start an interactive shell session inside the database container:

docker exec -it db bash

1. Import the DML Dump into PostgreSQL

Once inside the container, use the following command to populate the database:

psql -U postgres -d postgres -f /tmp/dml\_dump.sql

1. Exit the Container

After the import is complete, exit the container:

exit

# 

# API Documentation

Link to [API documentation](https://restless-robot-70850.postman.co/workspace/a6c42954-0266-45ad-8e38-7752bda16afa/collection/39425129-5764d51a-c83a-406c-89e0-abb79ce4e8f8) on postman

You may need to request access. Otherwise, view [the ethicdDashboard GitHub](https://github.com/Okanagan-College-COSC470-Fall-2024/the-project-team-3) Repo and download the API documentation file from the “documentation” folder in the root of the project and import it to [Postman](https://www.postman.com/).

If you are struggling with Postman, you can also just look at the Controllers file in the backend and look at the routes in there. It is less straightforward but it will give you the same information.

## What API routes do we have?

Assignments routes

Authorization routes

Students routes

Question routes

Feedback routes

Grade routes

Enrollment routes

Case Study routes

Form Description routes

Slider Question routes

Guest routes

## Getting started guide

To start using the APIs, you need to know:

* The API returns request responses in JSON format. When an API request returns an error, it is sent in the JSON response as an error key. Some APIs return message keys rather than error keys which will be highlighted in the individual route details.
* The frontend uses axios to make requests to the API.

## Api jwt token

To make an api request, you must have the proper authorization header using a valid jwt token.

* frontend/app/utils/api-auth.ts will add the authorization header using the jwt token in local storage
* Run backend/utils/dev\_token.py to generate a jwt token to use in manual api requests and tests

## Student

The following endpoints let you manage and retrieve information about student users.

﻿

### **GET Get all students**

http://localhost:4000/api/flask/students

Returns a JSON list full of all student data (passwords are hashed).

### **PATCH Update consent**

http://localhost:4000/api/flask/student/consent

Update the consented column in the database to true or false

﻿Body JSON

json

{

"consent": "no",

"id": 1

}

### **GET Get consent**

http://localhost:4000/api/flask/student/consent?user\_id=1

Returns a JSON with a true/false value of the users data consent status

﻿

### **GET Get email**

http://localhost:4000/api/flask/student/email?student\_id=1

Returns a JSON containing a string of the students email

## Authorization

The following endpoints perform authentication, user registration and session handling.

﻿

### **POST Register student**

http://localhost:4000/api/flask/auth/register-student

Student users need to be invited to create an account, so in order to test this API you must first add the email for the student you want to register into the backend database in the students table.

A student is "invited" if their email is in the students table. However, if a student's email is in the database AND they already have a password/name set up, their account has already been created.

Body JSON

json

{

"name":"StudentName",

"email":"test@mail.com",

"password":"Password1",

"consent":"false"

}

### **POST Login student**

http://localhost:4000/api/flask/auth/login-student

This will return a JSON object containing the token, with the key "token"

Also returns "id" for student ID, and "name" for the students name.

Body JSON

json

{

"email":"uniquetest@mail.com",

"password":"Password1"

}

### **POST Validate token**

http://localhost:4000/api/flask/auth/validate-token

Returns a JSON with a success message and the users email decoded from the token

Body JSON

json

{

"token":"insert token here"

}

### **POST Change student password**

http://localhost:4000/api/flask/auth/change-student-password?oldPassword=old&newPassword=new&id=1

Changes the students password in the database, and returns a JSON success message if completed

﻿Body JSON

json

{

"id":"1",

"oldPassword":"Password1",

"newPassword":"Password1"

}

### **POST Login professor**

http://localhost:4000/api/flask/auth/login-professor

﻿Checks the credentials against the database, and if successful returns a JSON success message, token, id, and name associated with the credentials (This route is unused in this version)

Body JSON

json

{

"email":"test@test.com",

"password":"Password1"

}

### **POST Register professor**

http://localhost:4000/api/flask/auth/register-professor

﻿Creates credentials for a new professor in the database, provided the email matches an existing row. Returns JSON with id, name, email, and password(hashed)

Body JSON

json

{

"name":"ProfessorName",

"email":"uniqueprof@mail.com",

"password":"Password1"

}

### **POST Register TA**

http://localhost:4000/api/flask/auth/register-ta

﻿﻿Creates credentials for a new TA in the database, provided the email matches an existing row. Returns JSON with id, name, email, and password(hashed)

Body JSON

json

{

"name":"TAName",

"email":"uniqueta@mail.com",

"password":"Password1"

}

### **POST Register as Guest**

http://localhost:4000/api/flask/auth/register-guest

Creates a new guest user in the students table, using a randomly generated email and password. Populates the guest user with a class and three assignments of varying completion. Automatically logs in with this new user. Return JSON with, success message, token, id, and name.

﻿

### **POST Change student email**

http://localhost:4000/api/flask/auth/change-student-email

Updates the students email in the database, needs correct credentials. Returns JSON with success message containing the new email

﻿

Body JSON

json

{

"student\_id":"2",

"new\_email":"stu@email.com",

"password":"Password1"

}

### **DELETE Delete student**

http://localhost:4000/api/flask/auth/student

Deletes the student and any related data completely from all database tables if credentials match and they have not consented to the data collection agreement and logs them out, returns JSON with success message and data consent. If they have consented to the data collection then their email and name will be deleted and the deleted column with be changed to true, returns JSON with success message and data consent

﻿

Body JSON

json

{

"student\_id":"17",

"password":"Password1"

}

## Assignments

These endpoints support assignment data saving and retrieval. Also support retrieving assignments for a particular user.

### **POST Save answers**

http://localhost:4000/api/flask/assignment/save-form

Saves the form answers into the db. First it deletes all the old answers associated with the form then new answers are saved. Each answer from the “answers” list in the request JSON will have an entry into the answers table. Returns a JSON success message.

Body JSON

json

{

"student\_id": "1",

"assignment\_id": "1",

"case\_study\_id": "1",

"form\_name": "dilemma",

"answers": {

"dilemma-0": "conflict\_of\_interest",

"dilemma-1": "false",

"dilemma-2": "false",

"state-the-problem": "test",

"gather-facts-1": "i dont know!\n",

"gather-facts-2": "somewhere really cool",

"gather-facts-3": "like yesterday?",

"stakeholder-name-0": "?!",

"stakeholder-directly-0": "directly",

"stakeholder-indirectly-0": "false",

"stakeholder-name-1": "test!",

"stakeholder-directly-1": "false",

"stakeholder-indirectly-1": "indirectly",

"stakeholder-name-2": "test!",

"stakeholder-directly-2": "directly",

"stakeholder-indirectly-2": "false",

"stakeholder-name-3": "test!",

"stakeholder-directly-3": "false",

"stakeholder-indirectly-3": "indirectly",

"stakeholder-name-4": "test!",

"stakeholder-directly-4": "directly",

"stakeholder-indirectly-4": "false",

"stakeholder-name-5": "test!",

"stakeholder-directly-5": "false",

"stakeholder-indirectly-5": "indirectly",

"stakeholder-name-6": "test!",

"stakeholder-directly-6": "directly",

"stakeholder-indirectly-6": "false",

"option-title-0": "my option 1",

"option-description-0": "my description 1",

"option-title-1": "my cooler option 2",

"option-description-1": "my cooler description 2",

"option-title-2": "coolness option",

"option-description-2": "coolness description",

"option-title-3": "cute option",

"option-description-3": "cute description",

"option-title-4": "silly option :P",

"option-description-4": "silly description",

"harm-yes-0": "yes",

"harm-no-0": "false",

"publicity-yes-0": "yes",

"publicity-no-0": "false",

"reversible-yes-0": "yes",

"reversible-no-0": "false",

"harm-yes-1": "false",

"harm-no-1": "no",

"publicity-yes-1": "false",

"publicity-no-1": "no",

"reversible-yes-1": "false",

"reversible-no-1": "no",

"harm-yes-2": "yes",

"harm-no-2": "false",

"publicity-yes-2": "yes",

"publicity-no-2": "false",

"reversible-yes-2": "yes",

"reversible-no-2": "false",

"harm-yes-3": "false",

"harm-no-3": "no",

"publicity-yes-3": "false",

"publicity-no-3": "no",

"reversible-yes-3": "false",

"reversible-no-3": "no",

"harm-yes-4": "yes",

"harm-no-4": "false",

"publicity-yes-4": "yes",

"publicity-no-4": "false",

"reversible-yes-4": "yes",

"reversible-no-4": "false",

"tentative-choice-0": "false",

"tentative-choice-1": "false",

"tentative-choice-2": "false",

"tentative-choice-3": "false",

"tentative-choice-4": "silly option :P",

"num\_stakeholders": "7"

}

}

### **POST Submit answers**

http://localhost:4000/api/flask/assignment/submit-form

Saves the form answers into the db. First it deletes all the old answers associated with the form then new answers are saved. Each answer from the “answers” list in the request JSON will have an entry into the answers table. Creates a new entry into the submissions table, for the assignment\_id, student\_id, and the form\_id. Returns a JSON success message..

Body JSON

json

{

"student\_id": "1",

"assignment\_id": "2",

"case\_study\_id": "2",

"form\_name": "dilemma",

"answers": {

"dilemma-0": "false",

"dilemma-1": "false",

"dilemma-2": "false",

"state-the-problem": "explosions!",

"gather-facts-1": "it did!",

"gather-facts-2": "my house!",

"gather-facts-3": "fuck you!",

"stakeholder-name-0": "a",

"stakeholder-directly-0": "false",

"stakeholder-indirectly-0": "false",

"stakeholder-name-1": "b",

"stakeholder-directly-1": "false",

"stakeholder-indirectly-1": "false",

"stakeholder-name-2": "c",

"stakeholder-directly-2": "false",

"stakeholder-indirectly-2": "false",

"stakeholder-name-3": "d",

"stakeholder-directly-3": "false",

"stakeholder-indirectly-3": "false",

"stakeholder-name-4": "e",

"stakeholder-directly-4": "false",

"stakeholder-indirectly-4": "false",

"stakeholder-name-5": "",

"stakeholder-directly-5": "false",

"stakeholder-indirectly-5": "false",

"stakeholder-name-6": "",

"stakeholder-directly-6": "false",

"stakeholder-indirectly-6": "false",

"option-title-0": "",

"option-description-0": "",

"option-title-1": "",

"option-description-1": "",

"option-title-2": "",

"option-description-2": "",

"option-title-3": "",

"option-description-3": "",

"option-title-4": "",

"option-description-4": "",

"harm-yes-0": "false",

"harm-no-0": "false",

"publicity-yes-0": "false",

"publicity-no-0": "false",

"reversible-yes-0": "false",

"reversible-no-0": "false",

"harm-yes-1": "false",

"harm-no-1": "false",

"publicity-yes-1": "false",

"publicity-no-1": "false",

"reversible-yes-1": "false",

"reversible-no-1": "false",

"harm-yes-2": "false",

"harm-no-2": "false",

"publicity-yes-2": "false",

"publicity-no-2": "false",

"reversible-yes-2": "false",

"reversible-no-2": "false",

"harm-yes-3": "false",

"harm-no-3": "false",

"publicity-yes-3": "false",

"publicity-no-3": "false",

"reversible-yes-3": "false",

"reversible-no-3": "false",

"harm-yes-4": "false",

"harm-no-4": "false",

"publicity-yes-4": "false",

"publicity-no-4": "false",

"reversible-yes-4": "false",

"reversible-no-4": "false",

"tentative-choice-0": "false",

"tentative-choice-1": "false",

"tentative-choice-2": "false",

"tentative-choice-3": "false",

"tentative-choice-4": "false",

"num\_stakeholders": "7"

}

}

### **GET Get answers**

http://localhost:4000/api/flask/assignment/get-answers?user\_id=1&form\_name=dilemma&assignment\_id=1

Returns a JSON with a success message and a list of all the answers associated with the student\_id, assignment\_id and the form\_id.

### 

### **GET Is form submitted**

http://localhost:4000/api/flask/assignment/is-form-submitted?form\_name=dilemma&assignment\_id=1&student\_id=1

Queries database to check if a submission exists for this form\_id, assignment\_id and student\_id. Returns JSON with value true or false.

### **GET Get assignments for student**

http://localhost:4000/api/flask/assignments?user\_id=1

Returns a JSON array with all assignment details for the student.

### **GET Get assignments for student class**

http://localhost:4000/api/flask/assignments?user\_id=1&class\_id=1

Returns a JSON array with assignment details for a specific class for the student.

﻿

### **PATCH Set case study options**

http://localhost:4000/api/flask/assignment/set-case-study-option

Sets the case study option for the assignment. Returns JSON success message

﻿

Body JSON

json

{

"assignment\_id": 1,

"case\_study\_option": 1

}

### **GET Is assignment submitted**

http://localhost:4000/api/flask/assignment/submitted?assignment\_id=1

Checks the submissions table to see if there is a submission for each form. Returns JSON message true/false

## Questions

API routes related to the custom/dynamic questions that professors can add to case studies

﻿

### **GET Get questions**

http://localhost:4000/api/flask/questions?form\_name=life-path&case\_study\_id=1

Returns a JSON list of all the dynamic question text associated with the form and case study

### **GET Get total questions**

http://localhost:4000/api/flask/questions/total?form\_name=generations-form&case\_study\_id=1

Returns a JSON with the total amount of dynamic questions associated with that form and case study

## Feedback

API routes related to professor comments/feedback that show up on the forms after they have been submitted

﻿

### **GET Get feedback for form**

http://localhost:4000/api/flask/feedback?form\_name=life-path&assignment\_id=1

Returns a JSON list of dictionaries containing the key and value of the feedback

## Grade

API routes related to student grades

﻿

### **GET Get grade**

http://localhost:4000/api/flask/grade?form\_group=Consequences&assignment\_id=1

Returns JSON with the grade of the student for that form and assignment

### **GET Get grades**

http://localhost:4000/api/flask/grades?assignment\_id=1

Returns JSON list with all of the grades for each form for that assignment

## Enrollment

API routes used to get information about enrollments.

﻿

### **GET Get enrollments**

http://localhost:4000/api/flask/enrollments?student\_id=1

Returns JSON list of the students enrollments, class\_id, class\_name, professor

### **POST Enroll student**

http://localhost:4000/api/flask/enrollment

Adds an entry into the enrollment table. Returns JSON success message, and class name

Body JSON

json

{

"student\_id":"1",

"class\_id":"1"

}

## Case Studies

API routes used to get information about case studies.

﻿

### **GET Get options**

http://localhost:4000/api/flask/case-study/options?case\_study\_id=1

Returns JSON list of all the case\_study\_options associated with that case\_study\_id

## Form Description

API routes used to manage form descriptions.

Form descriptions are the introductory blurbs of text that exist before some forms, professors can edit these descriptions and add links to them.

﻿

### **GET Get description**

http://localhost:4000/api/flask/form-description?assignment\_id=1&form\_name=cons-util-bentham

Returns JSON of the form description associated with that assignment\_id and form\_name

## Slider Questions

API routes used for managing slider questions.

These refer to the types of questions which use sliders/range inputs. They consist of the question, and the labels for either end of the slider.

﻿

### **GET Get Slider Questions**

http://localhost:4000/api/flask/slider-questions?case\_study\_id=1&form\_name=generations-form

Returns a JSON list of all the slider questions associated with that case\_study\_id and form name

### **GET Get Total Slider Questions**

http://localhost:4000/api/flask/slider-questions/total?case\_study\_id=1&form\_name=generations-form

Returns a JSON with the total number of sliders questions on that form for that case\_study\_id

## Guest

API routes for handling guests.

﻿

### **POST Set up guest**

http://localhost:4000/api/flask/guest/setup

﻿Adds the proper entries into the database to allow guest mode to work. Only needed when setting up a new database. Returns JSON success message

### **DELETE Delete all guests**

http://localhost:4000/api/flask/guest/delete-guests

Deletes all the guest users and all the data associated with them in the database. Returns JSON success message

# Architecture

MVC - Model View Architecture

The backend folder holds the Model and Controllers while the frontend folder holds the Views.

## Model

We are using SQLAlchemy in our Flask app as our ORM (Object Relational Mapping).

We have created classes for each entity in our database. All models are in the Model file in the backend folder.

## View

We are using Next.js for our frontend. Each file titled “page.tsx” is a view. All views are in the /frontend/app folder.

## Controller

We are using API calls (routes) to handle information flow from the database to the frontend, all of our controllers are in the Controller file in the backend folder.

# 

# DB Diagram

[Link to view diagram online](https://dbdiagram.io/d/EthicsDashboard-67b7a246263d6cf9a0eb57cc)



## **Tables Overview**

### **1. Submissions**

* Holds information for a submitted assignment.

### **2. Grades**

* Holds the grades for an assignment.
* **Form group** contains the following possible values:
  + Seven Step Method
  + Consequences
  + Role
  + Past Actions
  + Reason
  + Care Ethics
  + Intersectionality
  + Seven Generations
  + Virtue Ethics
  + Life Path
  + Universal Principles
* *Note: Creating a form\_group table might be a good improvement to the database.*

### **3. Students**

* Stores student information.

### **4. Assignments**

* Represents an assigned case study, associated with a student.

### **5. Answers**

* Stores the answers submitted for an assignment.

### **6. Enrollments**

* Captures the relationship between classes and students.

### **7. Classes**

* Stores information about classes.

### **8. Forms**

* Holds form names and their corresponding IDs.

### **9. Professors**

* Stores professor information.

### **10. TAs (Teaching Assistants)**

* Stores TA information.

### **11. Case Studies**

* Holds templates for case studies.
* When a professor creates a case study with custom questions, it is stored here.
* Assignments pull particular case study templates from this table.

### **12. Case Study Options**

* Each case study has several options for the "Case" the student can analyze.
* These options are stored here.

### **13. Slider Questions**

* Stores custom slider-type questions created by professors.

### **14. Dynamic Questions**

* Stores custom text input questions created by professors.

### **15. Form Descriptions**

* Some forms have an introductory paragraph.
* Professors can edit and add links.
* This content is pulled from this table.

### **16. Feedbacks**

* Holds professor comments associated with a particular assignment and form.
* Has a section\_key column which represents which section of the form the feedback is for.

#### **Section Key Values**

* section\_key represents which part of a form the comment is for.
* Valid values for section\_key include:

Seven Step Method

* select-dilemma - Comment box for selecting the dilemma at the top of the form.
* test-options - Comment box for the test options section.
* tentative-choice - Comment box for the tentative-choice section.
* dilemma - Comment box for the whole form.

Relations

Care Ethics

* care-form-stakeholders - Comment box for stakeholders section of Care Ethics form.
* care-form - Comment box for the whole Care Ethics form.

Intersectionality

* stakeholder-feedback-${index} - Feedback for a stakeholder in the intersectionality form.
* intersect-form - Feedback for the whole intersectionality form.

Seven Generations

* generations-stakeholders - Comment box for the stakeholders section.
* generations-form - Comment box for the whole form.

Consequences

Stakeholders

* stakeholder-analysis - Comment box for stakeholder analysis in the Stakeholders form.
* cons-stakeholders - Comment box for the whole Stakeholders form.

Utilitarianism

* cons-util-bentham - Feedback for the whole Bentham form.
* cons-util-mill - Feedback for the whole Mill form.

Action and Duty

Reason

* critical-questions - Comment box for the whole critical questions form.
* categorical-imperatives - Comment box for the whole Categorical Imperatives form.

Roles

* moral-duties - Comment box for the moral duties section.
* personal-sacrifices - Comment box for the whole form.

Past Actions

* duties-sliders - Feedback for sliders on the Past Actions form.
* duties-versus-actions - Comment box for the whole form.

Character and Virtue

Virtue Ethics

* domains - Feedback for the domains section.
* virtue-ethics - Feedback for the whole form.

Life Path

* life-path-sliders - Feedback for sliders on the Life Path form.
* life-path - Feedback for the whole form.

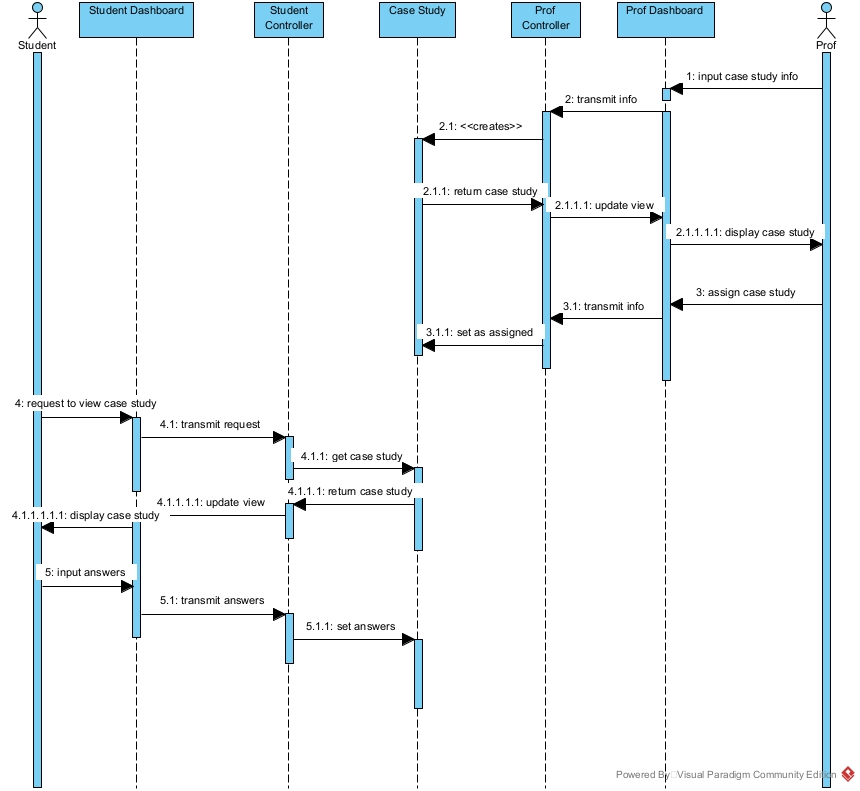
Universal Principles

* up-sliders - Feedback for the sliders.
* universal-principles - Feedback for the whole form.

# 

# Sequence Diagram

Interaction with Professor side through the shared DB.



# 

# How to add a model

1. Define the model

Create a new Python file in your models directory (e.g., models/my\_model.py) and

define the model using SQLAlchemy.

**Example:**

from .db import db, ma

class MyModel(db.Model):

\_\_tablename\_\_ = "my\_models" # Replace with your table name

id = db.Column("id", db.Integer, primary\_key=True)

name = db.Column("name", db.String, nullable=False)

def \_\_init\_\_(self, name):

self.name = name

def json(self):

return {'id': self.id, 'name': self.name}

def \_\_repr\_\_(self):

return f"MyModel({self.id}, {self.name})"

@classmethod

def get\_by\_id(cls, id):

return cls.query.filter(cls.id == id).first()

@classmethod

def get\_by\_name(cls, name):

return cls.query.filter(cls.name == name).first()

1. Create a Schema for Marshmallow

To serialize and deserialize the model, create a schema using Marshmallow.

class MyModelSchema(ma.SQLAlchemyAutoSchema):

class Meta:

model = MyModel

session = db.session

load\_instance = True

1. Register the model in \_\_init\_\_.py

To ensure the model is recognized by the application, add an import in the models/\_\_init\_\_.py file.

from .my\_model import MyModel, MyModelSchema

Then, add MyModel and MyModelSchema to \_\_all\_\_:

\_\_all\_\_ = [

# Other models...

'MyModel',

'MyModelSchema',

]

1. Manually add the model to your database with DDL

(we don’t have migration set up to capture changes to the DB after the initial container has been built, if you don’t want to do this you have to delete your DB container and rebuild it from scratch, but you will lose all data inside it).

# 

# How to add a controller/API route

Follow these steps to create and register a new controller in your Flask backend:

### **1. Navigate to the Controllers Directory**

Go to the backend/controllers directory in your project.

### **2. Create a New Controller File**

Create a new Python file named after your desired controller. The file name should reflect its purpose, e.g., student\_controller.py for handling student-related endpoints.

### **3. Import Necessary Modules**

At the beginning of your new controller file, import the required modules:

from flask import Blueprint, jsonify, make\_response, request

from .models import YourModel # Replace with your actual model

### **4. Register Your Controller with a Blueprint**

Create a Blueprint for your controller. The first argument should be a unique identifier for your controller.

bp = Blueprint('your\_controller', \_\_name\_\_) # Replace 'your\_controller' with an appropriate name

### **5. Define API Routes**

Create your API routes within the controller. Below is an example of a GET request to fetch all students:

@bp.route('/api/flask/students', methods=['GET'])

def get\_students():

try:

students = YourModel.query.all() # Replace with your actual model

students\_data = [{'id': student.id, 'name': student.name} for student in students]

return jsonify(students\_data), 200

except Exception as e:

return make\_response(jsonify({'message': 'Error retrieving students', 'error': str(e)}), 500)

Replace:

* /api/flask/students with the appropriate endpoint.
* methods=['GET'] with the applicable HTTP methods (POST, PUT, DELETE, etc.).
* YourModel.query.all() with the relevant query.

### **6. Register the Controller in app.py**

Open backend/app.py and ensure the new controller is imported:

from .controllers import your\_controller # Replace with your controller file name

Then, register the blueprint in the create\_app() function:

app.register\_blueprint(your\_controller.bp)

### **7. Restart Your Flask Server**

To apply the changes, rebuild the docker containers:

docker-compose build

docker-compose up

### **Summary of Steps**

1. **Navigate to** /backend/controllers.
2. **Create a new Python file** named after your controller.
3. **Import necessary modules** (Blueprint, jsonify, make\_response, and your models).
4. **Register a Blueprint** and define API routes.
5. **Add the controller to app.py** and register its blueprint.
6. **Rebuild the docker containers** to apply changes.

# 

# How to add a view

Our frontend uses **Next.js** with an **App Router structure**, so follow these steps to properly add a new page or UI component.

### **1. Creating a New Page**

To add a new page:

1. **Navigate to** /frontend/app.
2. **Create a new folder** inside app/, named after your new page.  
   * For example, if you're adding a "profile" page, create /frontend/app/profile.
3. **Inside the new folder, create the following files:**
   * page.tsx → This is the main entry point for the page.
   * layout.tsx → (Optional) If your page needs a custom layout, define it here.

#### **Example page.tsx:**

export default function ProfilePage() {

return (

<div>

<h1>Profile</h1>

<p>Welcome to your profile page!</p>

</div>

);

}

### **2. Adding UI Components**

If your page requires reusable UI elements, **do not** define them directly in page.tsx. Instead:

1. **Navigate to** /frontend/app/ui.
2. **Create a new folder** for your page’s UI components.  
   * If the page is inside the dashboard, create a **subfolder inside the dashboard** (e.g., /frontend/app/ui/dashboard/profile).
3. **Add UI components or forms** inside this folder.

#### **Example Component (/frontend/app/ui/profile/ProfileForm.tsx):**

"use client";

import { useState } from "react";

export default function ProfileForm() {

const [name, setName] = useState("");

return (

<form>

<label>Name:</label>

<input value={name} onChange={(e) => setName(e.target.value)} />

<button type="submit">Save</button>

</form>

);

}

1. **Import UI Components into page.tsx** Modify page.tsx to include the UI component:

import ProfileForm from "@/app/ui/profile/ProfileForm";

export default function ProfilePage() {

return (

<div>

<h1>Profile</h1>

<ProfileForm />

</div>

);

}

### **3. Reusable Components & Utility Functions**

* **Reusable UI components** (used across multiple pages) go in:  
   /frontend/app/ui/components/  
  + Example: Buttons, Modals, Custom Input Fields.
  + Example file: /frontend/app/ui/components/Button.tsx
* **Utility functions** (helper functions, data formatting, API calls, etc.) go in:  
   /frontend/app/utils/  
  + Example: Fetching user data, formatting timestamps.
  + Example file: /frontend/app/utils/fetchUser.ts

#### **Example Utility Function (fetchUser.ts):**

export async function fetchUser(userId: string) {

const res = await fetch(`/api/users/${userId}`);

if (!res.ok) throw new Error("Failed to fetch user data");

return res.json();

}

### **4. Testing Your Page**

Once you've added your page and UI components, **start the development server** to test it:

pnpm dev

Visit http://localhost:3000/profile (or your page's route) to see it in action.

### **5. Summary of Steps**

**Create a new page:**

* Add a folder inside /frontend/app/.
* Create page.tsx and optionally layout.tsx.

**Add UI components:**

* Place page-specific UI in /frontend/app/ui/.
* Reusable components go in /frontend/app/ui/components/.

**Organize utility functions:**

* Store helper functions in /frontend/app/utils/.

**Test your page** by running npm run dev.

# 

# How to create and run tests

## **1. Navigate to the Test Directory**

All tests are stored in the /backend/tests/ folder.

cd backend/tests

If this directory doesn’t exist, create it:

mkdir -p backend/tests

## **2. Create a New Test File**

Create a new test file in the /backend/tests/ directory. The file should be named test\_<feature>.py, where <feature> is the feature being tested.

Example:

touch test\_slider\_questions.py

## **3. Import Necessary Modules**

At the top of your test file, import required dependencies:

import os

import json

import pytest

from datetime import datetime

from backend.models import CaseStudy, Form, SliderQuestion

* pytest is our testing framework.
* json is used for parsing responses.
* datetime is for handling timestamps.
* backend.models contains the database models needed for testing.

## **4. Write Your Test Function**

Each test function should follow this format:

* **Setup**: Create necessary database entries.
* **Action**: Make an API request using client.
* **Assertions**: Verify the response status and data.

## **5. Run Your Tests**

Ensure your virtual environment is activated, then run:

pytest

To run a specific test file:

pytest backend/tests/test\_slider\_questions.py

To see detailed output:

pytest -v

# 

# Using the Server and Deployment

The current server used for deployment is a ubuntu virtual machine. You can connect to it via ssh cosc471@213.255.209.162 with password cosc471 in a terminal window.

## Deployment on the Server

### 1. Connect to server via terminal window

ssh cosc471@213.255.209.162

When the request for a password appears enter

cosc471

### 2. Navigate to the ethicdDashboard directory

cd ethicdDashboard

### 3. Build the Docker containers

sudo docker compose build

### 4. Run the Containers

sudo docker compose up

## Production Testing Environment

When making any changes to the project, rebuilding the docker containers can sometimes take upwards of ten minutes, a workaround to waiting is to run npm in a production environment. The environment variables will use values from .env.

### Navigate to ethicdDashboard

1. Run docker compose using the development environment

sudo docker compose -f compose.dev.yml up —build

### 