Ethics Dashboard 1 Requirement 1

Brandon Krieg, Monica Rampaul, Sophia Joseph, Alvin Villafranca

Our software

We will be creating an Ethics dashboard as a resource for students in an applied ethics course to further their learning, provide extra help where needed, and submit their work. After submitting their work, the professor or the teaching assistant should be able to give the student a grade on the ‘My Progress’ page and add any comments necessary. Additionally, there will be a database which will hold the information of each student, along with their grades and submissions

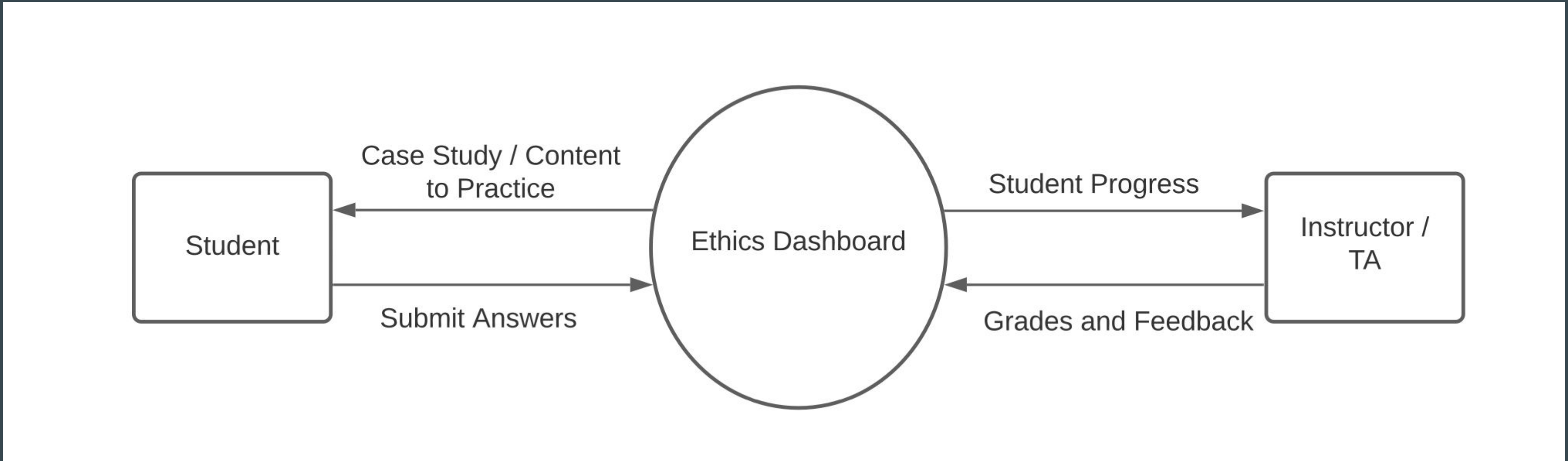
User Groups

There will be three main user groups for this software:

1) Students
   a) Students will be the main users of this application as they will be using it as a resource to their learning and will be able to track their progress and submit their work throughout the course
   b) Should not be able to view the database with all the information
2) Teaching Assistants
   a) The teaching assistants for the course should be able to view the students grades and be able to perform any necessary modifications
   b) Should be able to view the database which holds all the information
3) Professor
   a) The professor of the course needs to have access to this application in a different way than the students do and be able to override the teaching assistant if deemed fit
   b) The professor needs to be able to access everyone’s grades and information, which should not be available to students

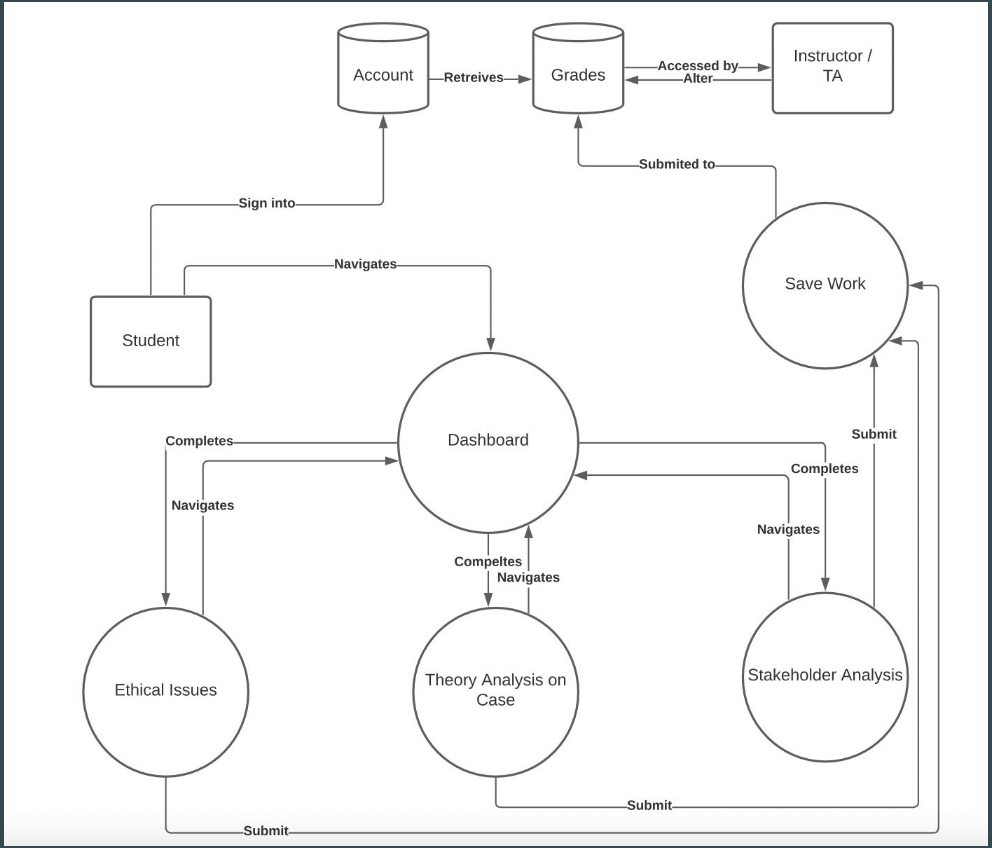
DFD - Level 0

- Student receives prompts to practice / answer
- Student submits answers to Ethics Dashboard
- The instructor or TA grades the student's work and gives feedback through the Ethics Dashboard
- The instructor and TA can access and view student progress from the Ethics Dashboard



DFD - Level 1

- The student signs in through the "Account" database, which is connected to the "Grades" database that is able to be accessed and altered by instructors and TAs. **This will be delivered for peer testing #2.**
- Student can navigate the Dashboard and complete Ethical Issues, Theory Analysis on Case, and a Stakeholder Analysis which each get submitted in order to save their work. **This will be delivered for peer testing #1.**
- Once the work is saved, it is submitted to the "Grades" database for grading and feedback from the instructor/TA. **This will also be delivered for peer testing #2.**



Functional Requirements for Each Milestone

- Milestone 1: Structure and Navigation
  - Basic skeleton of the website(HTML) and the linked pages should be set up. This means each ethical theory tab (Utilitarianism, Deontology, Virtue Ethics, and Care Ethics) should be linked to the navigation of the main Dashboard page.
  - Basic styling using CSS should be set up and consistent through all pages
- Milestone 2: Scripts
  - Buttons can be clicked which will take you to the appropriate page
  - Forms / Assignment questions can be edited and submitted
  - Students can navigate around the site with ease
  - Students can view questions / prompts
- Milestone 3: Backend
  - User Account functionality (temporary testing without a database yet)
  - Professor can access student account for grading
  - Student and instructor reports page implemented
- Milestone 4: Database and Accounts
  - Databases up and running for accounts, grades, progress, and reports
- Milestone 5: Comprehensive Testing & Rollout
  - Testing of all features that work together and have been implemented
  - Databases are working and are manageable for the future and maintenance

Non-Functional Requirements and Environmental Constraints

HTML, CSS, JavaScript based, familiar with the team

Hosted by: https://www.okanaganbcwebhost.ca

Web hosting and domain name defined by client

1.95GBs of space but is flexible in terms of upgrades, stated by the client that this should be enough space but can be upgraded

Power consumption is handled by client and college, not an issue for development team

Future maintenance and development by future students and peers, thus code should be well documented and easy to understand

Tech Stack Introduction

Our choices are based on technology we are familiar with, popularity of tech, documentation and support, and whether or not the server host requires the use of a specific tech or doesn't even support the use of another. Our client is very flexible with the technology we use.

django CentOS

MySQL Apache JS HTML

Front End - HTML

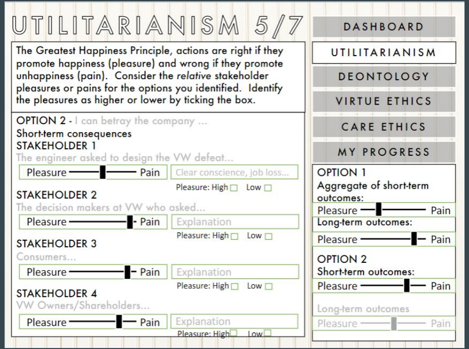
Our markup language of choice is HTML. Our reasoning is that it's fairly easy to develop in while still supporting many of the features that the client requested. Maintainability and updates are easy to document and implement due to the simple nature of HTML and its support of stylesheets, client-side and server-side scripts

Front End - CSS

Styling the dashboard will be done by CSS, since CSS already compliments HTML it should be no competition why we decide to go with CSS.

CSS allows for global styling which is not only easy for following the development process but also changes can be implemented without much effort

A lot of the pages of the dashboard will look similar

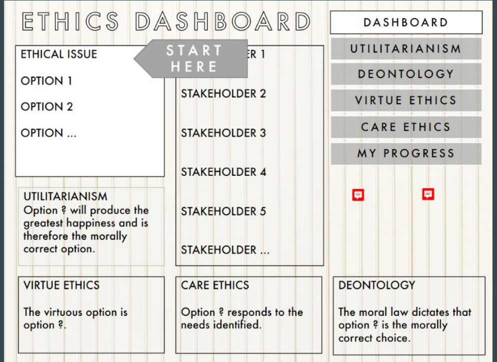


UTILITARIANISM 5/7

The Greatest Happiness Principle, actions are right if they promote happiness (pleasure) and wrong if they promote unhappiness (pain). Consider the *relative* stakeholder pleasures or pains for the options you identified. Identify the pleasures as higher or lower by ticking the box.

OPTION 2 - I can betray the company ...
Short-term consequences
STAKEHOLDER 1
The engineer asked to design the VW defeat...
Pleasure Pain | Clear conscience, job loss...
Pleasure: High ☐ Low ☐
STAKEHOLDER 2
The decision makers at VW who asked...
Pleasure Pain | Explanation
Pleasure: High ☐ Low ☐
STAKEHOLDER 3
Consumers...
Pleasure Pain | Explanation
Pleasure: High ☐ Low ☐
STAKEHOLDER 4
VW Owners/Shareholders...
Pleasure Pain | Explanation
Pleasure: High ☐ Low ☐

OPTION 1
Aggregate of short-term outcomes:
Pleasure Pain
Long-term outcomes:
Pleasure Pain
OPTION 2
Short-term outcomes:
Pleasure Pain
Long-term outcomes
Pleasure Pain



ETHICS DASHBOARD

ETHICAL ISSUE
OPTION 1
OPTION 2
OPTION ...

ER 1
STAKEHOLDER 2
STAKEHOLDER 3
STAKEHOLDER 4
STAKEHOLDER 5
STAKEHOLDER ...

UTILITARIANISM
Option ? will produce the greatest happiness and is therefore the morally correct option.

VIRTUE ETHICS
The virtuous option is option ?.

CARE ETHICS
Option ? responds to the needs identified.

DEONTOLOGY
The moral law dictates that option ? is the morally correct choice.

Front End - JavaScript

Javascript will further allow us to implement functions into the dashboard without the compromise of making the development process complicated since javascript can be easily implemented with CSS and HTML

Quite popular and a standard language to have some knowledge in for developers

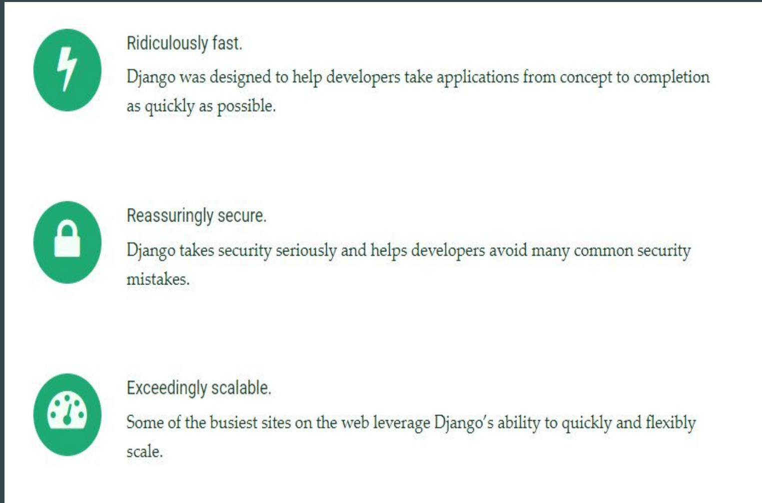
Backend - Django

Django is a high level web framework based on python which allows for fast development and design

Its free to use and is open source which means it is quite flexible for many web development projects

Growing in popularity

Django has templates which ensure database security



Ridiculously fast.

Django was designed to help developers take applications from concept to completion as quickly as possible.

Reassuringly secure.

Django takes security seriously and helps developers avoid many common security mistakes.

Exceedingly scalable.

Some of the busiest sites on the web leverage Django's ability to quickly and flexibly scale.

Backend - SQL, MySQL

Database - SQL:

Most popular query language for relational databases.

Massive support and documentation.

Team is familiar with SQL, and most developers familiar with web programming will be too.

Database Management System - MySQL:

Most popular DBMS.

Massive support and documentation, backed by Oracle.

Supported by numerous hosts.

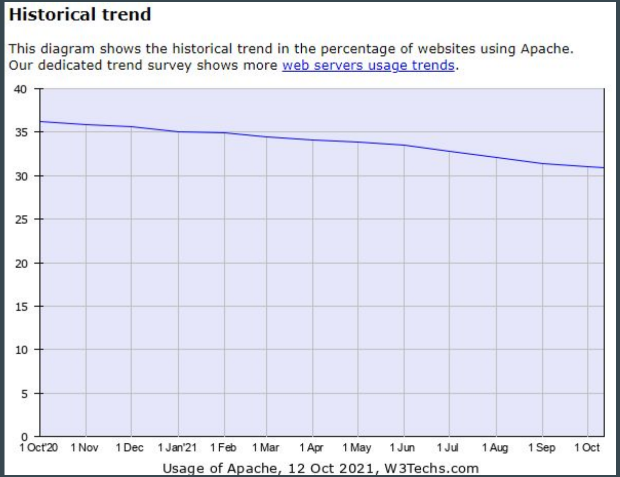
Backend - Apache Server and CentOS Linux

free and open-source software

Fairly common across the internet as its one of the oldest web deployment systems in history

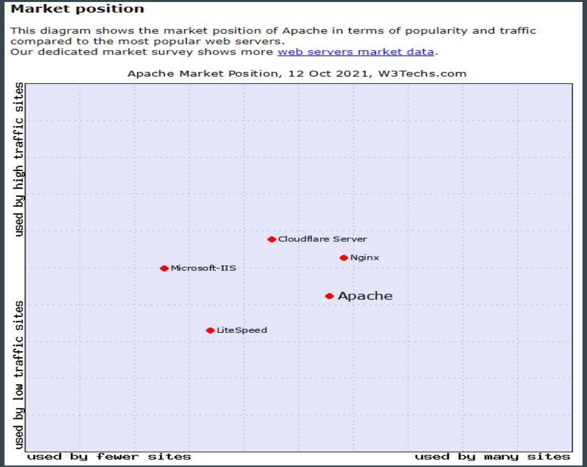
More secure than Ubuntu Server

Spinoff of Red Hat Enterprise Linux (RHEL)



Historical trend

This diagram shows the historical trend in the percentage of websites using Apache.
Our dedicated trend survey shows more web servers usage trends.



Market position

This diagram shows the market position of Apache in terms of popularity and traffic compared to the most popular web servers.
Our dedicated market survey shows more web servers market data.

Closing Remarks

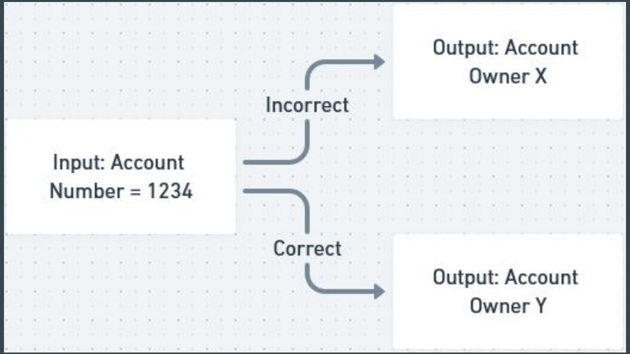
Our choice of tech stack was based-off a few factors including:

- Development Team comfortability
- Flexibility of the client (free reign)
- Popularity of present demographic
- Relation of project deliverables

General Testing Strategies

For each milestone we will adopt a functional testing strategy to validate that the application behaves as expected - this means we will be testing various inputs for expected outputs, or verifying the system does what it should.

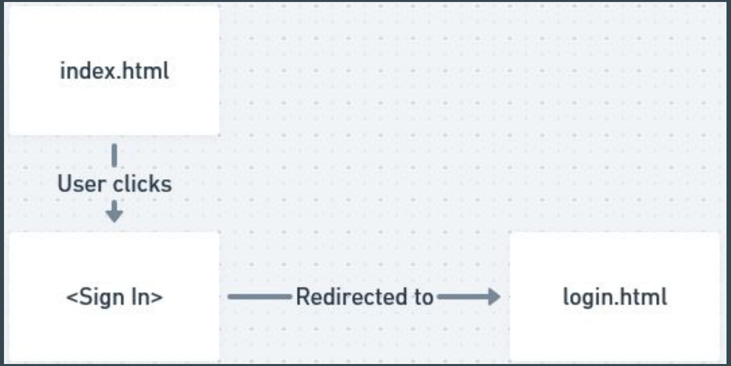
Near the end of the milestone, once all functional requirements have been met for that specific portion, we will adopt a non-functional testing approach to verify that the application performs adequately well - essentially, how well does it run under load, backwards compatibility, etc.

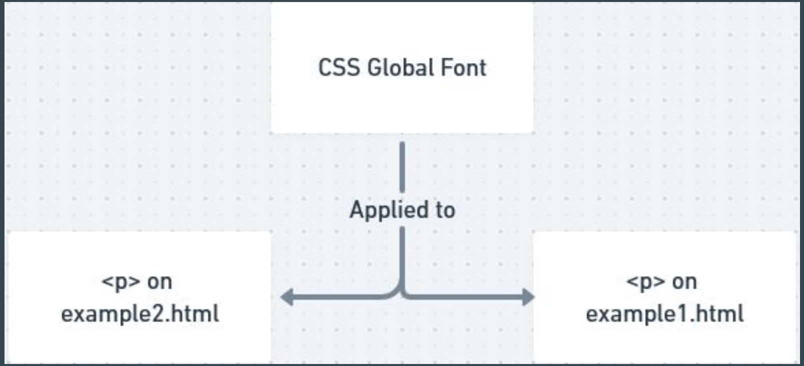


Testing HTML, CSS

The extent of testing for these two upper layers of our tech stack will be very limited, as they are quite static. Essentially, the HTML will be checked to make sure there are no broken links, the UI is correctly laid out, the correct elements are being displayed for the given page (e.g., radio buttons on a page for multiple choice questions.)

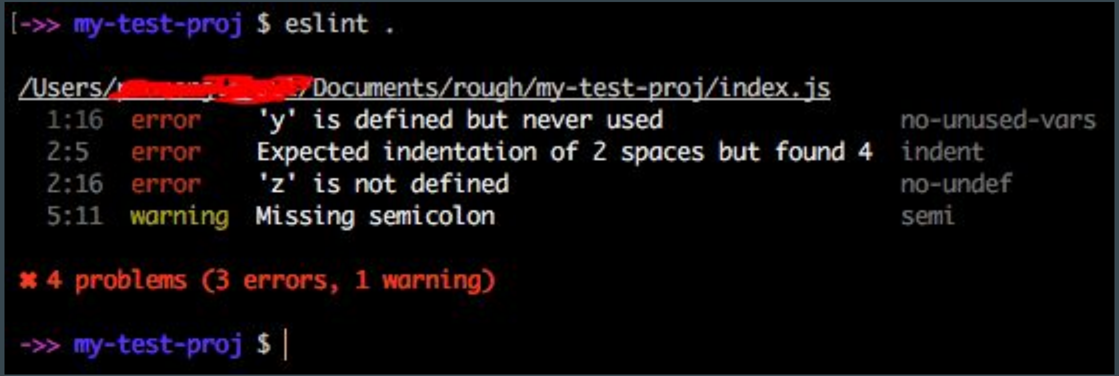
As for CSS, we will only need to make sure that the styles are correctly applied to each HTML page, and that the style aesthetic is appropriate.





Testing Client-side Scripting

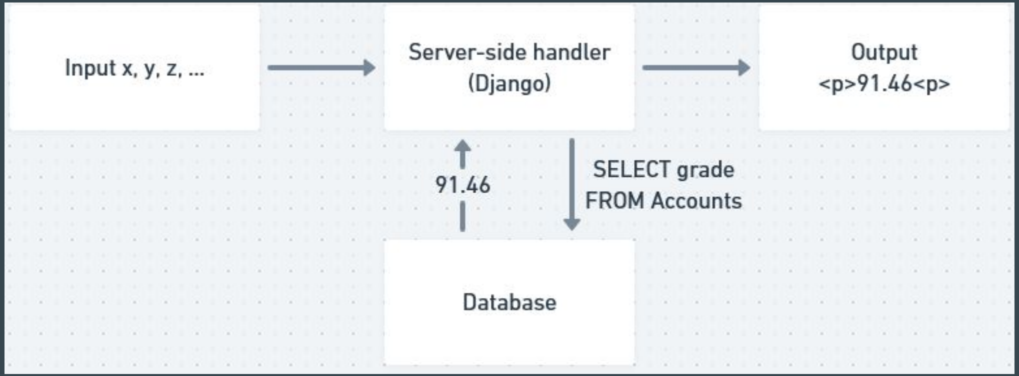
For our client-side, JavaScript, we will use the ESLint code linter. Since the extent of client-side scripting will be handling radio buttons, checkboxes, and other trivial elements, we will not need rigorous runtime testing. ESLint will be used to ensure code quality by searching for possibly problematic code. This will reduce any bugs or inefficiencies.



```
[->> my-test-proj $ eslint .

/Users/[redacted]/Documents/rough/my-test-proj/index.js
  1:16  error    'y' is defined but never used                no-unused-vars
  2:5   error    Expected indentation of 2 spaces but found 4  indent
  2:16  error    'z' is not defined                           no-undef
  5:11  warning  Missing semicolon                            semi

✖ 4 problems (3 errors, 1 warning)

->> my-test-proj $ |
```

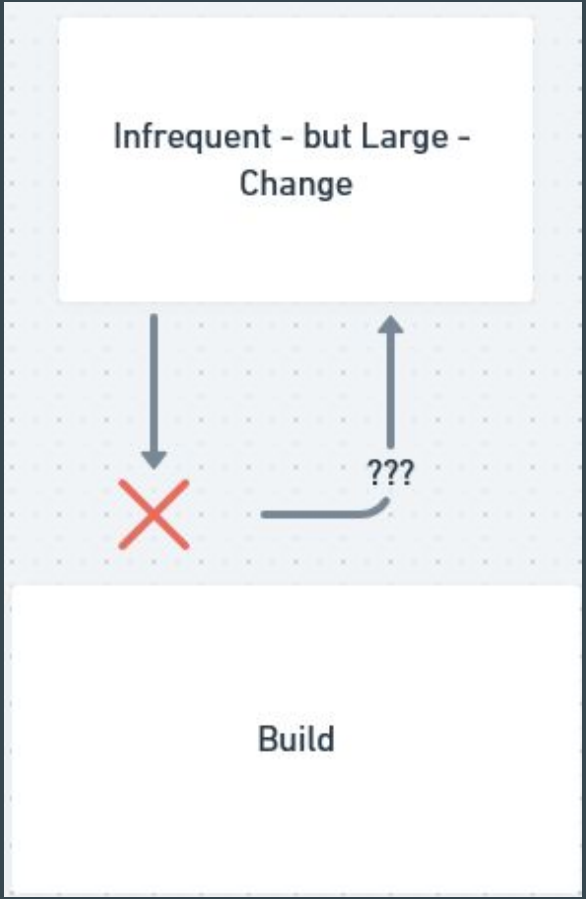
Testing Server-side Scripting and Database

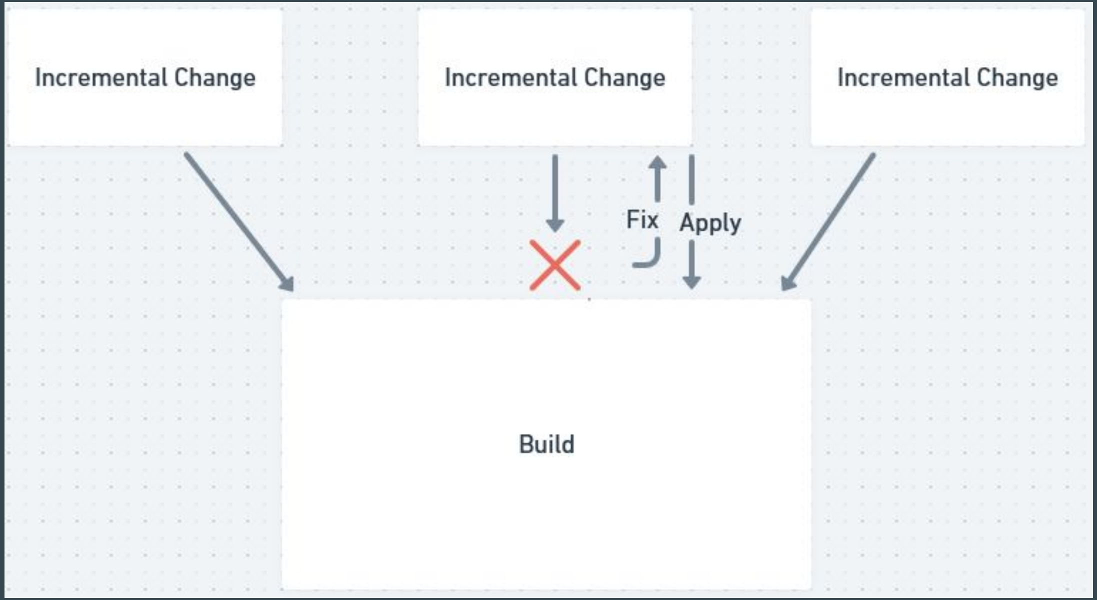
For our server-side, and the database that it accesses, we will use the Python standard `unittest`. This will allow us to determine if the system responds as expected with GET and POST requests from the user, as well as if correct results or correct manipulations to the database are completed through the script. This will be the most rigorous testing by far, as database correctness is of the highest importance.



Continuous Integration

To ensure continuous integration throughout the development cycle, we will adopt a very modular form of file interaction - as well as frequent code commits by all team members. A modular development environment ensures that any one change cannot break the entire system, as changes are made to individual files that are only *linked* to others. Frequent code commits ensure that any problems will be trivial, as the problem will have occurred in a small subset of code and thus be easily found and remedied.


Infrequent - but Large - Change
???
Build


Incremental Change
Incremental Change
Incremental Change
Fix
Apply
Build

Regression Testing

Since we will be developing the system in milestones (Basic structure and navigation first, client-side scripting, server-side scripting, then finally database implementation) we will need to ensure previous functionality is behaving as expected with the addition of these sweeping changes. Expanding upon previous tests, more inputs would be added to the test cases to reflect the added functionality (for example, does the HTML correctly display information from the database - e.g., account name at the top of the page.) Since our mode of continuous integration involves modular code and incremental changes, it will be easy to discover and fix problems that arise.