Open-Source Technology Use Report

Proof of knowing your stuff in CSE312

- site-packages/websocket/_abnf.py [framebuffer]
- site-packages/uvicorn/protocols/http/http_impl.py
- Starlets/requests.py -> handles http requests. Contains buffering code, as well as json parsing/prep, multipart form parsing
- Starlette/datastructures.py -> contains code to deal with request headers
- FastAPI is built ontop of starlette, basically all functionality is imported from starlette
- Creating an instance of a FastAPI object creates an instance of a starlette object

DISCLAIMER FOR GRADERS-

We made a mistake in choosing FastAPI as our framework. FastAPI is extremely abstracted- it is built on top of Starlette and Pydantic. The actual TCP server is run with uvicorn. Starlette handles all of the important webstuff for us, Pydantic deals with data validation and we don't really see much of those processes. Starlette maintains the bulk of the actual activity fastAPI instantiates. We know that starlette utilizes a few tools to handle the http and websocket functionality, specifically the python requests library, websockets library, as well as http_tools which is written in Cython. Getting stack traces is extremely difficult given how convoluted the connections are from library to library. I specifically went to Jesse to discuss how impossible fastAPI has been to report on, and he said not to worry about it and just do our best. So here is our best.

Guidelines

Provided below is a template you must use to write your report for each of the technologies you use in your project.

Here are some things to note when working on your report, specifically about the **General Information & Licensing** section for each technology.

- **Code Repository**: Please link the code and not the documentation. If you'd like to refer to the documentation in the **Magic** section, you're more than welcome to, but we'd like to see the code you're referring to as well.
- **License Type**: Three letter acronym is fine.
- **License Description**: No need for the entire license here, just what separates it from the rest.
- **License Restrictions**: What can you *not* do as a result of using this technology in your project? Some licenses prevent you from using the project for commercial use, for example.
- **Who worked with this?**: It's not necessary for the entire team to work with every technology used, but we'd like to know who worked with what.

Also, feel free to extend the cell of any section if you feel you need more room.

If there's anything we can clarify, please don't hesitate to reach out! You can reach us using the methods outlined on the course website or see us during our office hours.

[FastAPI]

General Information & Licensing

Code Repository	https://github.com/tiangolo/fastapi/tree/master/fastapi
License Type	MIT
License Description	● Permissive license, very limited restrictions on reuse ● Any software licensed under the terms of the MIT License can be integrated with software licensed under the terms of the GNU GPL
License Restrictions	● Unlike copyleft software licenses, the MIT License permits reuse within proprietary software, provided that all copies of the software or its substantial portions include a copy of the terms of the MIT License and also a copyright notice. ● As long as users include the original copy of the MIT license in their distribution, they can make any changes or modifications to the code to suit their own needs
Who worked with this?	Everyone

Use as many of the sections below as needed, or create more, to explain every function, method, class, or object type you used from this library/framework.

[APIRouter]

Purpose

Replace this text with some that answers the following questions for the above tech:

- APIRouter allows us to separate all path operations into its own file.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use APIRouter in src/controller/profile.py. It is imported on line two and the object is initialized on line 10.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L10

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - The APIrouter class creates a router object for us that maintains all of the information our app needs for defining an endpoint. The class is built on top of the starlette Router class. FastAPI simply creates the object so that it can be added to the main app.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

- First, we call APIRouter on line 13 of our profile.py file
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L13
- This creates an APIRouter object from line 457 of fastapi/routing.py
    - https://github.com/tiangolo/fastapi/blob/master/fastapi/routing.py#L457
- Creating this object is actually an instance of a starlette router. So we go to line 534 of starlette/routing.py.
    - https://github.com/encode/starlette/blob/master/starlette/routing.py#L534
- From there, the router is created with various init methods. Once those run the code returns back to profile.py.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py

[@router.post()]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - This method defines an endpoint for our webapp that allows for a post request. We can easily define the path that would be requested, and fastapi will take care of everything else under the hood.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 16 of our profile.py file. It is the endpoint we use to upload a profile picture to our app.
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controlle

r/profile.py#L16

Magic ★★ ° ˙ ° ☽ ° ⌒ ° ★ ≡ ✧ ⌯

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - FastAPI has a API Router class that we defined already. Using the @router.post method defines all of the necessary information like the path, uniqueID, and dependencies to create a route, and returns a new self.api_route for the router object in question.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start by defining @router.post() on line 16 of profile.py
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L16
- This takes us to line 922 of fastapi/routing.py. This is the return statement of the post method within the API Router class. It returns an api_route object to the API Router object.
    - https://github.com/tiangolo/fastapi/blob/master/fastapi/routing.py#L922
- Api_route defines all the necessary routing info, then returns the route in a decorator on line 646
    - https://github.com/tiangolo/fastapi/blob/master/fastapi/routing.py#L646

[@router.get()]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - This method defines an endpoint for our webapp that allows for a get request. We can easily define the path that would be requested, and fastapi will take care of everything else under the hood.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 27 of our profile.py file. It defines the get requests for getting a profile picture that a user uploaded, as well as grabbing CS:GO data, which is the primary function of our web app.
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L27

Magic ★★ ° ˙ ° ☽ ° ⁀ ° ★ ≡ ★ ⌯

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - FastAPI has a API Router class that we defined already. Using the @router.get method defines all of the necessary information like the path, uniqueID, and dependencies to create a route, and returns a new self.api_route for the router object in question.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start by defining @router.get() on line 27of profile.py
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L27
- This takes us to line 810 in fastapi/routing.py. This is the return statement of the get method within the API Router class. It returns an api_route object to the API Router object.
    - https://github.com/tiangolo/fastapi/blob/master/fastapi/routing.py#L810
- Api_route defines all the necessary routing info, then returns the route in a

decorator on line 646
    - https://github.com/tiangolo/fastapi/blob/master/fastapi/routing.py#L646

[Depends]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - Dependency injection is the ability of an object to supply dependencies of another object. Sometimes methods need certain data in order to function, thus we use one object to supply that data to another. With fastAPI, all we need to define is a Depends object within our APIRouter, and it will make sure our router has that data/object for its functionality.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 12 of our profile.py file. It defines the dependency of our router on the SQL database we use. For many of our endpoints, we will need the database in order to grab information for our clients.
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L12

Magic ★★ ° ˙ ˚ ☽ ˚ ⌒ ˚ ★ ≡ ★ ≀

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - Depends prepares our database by being called before the APIRouter is created. It does this by creating a Depends object that contains the database, then exposing it to the router. Depends also uses a cache functionality to save time, so when our database is already created it can quickly grab it without redoing the process.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start by defining some APIRouter values, specifically line 12 of profile.py where we set dependencies=[Depends(get_db)]. Get_db will return our database object.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L12
- Depends is defined on line 278 of fastapi/param_functions.py.
    - https://github.com/tiangolo/fastapi/blob/master/fastapi/param_functions.py#L278
- It returns a params.Depends object on line 281
    - https://github.com/tiangolo/fastapi/blob/master/fastapi/param_functions.py#L281
- The depends class is defined in fastapi/params.py on line 358. It sets the dependency within the init function as well as a boolean telling the function whether or not to use a cache.
    - https://github.com/tiangolo/fastapi/blob/master/fastapi/params.py#L358

[UploadFile]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - Uploadfile defines an object that has multiple methods for dealing with file uploads, like read, write, etc. It effectively abstracts all of the necessary code for dealing with file uploads that we would have had to do from scratch.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 17 of our profile.py file. It defines the input to our create profile picture post endpoint, which allows us to pass in a file object and read it to get its contents.
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L17

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - FastAPI takes in the file, and first validates it to make sure that it is of type starletteuploadfile. This makes sure that we are actually receiving a valid file. From there it creates an object of that type, which has all of our useful methods like read and write.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start on line 17 of profile.py. When a post request is made to /, the file is passed in and we create our UploadFile object.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L17
- This takes us to line 12 of fastapi/datastructures.py. In here we can see the UploadFile class is actually a wrapper for the StarletteUploadFile class. The fastAPI extension simply validates the file.

    - https://github.com/tiangolo/fastapi/blob/master/fastapi/datastructures.py#L12
- The StarletteUploadFile class is on line 420 of starlette/datastructures.py. In here we can see the properties of our new UploadFile object, which has an init method to define the filename, typing, and content type of whatever file we pass in.
    - https://github.com/encode/starlette/blob/master/starlette/datastructures.py#L420

[UploadFile.read()]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - Uploadfile.read is a method defined in the starletteuploadfile class. It allows us to read the data from any file we have passed in.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 19 of our profile.py file. We use it to read the contents of the profile picture a user uploaded to our webapp.
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L19

Magic ★★‧₊˚‧☾˚⁀➷‧₊˚★彡★༄

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
  - The read method here overloads the python standard library read method in python3.9/temp.py. Temp.py provides generic interfaces for creating temporary files and directories.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
  - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
  - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start on line 19 of profile.py. This is the file.read call that we make on the file the user uploaded.
  - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L19
- This takes us to line 456 of starlette/datastructures.py. This is the definition of the starletteuploadfile read method.
  - https://github.com/encode/starlette/blob/master/starlette/datastructures.py#L456
- Line 458 returns self.file.read. This takes us into the python standard library.
  - https://github.com/encode/starlette/blob/master/starlette/datastructures.py#L458
- Read is defined on line 730 of python3.9/tempfile.py within the spooledtemporaryfile class. From here we cant really get any deeper into the chain of calls due to the abstraction.

[FileResponse]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
  - FileResponse is a starlette class used by fastapi to send the http response necessary for sending back a file from a get request. We use it when someone makes a request for their profile picture.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.

- We use this on line 30 of our profile.py file. This return statement is the http response containing the file.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L30

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - This gets a bit difficult because of the abstraction in fastAPI. FileResponse is imported from fastapi, but it is actually just using the starlette class. The starlette class does not return an object, and instead it sends a response back to the client. Within the class, there is an async def __call__ function that deals with the asynchronous http response. Because it is asynchronous we can't really see where the http response is actually sent, but we can see that the content-length headers get set as well as things like the mime type.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start on line 30 of profile.py. This is the return fileresponse call that we make on the file the user requested.
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L30
    - This takes us to line 274 of starlette/responses.py. This is the init function of the fileresponse class from starlette.
        - https://github.com/encode/starlette/blob/master/starlette/responses.py#L264
    - Line 291 contains a method to guess the mime type of the file we are trying to send back to the client.
        - https://github.com/encode/starlette/blob/master/starlette/responses.py#L291
    - Line 294 calls self.init_headers, which is a method within the response class that sets the headers for the response.
        - https://github.com/encode/starlette/blob/master/starlette/responses.py#L291
    - Init_headers is on line 64. This method calculates the content length of the and also returns the content type.
        - https://github.com/encode/starlette/blob/master/starlette/responses.

py#L64
    - Once that is finished we go back into the fileresponse init method.
    - Line 320 of the FileResponse class contains the async def __call__ method that actually handles sending the file. Within it we can see that the method will deal with sending headers before it sends the body, which will happen in chunks. It uses an object of type Send that is defined in starlette/types.py, but it's very difficult to tell what is actually happening past this point.
        - https://github.com/encode/starlette/blob/master/starlette/responses.py#L64

[FastAPI]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - FastAPI creates an instance of the fastAPI class, which inherits its functionality directly from starlette. It is the main point of interaction for our API. Creating a fastAPI instance sets up routing, response, file parsing, basically everything that we would need to set up an API.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 18 of our main.py file. The app object we create gets called with uvicorn.run which is how we start our actual server.
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L18

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - FastAPI is built ontop of starlette, which is just another ASGI framework. When we create the instance of a FastAPI object, it creates class variables of all of the things you need for an API.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start on line 18 of main.py, where we create a FastAPI object
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L18
- This takes us to line 45 of fastAPI/applications.py which is the FastAPI class
    - https://github.com/tiangolo/fastapi/blob/master/fastapi/applications.py#L45
- From there we can see that the class is built ontop of the starlette class, which takes us to line 14 of starlette/applications.py
    - https://github.com/encode/starlette/blob/master/starlette/applications.py#L14

`

[HTTPException]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - HTTP Exception is a class imported from fastAPI that allows us to raise an http exception of any kind just by creating the object and calling “raise HTTPException()”. We use this to raise an exception when a username is already in use, or when the authentication information used to login is incorrect.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it’s used for a given purpose is fine as well.
    - We use this on line 20 and line 33 of our controller/auth.py file.
        - https://github.com/encode/starlette/blob/master/starlette/applications.py#L14
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controlle

r/auth.py#L33

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - HTTPException is a class that, like most things from fast api, is built on top of a starlette class. Specifically the StarletteHTTPException class. On initialization it takes in a status code, details, and headers. This will automatically set up the response for us, and we raise the exception it will send that response to the client.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- When we create a httpexception, we start on line 20 of auth.py.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.py#L20
- We then go into the fastapi/exceptions.py file, which contains the HTTPExceptions class on line 8.
    - https://github.com/tiangolo/fastapi/blob/master/fastapi/exceptions.py#L8
- From there we can step into StarletteHTTPException, which is on line 11 of starlette/exceptions.py.
    - https://github.com/encode/starlette/blob/master/starlette/exceptions.py#L11
- This class is built ontop of the Exceptions class from the standard library, which is on line 902 of python_stubs/-1776081290/builtins.py. We can actually go down all the way to the definition of Object, which is on line 732 of the same file.

- When we actually raise the exception we start on line 21 of auth.py. If a user already exists when trying to register, we raise a code 422 exception detailing the issue.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.py#L21
- From here, we start seeing a lot of the asyncio library as it deals with the raise call. We can follow the stack trace through many calls of various concurrency code all the way to asyncio/base_events.py, where we end up on line 596 within a while true loop which is inside the run_forever() method. I believe what happens here is the httpexception simply gets sent to the client, and then the server just goes back to waiting for something to happen.

[RedirectResponse]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - RedirectResponse is a starlette class used by fastapi to send the http response necessary for redirecting a client. We use it within our authentication code whenever someone registers or logs in to send them to either the login page or the homepage.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on lines 26, 40, and 48 respectively of controller/auth.py
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.py#L26
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.py#L40
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.py#L48

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - FastAPI actually doesnt even do anything on top of this class, this is a starlette class that just gets imported into fastapi/responses. RedirectResponse is the starlette class, which is built on top of the basic starlette Response class. This class has multiple methods for handling responses, such as setting and deleting cookies, rendering the content, and initializing headers. The redirect Response class provides functionality for creating a location header for the redirect response, and also defaults the response code to 307 if none is provided.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- On line 26, if we create the redirectresponse object.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.py#L26
- If we enter that object, it takes us to line 204 of starlette/responses.py. Here we can see that it sets a location header to whatever location we pass into the object.
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L204
- In the actual stack trace, we go to to line 204 of starlette/responses.py. It sets the redirect attributes like url to “/” (since I am tracing a login attempt, we redirect back to home).
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L204
- From there we go up to line 45 within the responses class in the same file, and call render content and then init headers. Once these are set we return back to the auth.py file to continue the operation.
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L57
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L64

[Form]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - Form is a FastAPI class. We use Form as an input type to our Register and login endpoints. Form allows us to take in the user information without manually parsing it, so we just get the inputs as intended.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on lines 16 and 30 respectively of controller/auth.py
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.py#L16
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.py#L30

Magic ★★ ° ˙ ° ☽ ° ⌒ ° ★ ≡ ★ ≪

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - Its extremely difficult to actually find where the parsing of Forms occurs, but fastAPI explains in the docs that in order to parse forms, you need to install python multipart. I have been digging through to try and find where that actually happens, but not only can i not find it, the stack traces show nothing. In the starlette/requests.py file there is a method called form where there is very clearly code to parse multipart forms on line 244, but it does not trace back to the Form class.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- On line 30 of controller/auth.py, our login endpoint takes a username and password as Forms.
- If we enter the Form object, we go to line 204 of fastapi/param_functions.py within the Form method. This returns a params.Form object on line 222.
- If we jump to that code, we go to fastapi/params.py line 279 which defines the

actual form class.
- The Form class is built on top of Body which is a class in the same file on line 235.
- Body is built ontop of FieldInfo. If we go to that code, we end up in pydantic/fields.py on line 93 which defines the FieldInfo class. At this point going any farther becomes pointless, as pydantic is not very human readable.

[response.set_cookie()]

Purpose

Replace this text with some that answers the following questions for the above tech:
- What does this tech do for you in your project?
    - The set cookie method allows us to quickly set a cookie of any kind before sending back a response. We use it to set an auth cookie when a user logs in.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 41 of controller/auth.py
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.py#L41

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - The set cookie method is defined in the starlette Response class, in starlette/responses.py. When it gets called, it creates an object from the http python library called http.cookies.SimpleCookie(). Simple cookie is built on BaseCookie which contains methods to handle creating and setting cookie values. Within the starlette code, it checks for the possible cookie directives and sets them if indicated by our method call. It eventually appends that cookie to the raw headers of the Response object.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start on line 41, where we call set cookie on our redirect response object.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.py#L41
- This takes us to line 104 of the starlette/responses.py file, which is within the Response class.
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L103
- From there, we call cookie: http.cookies.BaseCookie = http.cookies.SimpleCookie() which takes us to line 482 of python3.9/http/cookies.py. The init function runs and then we return back to the set_cookie method in responses.py.
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L115
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L103
- We go all the way down, setting directives until line 136 where we call cookies.output. This takes us back to python3.9/http/cookies.py to line 500.
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L136
    - https://github.com/python/cpython/blob/3.9/Lib/http/cookies.py#L500
- This formats all of our cookie information so that it follows http protocol, then we return back to the set_cookie method.
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L103
- It appends the cookie to the raw headers of the response and then returns back to auth.py.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.p

y

[response.delete_cookie()]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - The delete cookie method allows us to quickly remove a cookie of any kind before sending back a response. We use it to remove an auth cookie when a user logs out.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 49 of controller/auth.py
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.py#L49

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - The delete cookie method is defined in the starlette Response class, in starlette/responses.py. When it gets called, it creates an object from the http python library called http.cookies.SimpleCookie(). Simple cookie is built on BaseCookie which contains methods to handle creating and setting cookie values. Delete actually uses the set cookie method, but instead of setting values it sets everything to empty or none, so that the cookie that was there now no longer has a value.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start on line 49, where we call delete cookie on our redirect response object.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.py#L49
- This takes us to line 139 of the starlette/responses.py file, which is within the Response class.
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L139
- From there we actually end up calling set_cookie, but we set all of the values of the cookie to none, 0, or false. This takes us to line 103 of starlette/responses.py which is the set_cookie method.
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L103
- From there, we call cookie: http.cookies.BaseCookie = http.cookies.SimpleCookie() which takes us to line 482 of python3.9/http/cookies.py. The init function runs and then we return back to the set_cookie method in responses.py.
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L115
    - https://github.com/python/cpython/blob/3.9/Lib/http/cookies.py#L481
- On line 116 when we set cookie[key] = value, we go back into the python3.9/http/cookies.py file.
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L116
- The __setitem__ method runs, and when it sees that our value is empty, sets it as an empty string in the cookies dictionary on line 498 of python3.9/http/cookies.py file. After that we return back to the set_cookies method on line 103.
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L10

3
- We go all the way down, setting directives as 0 or none. For time to expire, it sets the current time so the now empty cookie gets removed completely. On line 136 we call cookies.output. This takes us back to python3.9/http/cookies.py to line 502.
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L136
- This formats all of our cookie information so that it follows http protocol, then we return back to the set_cookie method.
    - https://github.com/encode/starlette/blob/master/starlette/responses.py#L103
- It appends the cookie to the raw headers of the response and then returns back to auth.py.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/auth.py

[Query]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - Query allows us to take in a url encoding for our endpoint and automatically parses the data we need. We use it when someone searches for a steam id.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 38 of controller/pages.py
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/pages.py#L38

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - Query is set up the exact same way as Form. Its called as a method, which returns a Query object. The Query class is built on top of Param, and Param is built on top of Field info, the same class that is the base of Form. Pydantic presumably handles data type checking, but the actual parsing of the request happens in the starlette/requests.py file. There is a method called form where there is very clearly code to parse multipart forms on line 244, but it does not trace back to the Query class either.
    -
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- On line 38 of controller/pages.py, our profile endpoint takes steamID as a Query object.
- If we enter the Query object, we go to line 46 of fastapi/param_functions.py within the Query method. This returns a params.Query object on line 65.
- If we jump to that code, we go to fastapi/params.py line 104 which defines the actual query class.
- The Query class is built on top of Param which is a class in the same file on line 14.
- Param is built ontop of FieldInfo. If we go to that code, we end up in pydantic/fields.py on line 93 which defines the FieldInfo class. At this point going any farther becomes pointless, as pydantic is not very human readable.

[Request]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - Request allows for the automatic handling of incoming http requests. Requests is a class inherited directly from starlette, and it does everything from buffer, parse multipart forms, to parsing incoming cookies.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this basically anytime we get a request to our API. in controller/pages.py we use it on lines 20, 29, 38, 78, and 82.
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/pages.py#L20
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/pages.py#L29
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/pages.py#L38
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/pages.py#L78
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/pages.py#L82

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - The request class takes in the incoming http request and sets up the scope as well as a receive var and a send var. Request also has functionality for buffering if necessary within the stream function, and multipart parsing within the form method. The multipart parsing uses the python-multipart library to deal with multipart forms. It's difficult to get a full stack trace for the entire lifecycle of a request/response because of the nature of ASGI, but anytime a request is made we can see when the forms get parsed or buffered.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- For this stack trace, we will start in starlette/routing.py on line 59. This line defines a Request object from the scope of an app definition.
- Entering the object, we go to line 186 of starlette/requests.py which is the init method.
- The init method runs, setting the scope and some ASGI variables, then returns the object back to line 59.

[Websocket]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - Websocket sets up the functionality for our websockets. We use websockets for dms, as well as the functionality to see all logged in users.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 36 of main.py as well as throughout websocket.py to create websocket objects
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L36

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - The websocket class is taken by fastapi from starlette/websockets.py. The class is built ontop of the starlette httpconnection class, and it sets up all of the state variables necessary for a websocket interaction, such as setting the scope to websocket and establishing a client state. The class defines websocket methods like receive, send, and accept. Neither fastapi or starlette actually handle the handshake though, the handshake is done by uvicorn. The handshake will be shown with the websocket.accept() method, when you create a websocket object it has to call accept to actually connect.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- When a websocket object is created in main.py on line 36, there are multiple ASGI calls that occur.

    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L36
- After the async calls are all made, we end up on line 22 of starlette/websockets.py. This is within the Websocket class, which is built on top of the HTTP Connection class.
    - https://github.com/encode/starlette/blob/master/starlette/websockets.py#L22
- The init function runs, then returns a websocket object to line 78of starlette/routing.py
    - https://github.com/encode/starlette/blob/master/starlette/routing.py#L78
- Line 77 calls await func(session) on the new websocket object
- This takes us to line 260 of fastapi/routing.py. Here fast api gets any dependencies for the websocket.
    - https://github.com/tiangolo/fastapi/blob/master/fastapi/routing.py#L260
- When this function returns, we go back to line 77. When that returns, we end up on line 315 of starlette/routing.py in the handle method within the WebsocketRoute class. This is the end of starlette creating a route for the websocket endpoint we created.
    - https://github.com/tiangolo/fastapi/blob/master/fastapi/routing.py#L279
- We return to line 656 of starlette/routing.py. The route for our new websocket is now set up, the next step would be to connect to it.
    - https://github.com/tiangolo/fastapi/blob/master/fastapi/routing.py#L648

[WebsocketDisconnect]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - WebsocketDisconnect is a class defined in starlette/websockets.py that is used for catching a disconnect exception. We use it in main when someone disconnects from the websocket, the exception is caught and we remove the client from our websocket manager.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 68 of main.py
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L68

Magic ★★ ° ˚ ☽ ˚ ⌒ ˚ ★ ≡ ✧ ★ ≡

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - WebsocketDisconnect is built ontop of the base python Exception class for non-exit exceptions.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start on line 49 when a user disconnects from the website. This is a call to websocket.receive_text().
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L49
- This takes us to line 113 of starlette/websockets.py which is a call to self._raise_on_disconnect(message)
    - https://github.com/encode/starlette/blob/master/starlette/websockets.py#L113
- This takes us to line 105 of the same file, which is a call to raise WebSocketDisconnect(message[“code”])
    - https://github.com/encode/starlette/blob/master/starlette/websockets.py#L105
- The exception gets caught in main.py, so we jump to line 68 of main.py
    - https://github.com/encode/starlette/blob/master/starlette/websockets.py#L68
- We then remove the websocket from our websocket manager which uses our own code.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L69

[websocket.accept()]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - This allows us to wait to accept the initial websocket connection and perform initial logic before we keep the connection open to receive data like websocket.recieve_text(). We are able to authenticate the user and add the user to our class variable to keep track of the connection and username
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this in our main.py module as the first line in our websocket_endpoint.

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - websocket.accept() is a wrapper around the logic of the starlette websocket receive function
    - accept first checks if the client state is connecting, and if it is it calls self.recieve() which is in the WebSocket class
    - This inherits from starlettes HTTPConnection class which is the base class for all incoming HTTP connections (Request and websocket)
    - The receive function in the websockets module is the base for all ASGI websocket methods. If the client is connecting it sets the client state field to connected for future calls
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- Our first line of the websocket_endpoint in main.py is awaiting websocket.accept()
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L37
- Since this is the first time the client connects, it waits for the connect message using the receive method in starlette/websockets.py on like 75. The receive method is injected when the websocket class is instantiated as seen in starlette/websockets.py on line 24, with the receive method being line 29

- https://github.com/encode/starlette/blob/master/starlette/websockets.py#L75
- https://github.com/encode/starlette/blob/master/starlette/websockets.py#L21

[websocket.sendjson()]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
  - This function is how we broadcast json to all of the open websockets. Its how we can display all active users on the home page, as well as send direct messages.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
  - We use this on line 43 and 63 of main.py
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L43
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L63

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - Send json strings together the starlette library, uvicorn library, and the python websockets library. Starlette basically passes the message to uvicorn, which in turn passes it to websockets which actually handles the parsing of message chunks. The asynchronous calls make it hard to see the whole chain of calls, but we know the frame parsing occurs at the very end within the websockets library.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start on line 43 of our main.py file. After the user is verified, we call await websocket.sent_json()
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L43
- This takes us to line 134 of starlette/websockets.py. The function checks what type of message it is, then calls await self.send() on line 137.
- This takes us up to line 55 of the same file, within the send method. We go all the way down to line 68 where await self._send(message) is called.
- From here, we enter uvicorn/protocols/websockets/websocketss_impl.py. We are on line 208 within the async def asgi_send() function which handles async websockets.
- We go all the way down to line 252, skipping the handshake functionality since it has already been done. Here we call await self.send(data).
- This takes us to line 620 of websockets/legacy/protocol.py within the send method. The method runs then we return back to main.py line 43.

[websocket.receive_text()]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - This function is how we receive data at our websocket endpoint. It's how we receive all websocket messages. It is in the infinite loop in our websocket_endpoint method so we can continuously wait to receive data through our websocket.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 49 of main.py

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - Recieve_text is a wrapper around the same receive method mentioned previously. What it does is wait to receive the message from the receive method, then if its text then it calls json.loads to make it a json object, and if it is in bytes then it decodes it before returning the json object.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- It is first seen in our infinite loop in our websocket_endpoint in our main.py module.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L49
- The receive_text method is seen in starlette/websockets.py on line 107.
    - https://github.com/encode/starlette/blob/master/starlette/websockets.py#L107
- It then returns the text key of the message object, which is a str -> any pairing, which is most likely specific to the ASGI protocol that is implemented. The documentation for the ASGI specifications can be found here: https://asgi.readthedocs.io/en/latest/introduction.html

[CORSMiddleware]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - We are given an option using the add_middleware method on the FastAPI object which can intercept requests. The add_middleware method takes in a middleware_class and options. A specific option we used was CORSMiddleware that allows us to set the allowed hosts for cross origin resource sharing.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - This is used in our main.py module right after creating the FastAPI object around line 23.

Magic ★★ ° ˙ ˚ ☽ ˚ ⌒ ˚ ★ ≡ ✧ ≀

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    -
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- Once instantiated in line 22/23 of our main.py module, it instantiates a CORSMiddleware object with default values and just adds it to the user_middleware list in the Starlette object, which is inherited by the FastAPI object.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L22
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L23
- In the starlette docs it says "A list of middleware to run for every request.". So every request gets passed through the middleware.
- Once its called, indicated by the call dunder method (async def __call__) on line 74 of the starlette/middleware/cors.py module, it reads some of the request data and routes it either to the app method directly, or processes it with headers then sends it.
    - https://github.com/encode/starlette/blob/master/starlette/middleware/cors.p

y#L74

[StaticFiles]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - StaticFiles is a starlette class used by fastapi to mount static file locations to the app. This automatically creates routes for our web app so that it knows where to find our websockets.js file, which handles our dm functionality.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 20 of main.py
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L20

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - The Static Files class returns nothing, but rather adds the necessary path information to our fastAPI app so that it knows where to search for the required static files.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start on line 20 of main.py, where we call app.mount on the a StaticFIles object.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L20
- Stepping into the object creation, we go to line 42 of starlette/staticfiles.py. It sets the directory, packages, and then calls self.get_directories on line 51.
    - https://github.com/encode/starlette/blob/master/starlette/staticfiles.py#L42
    - https://github.com/encode/starlette/blob/master/starlette/staticfiles.py#L51
- This takes us to line 57 of the same file. We go through the function and return a list of the directories. In our case, it is simply “static”.
    - https://github.com/encode/starlette/blob/master/starlette/staticfiles.py#L57
- The rest of the init runs, then we return back to the mount call where the route gets added to our FastAPI() app.

[Jinja2Templates]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - We use Jinja2Templates to deal with our front end html templating. It is included in the fastAPI package, but is imported from the jinja2 package.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 30 of main.py, and line 14 of controller/pages.py, as well as line 11 of controller/profile.py.
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L30
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/pages.py#L14
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/profile.py#L11

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - Jinja2Templates works by identifying the directory where templates are, and then creating an environment around it that can be used to automatically populate templates and return template responses. The class itself contains the methods necessary for returning the responses and populating templates.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start on line 32 of main.py, where we call templates = Jinja2Templates(directory="templates"). This will create a jinja2template object that knows where the directory of our templates is.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L32
- This takes us to line 54 of starlette/templating.py. here we check that jinja2 is installed, then call self._create_env(directory) on line 65.

    - https://github.com/encode/starlette/blob/master/starlette/templating.py#L54
    - https://github.com/encode/starlette/blob/master/starlette/templating.py#L65
- This takes us to line 71 where the context of the request url is set. We then proceed to line 70 where we call jinja2.FileSystemLoader(directory)
    - https://github.com/encode/starlette/blob/master/starlette/templating.py#L71
    - https://github.com/encode/starlette/blob/master/starlette/templating.py#L75
- This takes us to line 184 of jinja2/loaders.py. We are within the init method for the filesystem loader, which sets up the path to our various html templates.
- We return back to starlette/templating.py and on line 79 call jinja2.Environment, passing in the loader we just created.
    - https://github.com/encode/starlette/blob/master/starlette/templating.py#L79
- This takes us to line 328 of jinja2/environment.py within the Environment class init method. It sets all of the necessary information then returns back to line 71 of starlette/templating.py.
    - https://github.com/encode/starlette/blob/master/starlette/templating.py#L79
- Finally the environment is returned to line 65 of starlette/templating.py.
    - https://github.com/encode/starlette/blob/master/starlette/templating.py#L65
- The init method of Jinja2Templates returns our new object to line 32 of main.py.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L32

[templates.TemplateResponse()]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - Template response sends a http response back when an endpoint is hit with a request, with the html file fully populated. We use it to serve all of our html pages.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this for every endpoint within controller/pages.py
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/pages.py#L25
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/pages.py#L34
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/pages.py#L73
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/pages.py#L79
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/pages.py#L84

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - Template Response uses jinja2 to find the template we want and load it into the jinja2environment which got created when we initialized a Jinja2Templates object. Once it loads in the template, it returns a http response fully formatted to send back to the client the HTML they requested.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start on line 34 of controller/pages.py within the home endpoint. Here we call templates.TemplateResponse().
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/controller/pages.py#L34
- This takes us to line 87 of starlette/templating.py within the TemplateResponse function. Here we check if the request is within the context variable.
- We continue to line 89 where we call get_template.
- This takes us to line 76 where we return self.env.get_template(name).
- This call takes us to line 992 of jinja2/environment.py. We check if the name passed is a template, and if not we go down to line 997 to call self._load_template()
- This takes us to line 943 of jinja2/environment.py. The method runs down to line 958 where we call self.loader.load()
- From there we go to jinja2/loaders.py line 119. This method is what actually does the lifting of loading in the template we want. Once it is done we return back to line 958 of jinja2/environment.py.
- The self._load_template method returns back to line 997 of the same file.
- The self.env.get_template() returns back to line 76 of starlette/templating.py within the get_template function.
- Finally we return back to the init method of TemplateResponse on line 89 of the same file.
- The template response returns the http response on line 34 of controller/pages.py.

[Uvicorn]

General Information & Licensing

Code Repository	https://github.com/encode/uvicorn
License Type	BSD 3-Clause
License Description	● A permissive license similar to the BSD 2-Clause License, but with a 3rd clause that prohibits others from using the name of the project or its contributors to promote derived products without written consent.
License Restrictions	Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: ● Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. ● Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. ● Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
Who worked with this?	

Use as many of the sections below as needed, or create more, to explain every function, method, class, or object type you used from this library/framework.

[Uvicorn-> Run]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
  - Uvicorn run allows us to create and run a ASGI server that our FastAPI application exists on top of.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
  - For local testing, we can use the line uvicorn.run(app, host="0.0.0.0", port=8000) in the main.py file, and for production with docker we have Uvicorn main:app —host 0.0.0.0 —port 8000 —reload on line 15 of our docker file.

        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/main.py#L76
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/Dockerfile#L15

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - Uvicorn creates an ASGI server by utilizing the asyncio library to handle asynchronous events. Asyncio effectively has eventloops that wait for certain things to happen while other things are occurring at the same time.

- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

- Starts at docker file line 15
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/Dockerfile#L15
- Uvicorn/main.py line 445 enters the run method
    - https://github.com/encode/uvicorn/blob/master/uvicorn/main.py#L445
- Creates server on line 542 of that method
    - https://github.com/encode/uvicorn/blob/master/uvicorn/main.py#L542
- Starts server on line 554 with server.run()
        - https://github.com/encode/uvicorn/blob/master/uvicorn/main.py#L554
    - This calls the run method on line 58 of Uvicorn/server.py
        - https://github.com/encode/uvicorn/blob/master/uvicorn/server.py#L58
    - That method returns a call to asyncio.run(self.serve(sockets=sockets)) on line 60
            - https://github.com/encode/uvicorn/blob/master/uvicorn/server.py#L60
        - In asyncio/runners.py, the run method is on line 8.
            - https://github.com/python/cpython/blob/3.9/Lib/asyncio/runners.py#L8
        - On line 44 it returns loop.run_until_complete(main)
            - https://github.com/python/cpython/blob/3.9/Lib/asyncio/runners.py#L44
        - That method is from line 611 of asyncio/base_events.py
            - https://github.com/python/cpython/blob/3.9/Lib/asyncio/base

_events.py#L611
        - On line 634, the method calls self.run_forever()
            - https://github.com/python/cpython/blob/3.9/Lib/asyncio/base_events.py#L634
        - That method is on line 588 of the same file
            - https://github.com/python/cpython/blob/3.9/Lib/asyncio/base_events.py#L588
        - Run_forever enters a try block that calls self._run_once() on line 596
        - _run_once() makes a call to handle._run() on line 1890
        - This takes us to asyncio/events.py line 80 of the Handle class. It calls self._context.run(self._callback, *self._args) which finishes setting up the ASGI server.
            - https://github.com/python/cpython/blob/3.9/Lib/asyncio/events.py#L80
    - The next part of this call occurs (self.serve(sockets=sockets)) which takes us to line 62 of Uvicorn/server.py.
        - https://github.com/encode/uvicorn/blob/master/uvicorn/server.py#L62
    - We get to line 77 of the serve method, which calls await self.startup(sockets=sockets)
            - https://github.com/encode/uvicorn/blob/master/uvicorn/server.py#L77
        - Startup is on line 87 of Uvicorn/server.py
            - https://github.com/encode/uvicorn/blob/master/uvicorn/server.py#L87
        - We go all the way through the method and get to server = await loop.create_server() on line 149
                - https://github.com/encode/uvicorn/blob/master/uvicorn/server.py#L149
            - This call takes us to asyncio/base_events.py
            - On line 1524 we create a server object after binding the sockets
                - https://github.com/python/cpython/blob/3.9/Lib/asyncio/base_events.py#L1524
            - Line 1527 is a call too server._start_serving()
                    - https://github.com/python/cpython/blob/3.9/Lib/asyncio/base_events.py#L1527
                - On line 317 in the same file we do one event loop, doing sock.listen(self._backlog) on line 318.
                    - https://github.com/python/cpython/blob/3.9/Lib/asyncio/base_events.py#L317
                - On line 319 self._loop._start_serving is called
                    - https://github.com/python/cpython/blob/3.9/Lib/asyncio/base_events.py#L319
            - Return a server object on line 1534 in asyncio/base_events.py
                - https://github.com/python/cpython/blob/3.9/Lib/asyncio/base_events.py#L1534
    - On line 80 we call await self.main_loop()
        - This takes us to line 222 of Uvicorn/server.py
        - The loop runs until the server gets an exit signal
            - https://github.com/encode/uvicorn/blob/master/uvicorn/server.py#L222

*This section may grow beyond the page for many features.

[SQLAlchemy]

General Information & Licensing

Code Repository	https://github.com/sqlalchemy/sqlalchemy
License Type	MIT License
License Description	● Permissive license, very limited restrictions on reuse ● Any software licensed under the terms of the MIT License can be integrated with software licensed under the terms of the GNU GPL
License Restrictions	● Unlike copyleft software licenses, the MIT License permits reuse within proprietary software, provided that all copies of the software or its substantial portions include a copy of the terms of the MIT License and also a copyright notice. ● As long as users include the original copy of the MIT license in their distribution, they can make any changes or modifications to the code to suit their own needs
Who worked with this?	Kyle

Use as many of the sections below as needed, or create more, to explain every function, method, class, or object type you used from this library/framework.

[sqlachemy -> Column]

Purpose

Replace this text with some that answers the following questions for the above tech: ● What does this tech do for you in your project? ○ Column is a sqlalchemy class that represents a column in a database table. We use column to create the sql database schemas that we need in order to store user information. ● Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well. ○ We use this on in the models files to create all of our tables. src/models/user and src/models/image.

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - SQLAlchemy Column object represents a column in a database table which is in turn represented by a Tableobject. When you create a column using sql alchemy, the constructor takes name, type and other parameters such as primary_key, autoincrement and other constraints. It then matches Python data to the best possible generic column data types defined in it.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start in models/image.py on line 13. Here we create the id column, passing in a type Integer and also indicating true for index and primary key
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/main/src/models/image.py#L13
- If we step into Column, we go to line 1320 of sqlalchemy/sql/schema.py within the init method of the Column class.
    - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/sql/schema.py#L1320
- We go all the way down to line 1677, parsing the arguments such as the name and type, until we call super(Column, self).__init__(name, type_)
    - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/sql/schema.py#L1887
- This takes us to line 4502 of sqlalchemy/sql/elements.py within the init method of the ColumnClause class. This class represents a column expression from any textual string.
    - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/sql/elements.py#L4502
- After this returns back to line 1320 of sqlalchemy/sql/schema.py we continue down the init setting things like foreign keys and comments, until exiting back to models/image.py line 13 to return the new column object that maps to an actual sql column.
    - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/sql/schema.py#L1320
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/models/image.py#L13

[sqlachemy -> Integer]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - The integer class is used to convert a python integer into a type recognized by sql.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - Found in image.py line 13 used to denote Integer type for column
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/models/image.py#L13

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - Its hard to tell exactly how the integer class works, as the code is very abstracted but when we use Integer, we are simply denoting to the sql database that it should expect an integer in that column. Within the sqlalchemy integer class there is no init method, but there are a few methods that we can assume perform type checks like python_type that returns an int.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- Line 13 of models/image.py defines a column with the type Integer.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/models/image.py#L13
- If we enter the Integer class, we go to line 349 of sqlalchemy/sql/sqltypes.py which defines the Integer class.
    - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/sql/sqltypes.py#L349
- Most of this is stubbed code, but we can see that it sets __visit_name__ to “integer” on line 353
    - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/sql/sqltypes.py#L353

[sqlachemy -> String]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - The string class is used to convert a python string into a type recognized by sql.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - Found in image.py (Line 14) and user.py
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/models/image.py#L14
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/models/user.py

Magic ★★ ° ˙ ° ☽ ° ⌒ ° ★ ≡ ★ ≫

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - The string class works similarly to Integer, but there is more going on in order to convert it to sql recognized data. At the top of the class there are comments explaining that python strings correspond to the VARCHAR type in SQL. The string class does have an init method that sets the length of the string, as well as variables designating if it needs to be converted to unicode.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start on line 14 of models/image.py where we define a filename column using the String type.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/models/image.py#L14
- Stepping into the string type takes us to line 172 of sqlalchemy/sql/sqltypes.py which defines the String class.
    - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/sql/sqltypes.py#L172

- Most of this is stubbed code, but we can see that it sets __visit_name__ to “string” on line 184.
    - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/sql/sqltypes.py#L184

[sqlalchemy.orm -> Session]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - Session basically prepares our database to do operations when we are going to use it. We use it in any endpoint that needs to get or place information in the database.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it’s used for a given purpose is fine as well.
    - Used frequently such as in main.py Line 38, and controller/profile.py line 20
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/main.py#L38
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/controller/profile.py#L20

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - Session establishes all conversations with the database and represents a “holding zone” for all the objects we loaded or associated with it during its lifespan. It provides the interface where SELECT and other queries are made that will return and modify ORM-mapped objects. The ORM objects themselves are maintained inside the Session, inside a structure called the identity map - a data structure that maintains unique copies of each object.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We will start on line 29 of controller/pages.py which defines the home endpoint. When that endpoint is hit, a session is created using Depends(get_db)
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/controller/pages.py#L29
- We can actually enter the stack trace by setting a breakpoint at line 1118 of sqlalchemy/orm/session.py which is within the Session class init method.
    - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/orm/session.py#L1211
- The init method sets variables like the autocommit var, and goes all the way down to line 1152, where it jumps to the return statement of the warned method on line 309 of sqlalchemy/util/deprecations.py. I don’t understand why this happens, but it must be some asynchronous call to warned because it doesn’t seem to be called in sqlalchemy/orm/session.py (these line references are made to local code).
    - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/util/deprecations.py#L294

[create_engine]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - Creates starting point (engine) for the database.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 7 from database.py
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/database.py#L7

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - The Engine is the starting point for any SQLAlchemy application. It's "home base" for the actual database and its DBAPI, delivered to the SQLAlchemy application through a connection pool and a Dialect, which describes how to talk to a specific kind of database/DBAPI combination. Create_engine creates a Dialect object tailored towards PostgreSQL, as well as a Pool object which will establish a DBAPI connection at localhost:5432 when a connection request is first received.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- We start in database.py on line 7 where we call create_engine
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/database.py#L7
- Following the stack trace takes us to line 279 of sqlalchemy/util/deprecations.py within the warned method. The warned method runs, but pycharm shows frames that arent available and then all of a sudden we return back to line 7 in database.py. It doesnt seem possible to actually fully stack trace this method.

[sqlalchemy.ext.declarative -> declarative_base]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - Constructs a base class for our user and image classes. Declarative_base sets up the architecture we need to use the ORM functionality of mapping python data classes to sql classes.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 11 of database.py
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/database.py#L11

Magic ★★ ° ˖ ° ☽ ° ⌒ ˖ ° ★ ≡ ★ ˖ ⌯

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - The declarative base function has comments that say "The new base class will be given a metaclass that produces appropriate :class:`~sqlalchemy.schema.Table` objects and makes the appropriate :func:`~sqlalchemy.orm.mapper` calls based on the information provided declaratively in the class and any subclasses of the class." Its very hard to actually tell what logic is occurring to link the python classes to sql data. When we try to stack trace the function, we get the same result as create_engine, where pycharm just says frames not available and then ends up in the "warned" function.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.
    - If there is more than one step in the chain of calls *(hint: there will be)*, you must provide links for the entire chain of calls from your code, to the library code that actually accomplishes the task for you.
    - Example: If you use an object of type HttpRequest in your code which contains the headers of the request, you must show exactly how that object parsed the original headers from the TCP socket. This will often involve tracing through multiple libraries and you must show the entire trace through all these libraries with links to all the involved code.

*This section may grow beyond the page for many features.

- It is literally impossible to stack trace this function, but we know that the declarative base function is on line 362 of sqlalchemy/orm/decl_api.py.

[sqlalchemy.orm -> sessionmaker]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
    - Sessionmaker does the lifting involved in getting a database session up and running. We use it indirectly whenever we call get_db() in dependencies.py, which calls sessionLocal which we set equal to sessionmaker. Sessions manage a conversation with a database when it occurs.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
    - We use this on line 9 in database.py
        - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/database.py#L9

Magic ★★ ° ˚ ☽ ˚ ⌒ ˚ ★ ≡ ★ ≫

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- Sessionmaker simply creates a session whenever it gets called. In order to create a session the sessionmaker needs an engine, which we create with create engine and then pass into sessionmaker. Running the stack trace shows us that it simply runs an init method that binds the engine, sets some other variables and then creates a subclass of session that has specific attributes to that engine instance.

- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.

- We start on line 9 of database.py where we call sessionmaker on the engine we created.
  - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/database.py#L9
- This takes us to line 4404 of sqlalchemy/orm/session.py.
  - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/orm/session.py#L4404
- The init function of sessionmaker runs and then we return back to line 9 of database.py with a session maker ready to create our sessions anytime we need the database.

[create_all]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
  - Create all is the method that actually creates the tables we want within the sql database.
- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
  - We use this on line 18 in main.py
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/main.py#L18

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- Create_all creates all of the tables stored in the metadata instance. This function is conditional, and won't recreate any already existing tables in the database. The function calls ddl.SchemaGenerator in order to actually create the tables.

*This section may grow beyond the page for many features.

- We start on line 16 of main.py, where we call create all on the engine we created in the database file.
  - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/main.py#L18
- This takes us to line 5256 of sqlalchemy/sql/schema.py within the create_all function.
  - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/sql/schema.py#L5256
- We run down to line 5281where we call ddl.SchemaGenerator within bind._run_ddl_visitor. That function runs and returns, then we follow the outer method call.
  - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/sql/schema.py#L5281
- Bind._run_ddl_visitor takes us to line 2874 of sqlalchemy/engine/base.py which calls self.begin()
  - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/engine/base.py#L2874
- Self.begin() takes us to 3035 (local) of sqlalchemy/engine/base.py within the begin method. It runs and we return back to _run_ddl_visitor
  - https://github.com/sqlalchemy/sqlalchemy/blob/main/lib/sqlalchemy/engine/base.py#L2868
- This returns back to 4864, then finally we return back to line 18 of main.py with our initialized tables.
  - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/main.py#L18

[requests]

General Information & Licensing

Code Repository	https://github.com/psf/requests
License Type	Apache Software License (Apache 2.0)
License Description	• Under the Apache license, users are permitted to modify, distribute, and sublicense the original open source code. Commercial use, warranties, and patent claims are also allowed with the Apache license. The terms and conditions under the license don't place any restrictions on the code, but end users cannot hold the open source contributors liable for any reason.

License Restrictions	● Apache license requires developers to disclose any major changes they make to the original source code. The modified source code does not need to be revealed, but a notice of the modification is required. However, any unmodified code must retain the Apache license.
Who worked with this?	Devin

Use as many of the sections below as needed, or create more, to explain every function, method, class, or object type you used from this library/framework.

[Requests.get]

Purpose

Replace this text with some that answers the following questions for the above tech:

- What does this tech do for you in your project?
  - Used in pages.py to retrieve data from steam api as json data containing user stats and headers

- Where specifically is this tech used in your project? Give us some details like file location and line number, if applicable. If too cumbersome, a general description of where it's used for a given purpose is fine as well.
  - Located in pages.py on line 44 as
  - requests.get(url=url, params=headers).json()
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/controller/pages.py#L44

Magic

Dispel the magic of this technology. Replace this text with some that answers the following questions for the above tech:

- How does this technology do what it does for you in the **Purpose** section of this report? Please explain this in detail, starting from after the TCP socket is created. Remember, to be allowed to use a technology in your project, you must be able to know how it works.
    - Requests .get goes through a few levels of abstraction before sending the get request to the url. The get method actually calls the request method to create a new session which establishes the http connection. From there the session object calls request, and continues down a chain of calls to various request methods until getting to the send methods, which actually parse the http request and extract all the headers before sending and receiving the response.
- Where is the specific code that does what you use the tech for? You ***must*** provide a link to the specific file in the repository for your tech with a line number or number range.

*This section may grow beyond the page for many features.

- We start on line 44 of controller/pages.py where we call requests.get on the steam csgo tracker api url.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/controller/pages.py#L44
- If we step into the method, we go to line 73 of requests/api.py within the get method. Line 73 returns request('get', url, params, **kwargs)
    - https://github.com/psf/requests/blob/main/requests/api.py#L73
- This takes us to line 58 of the same file where sessions.Session() is called
    - https://github.com/psf/requests/blob/main/requests/api.py#L58
- This takes us to line 389 of requests/sessions.py within the session init method.
    - https://github.com/psf/requests/blob/main/requests/sessions.py#L389
- The method runs and we return back with a session object to line 60.
- On line 59 we call session.request. This takes us to line 516 of requests/sessions.py within the request method. The method starts by creating a Request object.
    - https://github.com/psf/requests/blob/main/requests/api.py#L59
- This takes us to line 231 of requests/models.py within the Requests class init method.
- Our request object is created and then we return to line 516 of requests/sessions.py.
- On line 573 we call prepare_request, which takes us to line 440 of requests/sessions.py within the prepare_request method.
    - https://github.com/psf/requests/blob/main/requests/sessions.py#L573
- The method runs and then returns our fully prepared get request to line 528 of requests/sessions.py
- The request method continues all the way down to line 587 of request/sessions.py where we call self.send(prep, **send_kwargs)
    - https://github.com/psf/requests/blob/main/requests/sessions.py#L587
- This takes us to line 671 of requests/sessions.py within the send method.
- It runs all the way down to line 701 where we call another send method, adapter.send(request, **kwargs)
    - https://github.com/psf/requests/blob/main/requests/sessions.py#L701
- This takes us to line 436 of requests/adapters.py within the send method
- This method actually sends out the request, then at the very end calls self.build_response(request, resp) on line 584.

    - https://github.com/psf/requests/blob/main/requests/adapters.py#L584
- This takes us to line 296 of the same file within the build_response method.
    - https://github.com/psf/requests/blob/main/requests/adapters.py#L296
- The method runs and returns a response object back to line 584 of requests/adapters.py.
- We return back to line 701 of requests/sessions.py
- The send method continues down to return the response on line 699 of the same file.
- Finally we return back to line 587 of requests/sessions.py with our response within the request method
- We return back to line 51 of requests/api.py with our response
- We return back to line 73 of requests/api.py within the get method
    - https://github.com/psf/requests/blob/main/requests/api.py#L73
- Finally our response data is returned back to line 44 of controller/pages.py where we convert it to json.
    - https://github.com/ajmanzel/AllGasNoBrakes/blob/71d7c71275a6044dd2b19dde5fcc15baf3767e26/src/controller/pages.py#L44