**Technology used:**

Django
https://www.djangoproject.com/

**What it accomplishes:**

A. Establishing TCP server
B. Handling HTTP requests
C. Building web pages
D. HTTP responses and HTML templating
E. Interacting with a database

**How it accomplishes the above:**

A. By default, the runserver command starts the development server on the internal IP at port 8000.
    - The runserver command is accessed from the *manage.py* file generated when a django project is created. (A django project is created through running the "django-admin startproject <filename>" command).
        - The project is generated through *management/templates.py*
        - To run the server, one needs to type into the terminal "python manage.py runserver" or "python3 manage.py runserver". *manage.py* commands are facilitated through the *management/base.py* file.
    - *runserver.py* sets up the TCP socket server and specifies the port to connect to. Receiving of request data is facilitated through the *ServerHandler* class in *basehttp.py*, where buffering and request data recording is handled through the *LimitedStream* class in *handlers/wsgi.py*
    - Upon adding of Django Channels, a new key is added in our main folder *django_project*'s *settings.py*, called ASGI_APPLICATION, setting it equal to a variable in our *routing.py* file. The use of the *settings.py* folder is facilitated through *manage.py-tpl*.

B. Any request to the Django application is checked in the urls.py file and the pattern for the requested url is matched.

    - If the URL is found in the file a corresponding function to the same URL is rendered and the execution goes to the controller (views.py). The method is then executed and a response is generated which can be a simple HTTP response or any HTML template.

- In case the URL is not matched, it creates a not found error which will redirect to a 404 page.
- *basehttp.py* holds the *WSGIRequestHandler* class which parses the HTTP request header

C. In Django, each page is facilitated by separate folders within the directory. For example, the login page is facilitated by the "login" folder. The main folder that coordinates the pages is called *django_project*, which has references to all the page folders.
   - These folders are created through django's *handle()* method in *startapp.py*, which uses functions from management/*templates.py* to create a folder of a specified name with a migrations folder, *admin.py*, *apps.py*, *models.py*, *tests.py*, *urls.py*, and *views.py*.
     - *startapp.py*
     - *management/templates.py*
   - The paths that lead to these pages are specified in the *django_project* folder in *settings.py* in the list called *INSTALLED_APPS*. Each page folder has an *apps.py*, where a config class is contained. A string referencing the config class is what is added to *INSTALLED_APPS*.
   - In the *django_project* folder, there is also a python file called *urls.py*, which links specific paths to each page folder's respective *urls.py*.
     - In a page folder's *urls.py*, it looks at any possible paths, and executes a function depending on the path. The functions that are executed are located in the page folder's *views.py*. Usually, these functions serve the purpose of rendering the page.
   - A rendered HTTP response is sent with django's *render()* function in *shortcuts.py*. Refer to D to see how the function works.

D. The HTTP response is built with django's *render()* function. The return value of *render()* is used as the response. *render()* would be called in various functions within a page folder's *views.py*. For our purposes, we put in three arguments within *render()*. The first argument is the HTTP request sent by the client, the second argument is the HTML file we want to render, and the third argument is any data that needs to be rendered in the HTML file through templating.
   - The *render()* function in *shortcuts.py* first creates the body by calling *render_to_string()* within *loader.py*, which gets the specific templating engine used in *engine.py*. Within *engine.py*, string parsing of the HTML template is done through functions within *template/base.py*.
   - After the body is done, the body is passed into *HTTPResponse()* within *response.py* to prepend the header to the body.
   - The HTML templating uses a similar template language to Jinja. In the HTML file, text in double braces {{ }} signals a variable. Text within {% %} signifies a code block (such as a loop or conditional). And each block has an end block

notification to signify the end of a code block (Example: for loops in {% endfor %}).
        - For lists and dictionaries, keys and indexes use dot notation (Example: dict.word for finding the value of the key “word” in the dictionary “dict”; list.0 for finding the element at index 0).
    - When using Django’s templating, dangerous characters that might lead to HTML injections are automatically escaped. The character escape functionality is found in *utils/html.py* in the *escape()* function.
    - Cookies are set through *http/response.py* through the *http/cookie.py* import

E. Django uses models to access the database. Django’s models provide an Object-relational Mapping (ORM) to the underlying database, for our case, MySQL.
    - A model contains the fields and behaviors of the data stored, whose functions are found in *db/models/*.
        - A database is created through *mysql/creation.py*
        - Each model is a Python class that subclasses django.db.models.Model
        - Each attribute of the model represents a database field
            - Default database fields are found and validated in *db/models/fields/__inity__.py*.
            - Attaching of models to the database happens in *db/models/base/py*.
    - Django automatically gives a database-abstraction API that lets you create, retrieve, update and delete objects.
        - Creating objects, if new, happens with save() found in *db/models/base/py*. If an object already exists, save() will update it in the database.
        - After objects are created, Django adds a manager to each object with the _prepare() function found in *db/models/base/py*.
        - Managers can be used to directly access an object with the . (dot) operator
    - Migrations are Django’s way of propagating changes to models into the database schema, found in *db/migrations/*.
        - The migration happens in the handle() function in *management/commands/migrate.py*.
    - Regarding security: Django’s querysets are protected from SQL injection since their queries are constructed using query parameterization found in *db/models/query.py*. A query’s SQL code is defined separately from the query’s parameters. Since parameters may be user-provided and therefore unsafe, they are escaped by the underlying database driver.

**Licenses / TOS:**

The License of Django is very straightforward and only needs to meet three criteria:

1. Redistributions of source code must maintain the copyright notice.
2. Redistributions in binary must maintain the copyright notice in documentation.
3. Neither the name Django nor its contributors may be used to endorse or promote products derived from the software.

For our project, all three of the above are met and will be maintained.

*License found here.*