Politecnico di Torino

Master's Degree of ICT for Smart Socities

Graduation Session November 2025

Designing and Evaluating a Patient-Centred Web Platform for Pre- and Post-Treatment Cancer Care

Supervisors

Prof.michela MEO

Prof.guido PAGANA

Candidate

elaheh LOTFIMAHYARI

Abstract

This project demonstrates the architecture and implementation of a web-based platform intended for bowel cancer patients undergoing a prehabilitation program before surgical intervention. The regimen, which is typically five weeks long, encompasses various exercises and methods to enhance the patient's flexibility, muscle strength, balance, breathing, and stoma, along with diet and stress management segments. The doctors have to keep track of the patients' progress very closely during this time, but it is quite a tedious and challenging task to be done on paper. The application in question makes the work of the medical staff much easier by giving patients the opportunity to record their daily and weekly activities in secure online forms. They can note down their exercises, diet, and general health through a few simple clicks, and also go through the preoperative surgery checklists, which are given in a very lucid manner. Physicians are allowed to log in to a dashboard to get an overview of the patient's progress, see the level of adherence to the program, and give the necessary feedback in time. Moreover, the system provides an option for the export of patient data into Excel files, easing the processes of data analysis and storage for medical professionals. The integration of confidential data access, patient support, and non-stop supervision is what this assignment is all about, anticipating the prehabilitation period of five weeks to be a walk in the park for the patients and a breeze in the management for doctors, making it safe, efficient, and leading to a quicker recovery process in the vicinity of surgery. Moreover, this app is the perfect example of how prehabilitation can be the very first step in a patient's journey of cancer care. By the help of carefully planned exercises and daily monitoring done in a structured manner, patients get the drive to be physically ready on their own, as this has been demonstrated to make the recovery after surgery faster. In addition, the portal makes a more frequent and detailed follow-up possible at the time of rehabilitation, thus the chances of those patients who may be the victims of a relapse of neoplasia going unnoticed are lowered.

Summary

The project was born on the principle of the simplest thing possible: develop a functioning web application useful for something. My primary objective right from the start was gathering disparate bits of technology—such as Python, Django, HTML, CSS, JavaScript, and SQLite—and getting them all to coexist as an integrated whole. It was loose ideas about tools at first, but then, step by step, building them into an application with an evident design, robust backend, and nice UI.

The platform is designed to be with the patients at the cancer care journey, which is the most challenging to any human beings, starting from prehabilitation before operation, going through the perioperative period and finally, rehabilitation and follow-up. Hence, the app facilitates the patient's participation in the recovery process which is one of the keys to a better postoperative outcome and also ensures that the patient is not totally left without clinical supervision in the long run.

It all started as research and planning. Before jumping into coding, I investigated the available systems and referred to how others had handled the similar issues. It gave me an understanding of where it worked well, where it didn't work at all, and where it could improve. I also learned about the technologies I had to utilize. Some of them, such as HTML and CSS, I had learned before, but others, such as the structure of Django and database manipulation, required more learning. Each new thing learned acted as the basis for the system I was set to develop.

Actually the hard part was integrating all those components. Backend technology being Django created the heart of the system. It took care of the database using models, data management using views, and logically linking it all together. The actual database managed using SQLite was basic but efficient in storing the data the application required for it to run smoothly. Simultaneously, the frontend was user-orientated. Employing the use of HTML, CSS, Bootstrap, and some JavaScript, I endeavored to make the interface clear, clean, responsive so that anyone who used it could quickly point it out.

As the project expanded, I started getting an understanding of how integrated the frontend and backend actually are. The frontend is superficial, where people see it and use it ,but only as powerful as the supporting backend. In just the same

way, the backend gets to deposit and manipulate dat,a but without an effective interfa,ce users could not benefit from its potential. It was this interlink between design and functionality where the most important learning for me was gained when carrying this project through.

During the process, I also had an eye for documenting. Screenshots, diagrams, and pieces of code became the way not only to monitor the progress but also to justify the decisions. It may sound as an additional activity, but as it turned out quickly, it turns out to be highly valuable when others may use or develop the system further. It's the way of leaving the map behind not just for others but for myself too, in the event I return to doing this project at some point in the future.

Working on completing the system was both technical and personal. Technically, I built an application where by the web application standards it was functional, stable, and easy to use. It had data handled smoothly in the backend, presented clear interfaces on the frontend, and had the right database design supporting the performance of the system. Personally it taught me how much I could absorb just by going on some actual project and sticking at it when it was complex or intimidating. Looking to the future, I realize there are numerous ways this system could become better or larger. It could add features, it could make the interface more interactive still, it could make the database stronger so it could deal with larger and more involved data sets. It could also improve security as well as performance, particularly if it was going to be scaled up. Even compatibility on the small device level as well as further automation could make it stronger. In some ways, this project seems just right as an initial version—you could definitely run it but it also leaves something for the future for creativity as well as expansion.

Ultimately, it isn't just about creating the web application. It's about the process of starting at an idea, learning the technology, overcoming the challenges, and watching it all coalesce into an operational system. It demonstrates both the technical ability I accomplished as well as the personal growth I achieved during the process. Most importantly, it provides me the assurance I could attempt projects of the similar nature in the future and continue getting better as a developer.

Acknowledgments

I would like to express my deepest gratitude to my supervisors, Prof. Michela Meo and Prof. Guido Pagana, for their valuable guidance, continuous support, and encouragement throughout the development of this thesis.

My sincere thanks go to Dr. Serena Perotti for her insightful feedback and clinical guidance, which significantly contributed to the improvement of this work.

I am also especially grateful to my family and my daughter Rasta for their unconditional love, patience, and support during this journey. Without their presence and motivation, this accomplishment would not have been possible.

Finally, I would like to thank all the friends and colleagues who supported me both technically and emotionally, making this experience meaningful and memorable.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

When someone is told they have cancer, life suddenly fills up with hospital visits, new instructions, and endless bits of paper to remember. One day it's a scan, the next it's blood tests, and in between there are phone calls and notes from the doctor that are easy to misplace. It can feel like a full-time job just trying to stay on top of everything.

Now imagine having one simple place on your phone or computer where all of this lives: your calendar, your doctor's notes, and even a personal space to jot down how you're feeling or what you've noticed about your body. No more sticky notes on the fridge or lost appointment cards. Patients know exactly what's coming next, and doctors can instantly see the updates patients share. It makes the whole process smoother, less stressful, and helps everyone stay connected and on track.

The aim of the project is to provide the clinical pathway in oncology with a full range of support starting from prehabilitation, through perioperative care, and further into rehabilitation and long-term follow-up. Ultimately, the platform wants to be a factor in speeding up recovery and lowering the co-morbidity risk through the patient becoming more actively engaged in preoperative preparation and, at the same time, by more intensive postoperative monitoring being enabled.

1.2 Problem Statement

Managing cancer care information is still a challenge in many places. Paper notes, printed schedules, and verbal instructions are easily misplaced or misunderstood, and updates often arrive too late. While some hospitals rely on electronic health record (EHR) systems, these platforms are not always designed with patients in mind—especially those receiving treatment outside the hospital or in community care, where quick and simple access is crucial.

Patients need a tool that feels easy to use and secure, one that lets them:

- Create an account and log in safely.
- Find clear instructions for before and after their treatments.
- Upload documents or complete forms within given deadlines.
- View their appointments, locations, and necessary materials through a calendar.

On the other side, doctors need a way to quickly check and organize the information their patients provide. Existing systems rarely combine these needs with features like generating structured summaries or exporting patient history into formats such as Excel, which are helpful for follow-ups, research, or record keeping. At the same time, any solution has to remain easy to navigate, accessible to different users, and compliant with privacy rules such as the GDPR.

1.3 Research Questions

This project is shaped by a few central questions that guided the design and development process:

- **RQ1:** How can a web-based application be built to support cancer patients in keeping track of tasks, filling out forms, and managing appointments before and after treatment?
- **RQ2:** What design choices and features can make the platform simple for patients to use while also giving doctors an efficient way to review the information provided?
- **RQ3:** To what degree does the system succeed in meeting the needs of usability, accessibility, and data security within a healthcare setting?
- **RQ4:** How can the platform include an export option (Excel sheets) so that doctors can easily access a clear record of patient submissions?

1.4 Contributions

The main contributions of this work can be summarized as follows:

- **Requirements Gathering:** Identifying the needs of both patients and doctors for a shared platform that can support cancer care routines.
- **System Development:** Building a working web application with Django as the backend and HTML/CSS for the frontend. The platform includes patient registration, secure login, interactive forms, a digital notebook for materials, calendar integration, and visit information.
- **Doctor's Dashboard:** Developing a dedicated interface for doctors to view patient histories, review forms, and generate structured Excel exports of

patient submissions.
- **Evaluation:** Testing the platform with respect to ease of use and accessibility, while checking that it aligns with data protection and web accessibility guidelines (such as GDPR and WCAG 2.2).

1.5 Scope and Boundaries

This thesis focuses on creating and testing a digital tool that helps organize and support cancer care.

Scope

The system includes:
- Patient registration and login.
- Interactive forms with time-based submissions.
- A calendar showing tasks, appointments, and required materials.
- Directions and location details for upcoming visits.
- A dashboard for doctors to review and manage patient submissions.
- Export of patient data into Excel for clear, structured reports.

Boundaries

The project also has clear limitations:
- The system does not provide diagnosis or treatment tools.
- All information is supportive and organizational only.
- No real patient data is used in testing; instead, artificial data is applied to respect privacy laws.

Chapter 2

Background and Related Work

2.1 Cancer Care Pathways

Going through cancer treatment is not a single event but a long and complicated journey that usually unfolds in three main phases: finding out about the disease, undergoing treatment, and then trying to heal and adjust after treatment is completed. Each of these steps creates its own pressures and difficulties, not only for the patient but also for family members and caregivers who walk alongside them.

The first stage, diagnosis, is often described as one of the most stressful moments of the entire cancer pathway. Patients are asked to go through a series of medical examinations—such as blood work, biopsies, ultrasounds, CT scans, or MRIs—that can feel overwhelming. The sheer number of appointments, along with instructions that are sometimes confusing or provided by different doctors in separate settings, can leave people uncertain about what they are supposed to do next. This uncertainty frequently heightens fear and anxiety, as described by the World Health Organization, which has emphasized how fragmented communication in healthcare increases the emotional toll on patients [1].

When the treatment stage begins, the challenges take on a new dimension. Depending on the type and stage of cancer, patients may face surgery, chemotherapy, radiation therapy, targeted drugs, or a combination of these. Each of these comes with its own strict schedule and unique side effects. For example, chemotherapy may cause fatigue, nausea, or hair loss, while radiation might lead to burns or localized pain. Patients must keep track of medications, attend appointments on time, and stay aware of possible complications. Studies such as Richards et al. (2020) have shown that patients who lack proper guidance during this stage often struggle with sticking to their treatment schedules and, in turn, experience worse outcomes [2].

The third stage—life after treatment—is often misunderstood. Many people think finishing chemotherapy or surgery means the end of the journey, but in reality, recovery brings its own set of difficulties. Survivors often have to manage scars, long-term side effects, or even disabilities caused by surgery. They also need to maintain strict routines for monitoring symptoms, taking prescribed drugs, and attending follow-up appointments that can stretch over several years. In addition, emotional health becomes central: survivors frequently report fear of recurrence, feelings of isolation, and the need for psychological counseling or community support [3]. For many, adjusting daily habits such as diet, physical activity, and sleep becomes a lifelong task.

Across all these stages, a recurring problem is that information is rarely delivered in a way that is clear, consistent, and easy to follow. When patients receive scattered instructions, are overloaded with technical jargon, or feel left alone to manage complex tasks, the results are almost always negative. They report more stress, they miss doses or appointments, and their overall quality of life drops. Research highlights that well-organized communication and patient-friendly tools can make a measurable difference. For example, structured guidance improves treatment adherence, reduces anxiety, and empowers patients to take a more active role in their care [3, 1].

This has led many researchers and clinicians to stress the importance of practical solutions that help patients navigate these care pathways more smoothly. One promising direction is the use of digital applications and web-based systems. These tools can gather all instructions in one place, send reminders about medication or appointments, and provide educational material in clear, simple language. By doing so, they can act almost like a digital assistant, reducing the feeling of being lost in a maze of procedures and forms. Studies on real-time digital monitoring systems, such as those by Richards et al. (2020), have shown encouraging results, especially in supporting patients during and after treatment [2].

In short, the cancer care pathway is not only a medical journey but also an emotional and organizational challenge. Diagnosis demands clarity, treatment requires structure and support, and post-treatment recovery calls for continuous monitoring and psychological care. Without tools that simplify this experience, patients are left vulnerable. With the right support, however, the journey becomes more manageable, offering patients and caregivers a stronger sense of control and dignity throughout the process.

2.2 Digital Health Platforms in Cancer Care

The use of digital health technologies has transformed patient management in healthcare, particularly in oncology. One such system, MyChart, has played a pivotal role in facilitating communication between patients and healthcare providers. These systems allow patients to access their health records, make appointment requests, and receive test results remotely, fostering convenience and patient engagement [4].

Similarly, mobile applications such as Carevive offer personalized care for cancer patients. Carevive is integrated with Electronic Health Records (EHRs) to offer personalized care plans, symptom management, and education, aiming to improve patient outcomes and make care coordination easier [5].

Despite these advances, there are many challenges. The majority of EHR systems are not designed to allow seamless data exchange, limiting the ability to export structured reports like Excel spreadsheets that summarize patient histories. This lack of interoperability could undermine comprehensive care and follow-up visits [6].

Moreover, while digital platforms enhance communication, they do not encompass the entire process of cancer care. Gaps exist in both pre-treatment preparation and post-treatment follow-up, where digital tools can play a significant role in providing sustained support and information for patients [7].

2.3 Usability and Accessibility in Digital Health

When it comes to digital health platforms, the design can really matter a lot in terms of how beneficial they are to the patient. Usability is about designing a system in a simple and intuitive way so that the user can get things done without ending up frustrated or confused. For instance, Nielsen's usability heuristics suggest that the system should clearly indicate what is happening, act as people expect things to in the real world, and give users a sense of control over what they are doing [8]. Researchers use instruments like the System Usability Scale (SUS) to measure how easy healthcare apps are to use and identify areas where they need to improve [9].

Accessibility is also crucial, particularly for patient populations who might have specific requirements, such as older citizens or those who are disabled. Guidelines like the Web Content Accessibility Guidelines (WCAG 2.2) help in making web pages and applications easy to perceive, operate, and understand, and easy to use across different devices [10]. Practical implementations for facilitation of accessibility include support for screen readers, keyboard navigation with clear controls, high contrast color schemes, and plain language that is easy to understand. All this notwithstanding, though, most mobile health apps and patient portals are inaccessible and even some people cannot use key functions [**moncy**].

At the level of daily practice, developers and designers need to pay close attention to usability and accessibility issues at all stages of creating a digital health tool. Usability testing may include observing patients utilizing the platform or asking them questions to find out how simple things are. Accessibility testing may include testing the app with screen readers, keyboard-only navigation, or color-blindness simulators to ensure that all users can use the platform effectively. By combining usability and accessibility considerations, online health platforms are more inclusive and enable patients to manage their care confidently and reduce the risk of mistakes or omission.

Others have demonstrated that, when applied, these principles can greatly enhance patient engagement. For instance, clear feedback and simple interface designs enable users to more effectively adhere to care instructions, whereas accessible content prevents vision or motor impairment from excluding patients from digital health programs. Health technology researchers more and more focus on co-design methods, in which patients themselves contribute to the development and improvement of applications in order to meet their requirements better [9, 10, 8].

Lastly, the goal of emphasizing usability and accessibility in digital health is to make healthcare more equitable and effective. It is not only unpleasant, but also unhealthy, to use systems that are hard to use or unavailable. Therefore, investing time and resources in testing, iterating design, and adhering to proven standards is not an option—it is necessary to create tools that actually work for all patients, particularly the most vulnerable ones [**moncy**].

2.4 Privacy and Security in Healthcare Data

Healthcare information is very sensitive and subject to strict privacy and security laws. In the European Union, conditions like informed consent, minimisation of data, transparency, and the right to be forgotten are outlined in the General Data Protection Regulation (GDPR) [11]. These conditions emphasize that digital platforms must adopt a "privacy-by-design" approach, i.e., security and privacy must be incorporated into the development phase from the beginning.

Technically, digital health platforms are vulnerable to a number of threats, i.e., weak authentication, injection attacks, and cross-site scripting. Standards like the OWASP Top Ten identify these threats [12]. Best practice for securing this is by encrypting data in transit and at rest, applying multi-factor authentication, employing regular updates, and practicing secure coding. Trust in digital health systems depends primarily on strong privacy and security practices.

2.5 Comparison of Existing Solutions

As one examines the platforms that currently exist for healthcare management, it is clear that while they provide some useful tools, they fail to fill in the gaps for patients or providers. Hospital portals, for example, generally allow patients to view their medical records, lab results, and treatment histories. However, they do not routinely provide structured support for pre-treatment preparation or patient aftercare following procedures [13]. Similarly, mobile health management apps often offer reminders for medication or exercise and some educational content, but are not incorporated into the task workflow of care, lessening their actual usefulness in daily patient management [14].

Accessibility and compliance with privacy legislation vary across platforms, with some systems not adequately protecting sensitive patient data and others not providing the functionality to export health data into standard formats such as Excel for follow-up or research [15]. These shortcomings pose a challenge for patients and providers to maintain comprehensive, easily transportable records of treatment activities and health outcomes.

The system presented in this thesis is intended to bridge these gaps by incorporating a number of essential functions into a single platform. In the first instance, it provides secure patient registration and authentication processes to avert unauthorized access to health data [16]. Second, it includes pre- and post-treatment checklists that guide the patient step by step through what should be done before and after receiving medical treatment. These checklists serve to promote compliance with suggested procedures and minimize the possibility of errors or omitted steps within treatment regimens.

In addition, the platform incorporates a patient notebook feature that allows individuals to record instructions, observations, and progress over time. The electronic notebook permits ongoing monitoring by patients and allows doctors to see previous entries for a more comprehensive overview of each patient's progress. Calendar integration provides an extra layer of support by allowing patients to manage appointments and daily care tasks efficiently.

From the provider's perspective, an administrative dashboard offers healthcare providers an overview of patient submissions, allowing them to monitor adherence to care plans and provide feedback in a timely fashion. Finally, an export function enables the generation of Excel files containing complete patient histories, making it easy to analyze trends, generate reports, or create structured data for follow-up care [17].

By consolidating these elements, the system here put forward presents patients with a comprehensible and traversable pathway through treatment and gives clinicians a coherent, easily interpretable collection of information. The platform emphasizes both accessibility and confidentiality, offering a secure portal that supports enhanced care coordination and results for cancer treatment and other complex medical routines [13, 14, 15, 16, 17].

Chapter 3

Technologies

3.1 Python Virtual Environment

Perhaps the most important tool in Python to ensure the smooth and predictable running of your programs is the *Virtual Environment.* A Virtual Environment is essentially a self-contained folder on your computer that has a particular version of Python with its own isolated package and module set. This separation is necessary because the majority of Python applications rely on packages not included in Python's standard library. If different projects require different versions of the same package, their global installation can quickly lead to conflicts and bugs.

As an example, consider two Python projects: Project A might need version 1.0 of a library, and Project B depends on version 2.0 of the same library. If both projects are installed globally, then either project can fail to run if the correct version isn't available. Virtual Environments solve this problem by allowing each project to have its own isolated set of packages. This makes it possible to have multiple projects on one computer without needing to worry about incompatibilities between their dependencies.

It's also convenient to use a Virtual Environment for collaboration and deployment. If a project has an isolated environment, it is easier to deploy to another machine or share with collaborators because dependencies are explicitly declared. Using tools like `pip`, you can install the same versions of packages declared in a file called `requirements.txt` so that everyone working on the project has the same setup. It would be difficult to recreate results or debug issues due to unmatched package versions without this isolation.

Figure 4.1 illustrates this idea: multiple virtual environments can exist on a single machine, each with its own Python interpreter and packages, allowing developers to safely run different applications simultaneously.

The concept of Virtual Environments has been standardized in Python with `venv`, which was introduced in Python 3.3, and is now best practice for any professional Python project [18]. There also exist additional resources and tutorials for learning how to properly use and understand Virtual Environments, such as Real Python's tutorial on *Python Virtual Environments: A Primer* [19], and W3Schools' reference for virtual environments in Python [20].

By using Virtual Environments on a routine basis, developers avoid exacerbating the "dependency hell," ensure compatibility between projects, and maintain a clean, well-structured Python development environment. [18, 19, 20]

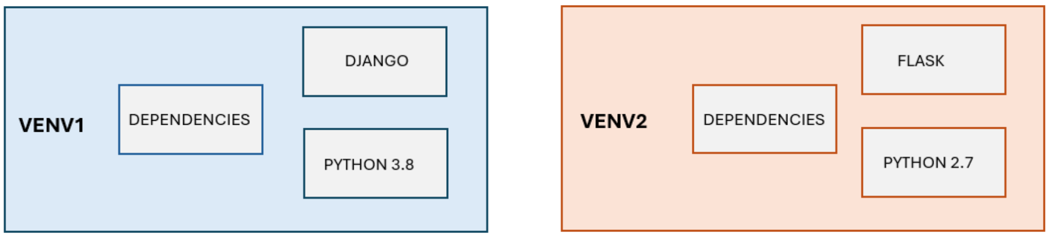


Figure 3.1: Virtual Environment with separate Application, dependencies, and Python version.

3.2 Django Framework

Django is a Python web framework with advanced features that support quicker development of secure and sustainable websites. It was originally developed by Adrian Holovaty and Simon Willison in 2003 and then released in 2005. Now its maintenance is done by the Django Software Foundation [21].

It is intended to simplify the creation of dynamic, database-driven websites. It facilitates reusability, rapid development, and the "don't repeat yourself" (DRY) principle. This enables developers to focus on application-specific logic instead of repeating the same items like common functionality [22].

Rapid Development: Django follows a "batteries-included" philosophy, which includes the following built-in features: an authentication system, an admin interface, and form handling. This allows for fast development from concept to deployment [23].

Security: Security is one of the key features of Django. It offers protection from frequent attacks like SQL injection, cross-site scripting (XSS), cross-site request forgery (CSRF), and clickjacking. This is particularly important in applications that handle sensitive information [23].

Scalability: Django supports multiple relational databases like PostgreSQL, MySQL, Oracle, and SQLite. It has an Object-Relational Mapper (ORM) that allows developers to interact with databases using Python code, thus making applications scalable based on increasing traffic and data [24].

Maintainability: The sound separation of concerns due to the Model-Template-View (MTV) architecture of the framework, coupled with compliance with DRY, makes it easier to maintain and expand Django applications in the long run [25].Figure 4.2

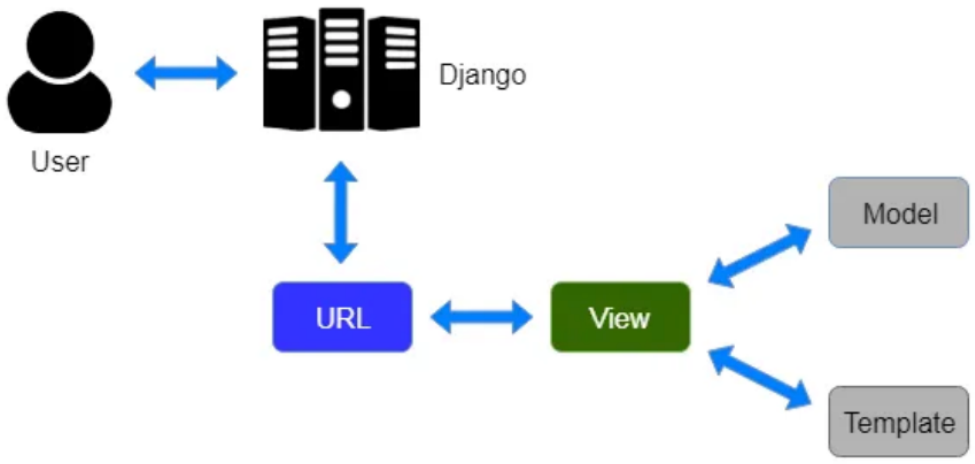


Figure 3.2: MTV Model.

Django employs the Model-Template-View (MTV) pattern, which is philosophically similar to Model-View-Controller (MVC). Under this structure:

- **Model:** Handles data and database schema (`models.py`).
- **Template:** Handles presentation and how to show data (`template.html`).
- **View:** Contains business logic and mediates models and templates (`views.py`).

This separation of concerns simplifies maintenance, updates, and scaling of web applications.Figure 4.3

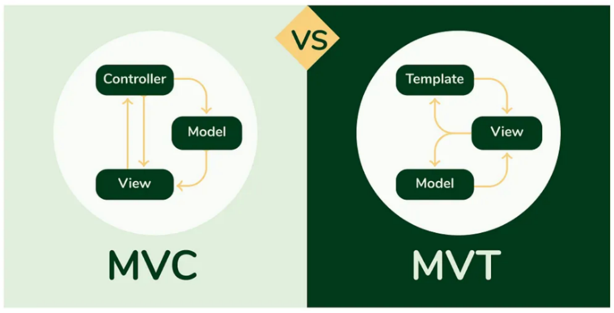


Figure 3.3: MVC VS MTV Model.

Django's design and capabilities make it a good candidate for creating applications that demand speed, security, and future-proof maintainability [21, 22, 23, 24, 25].

3.3 JavaScript

JavaScript used to be the non-interactive part of the web page, but now it is responsible for all the user actions and dynamic updates on the site. HTML structures a page, and styling is taken care of by CSS, but JavaScript is what makes the page react to user inputs and update itself without the user's intervention. Practically, this is the case when the user doesn't have to wait for a full reload in order to press a button, write on a form that is automatically validated, or other similar actions [26]. Basically, JavaScript talks to the Document Object Model (DOM), which is the normal tree-like structure that holds all the information about a web page. Thus, developers can, on the fly, manipulate texts, images, or links, or even complete page sections. A simple script can make sure a login form is correct or show a warning message without the need for the server's help, which is user-friendly [27]. To be more specific, developers are free to incorporate JavaScript code directly into HTML using the `<script>` tag, or better still, they can keep it in separate files which are more i... libraries like jQuery made the lives of developers easier by automating their tedious and repetitive tasks, and recently frameworks such as React or Vue.js have been providing the developers' with fast and efficient ways to build big scale, interactive applications [28]. JavaScript is basically the first to be standardized within the ECMAScript umbrella, which means that it is very stable and behaves quite similarly in different browsers. Most of the time, this standard is being updated, actually ends up bringing modern features, such as async/await, modules, and better object handling, to the language while trying to continue with all the previous versions [29]. In addition, it should also be mentioned that the presence of JavaScript is not merely restricted to the browsers any longer. Over the period of time, as a consequence of the great improv... Consequently, the applications written under the Node.js package model carry the nature of event-driven and are capable of handling thousands of connections effectively, which eventually leads to making the JavaScript technology one of the most versatile among the web development world that is still facing competition with the likes of [30].

Figure 4.4 shows how JavaScript connects the browser with the backend and database in a typical web system.

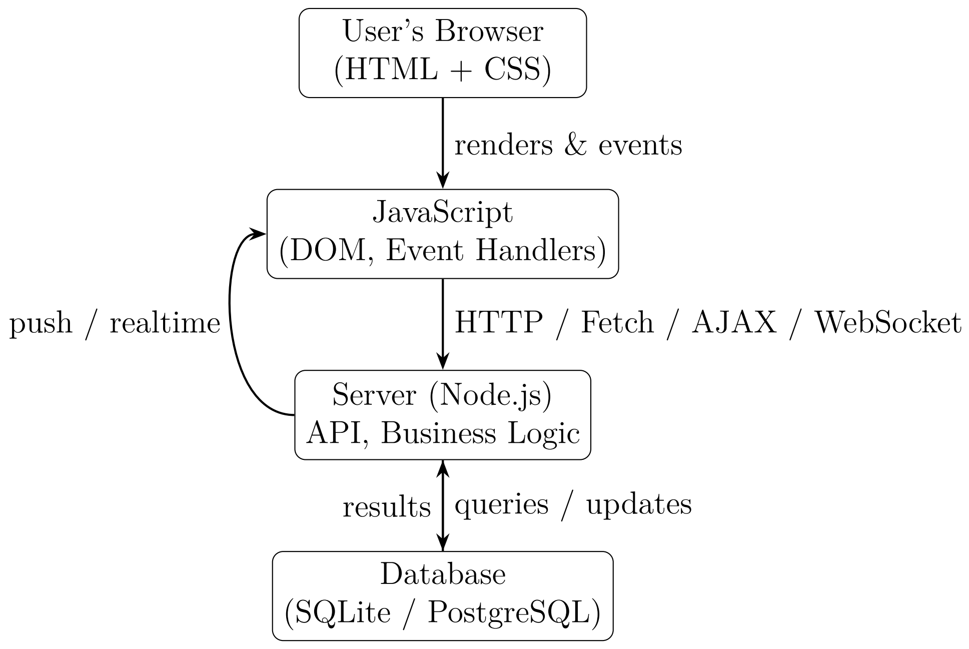

User's Browser (HTML + CSS)
renders & events
JavaScript (DOM, Event Handlers)
push / realtime
HTTP / Fetch / AJAX / WebSocket
Server (Node.js) API, Business Logic
results
queries / updates
Database (SQLite / PostgreSQL)

Figure 3.4: JavaScript in the browser and on the server (Node.js) working with a database.

3.4 HTML

HTML is the fundamental structure that underlies every webpage - you can consider it the frame that supports the entire application. Differently from programming languages, which carry out operations, HTML is a language that explains to the browsers how to organize the content visually. It establishes the properties and characteristics of titles, text, and images, and aside from these basic elements, it also introduces forms and links into the page. These elements are the smallest parts of the web's visible structure. By saying `<p>This is a paragraph</p>` it's like instructing a browser to display the text within the tags as a paragraph [31]. A progressive measure was the development of **HTML5**, which changed the tags into clearer and more meaningful ones called *semantic elements.* It contained such items as `<header>`, `<nav>`, `<article>`, `<section>`, and `<footer>`. Now, apart from machines, even people who look at the structure of a page can get an idea much faster. Those that are considered neither structural nor semantic are only tags and thus do not help human readers and programmatic ones, like screen readers, gain a better understanding of the page. The mentioned issue can be solved not only with the help of developers but also with the help of search engines and accessibility tools such as screen readers that change `<div>` tags into something more meaningful and logical [32, 33]. Usage of semantic HTML has a direct influence on accessibility. In a situation where a webpage is properly tagged, for instance, a user with the help of a screen reader can selectively listen to headings in order to find the required one quickly and thus save the time that would be spent in linear reading of the whole page. Along with inserting descriptions in images and labels in a form, users with visual impairments become another group of people who gain equal access to the webpage by ensuring that they can indeed use it. The accessibility criteria, such as WCAG 2.1, provide the best practices of the usage of semantic and well-structured HTML, which are the major factors in making web content accessible to a wide range of users [34, 35]. HTML in general is predictable and has a wide application that makes it a very valuable tool from a practical perspective. Any web browser on any device can understand it, and no additional software is needed. The developers can first create a basic page and then use CSS and JavaScript to make the page more attractive and interactive. The usage of semantic HTML helped the template retain its ease of maintenance, enabled a neat CSS and Django template integration, and facilitated the export of medical content into structured outputs in this thesis project.

Figure 4.5 shows a simplified diagram of how a semantic HTML document is structured. This clear separation of regions makes it easier to maintain templates and improves accessibility.

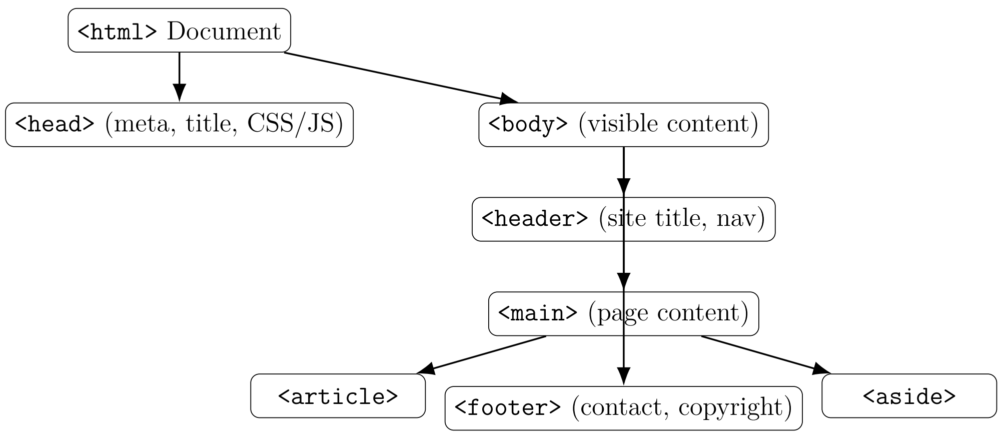

<html> Document
<head> (meta, title, CSS/JS)
<body> (visible content)
<header> (site title, nav)
<main> (page content)
<article>
<footer> (contact, copyright)
<aside>

Figure 3.5: Typical HTML document structure with semantic regions.

3.5 Bootstrap CSS

CSS frameworks are toolkits that provide developers with premade styles and layout rules to cut down the time significantly. One of the most popular and well-known frameworks is **Bootstrap** among others. Twitter engineers Mark Otto and Jacob Thornton created it in 2011 to be the result of front-end development quicker and more uniform across different projects [36]. Since then, it has transformed into a worldwide open-source project with user and contributor communities in almost every country [37]. Because of the built-in *grid system*, which is the main feature of Bootstrap and allows for the automatic adjustment of the layout for the various screen sizes, designing modern web applications has become much simpler with Bootstrap. This basically means that any website created using Bootstrap will still be visually appealing on a desktop monitor, a tablet, or a smartphone, and there won't be a need to write separate CSS rules for each device by the developers. In reality, this accomplishment saves an enormous amount of time and also reduces the number of mistakes when working with the responsive design [38]. The *cross-browser support* is another important aspect of Bootstrap as well as *cross-browser support*. Developers can be quite sure that their websites will be operational on Chrome, Firefox, Safari, and even on older versions of Internet Explorer with a few adjustments. Besides that, the framework comes with a collection of pre-styled components such as buttons, navigation bars, modals, and forms, which can be customized as needed. This equips the developers with a secure starting point as well as the necessary malleability to come up with one-of-a-kind designs. The **community around Bootstrap** has been a major factor in its success over time. A large number of themes, templates, and UI components are available without any charges, which in turn helps both beginners and professionals to create attractive websites in less time. The official documentation is reportedly one of the best in the area, as it provides examples, code snippets, and explanations that facilitate even newcomers to get started easily [39]. In general, Bootstrap is still considered a developer-friendly tool, as it combines the advantages of being user-friendly and being versatile. Regardless of the objective being a personal website, a business platform, or a healthcare management system, Bootstrap makes the process of rapid prototyping and delivering the final product, which is professional and aesthetically pleasing, possible.

3.5.1 SQLite3

SQLite is a small, convenient database that is integrated inside the app instead of a separate server process. As a result, the entire database engine is just a library that you link with, and it works without any installation or running a server. For daily development and small deployments, it is very convenient with the absence of a daemon to configure, a user to create, and a network connection [40, 41]. The especially great thing about SQLite is that the entire database is a single file on the disk. The single-file format makes it very easy to copy, back up, or transfer the database between different computers — you can, for instance, send the file through email or check it into a project folder. That portability is one of the main factors why SQLite developers use it when making prototypes, embedded apps, and tools where the ease of sharing is important. Besides, the format is also quite stable and widely adopted [42, 43]. Being a lightweight and zero-configuration program, SQLite is definitely the first choice for development, small web projects, single-user desktop apps, and mobile apps. It is usually installed as a local cache or staging store in front of a larger system before the data moves there. In short, if you are looking for a reliable and fast database without the trouble of handling a separate server, SQLite should be the one to go [44, 41]. However, there are disadvantages as well. On the one hand, SQLite can perform numerous simultaneous reads without any problem; on the other hand, the writes are serialized. That is to say, the writer will momentarily lock the database, redirecting it to a new location while updating the file. Normally, a client–server DB (e.g., PostgreSQL or MySQL) is what you would choose for multi-user, high-concurrency server applications. Nevertheless, for the types of operations in this project that involve writing patient daily notes and activity logs, exporting history to Excel, SQLite's simplicity, portability, and ACID guarantees are the factors that make it a very good solution [45, 43].

The graph shown in Figure 4.6 shows the differences between SQLite and the typical client-server databases.

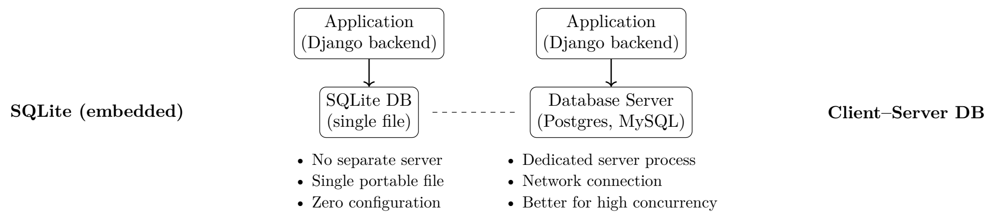

SQLite (embedded)
Application (Django backend)
SQLite DB (single file)
• No separate server
• Single portable file
• Zero configuration
Application (Django backend)
Database Server (Postgres, MySQL)
• Dedicated server process
• Network connection
• Better for high concurrency
Client–Server DB

Figure 3.6: Simple comparison: embedded SQLite vs client–server databases.

Chapter 4

Methodology

This project is a web application developed with the Django framework. Synchronously and asynchronously, the application can work, and it is still suitable for previous Python environments. The method is a gradual development process. It fuses the frontend, database, and backend parts, offering main code excerpts to demonstrate the implementation options.

4.1 User Interface and Web Views

The initial phase for building this online application was, without a doubt, the design of a user-friendly interface that would facilitate the users' interaction with the system. The UI was designed with HTML, CSS, and Django templates. This enabled it to be both responsive and accessible. The essence of the interface is to allow users to perform registration, log in, pick up activities, record work, and check the instructional content through an easy flow. The several pages of the interface represent different functions or user tasks. The essential parts of the UI are briefly explained below:

1. **User Registration and Login:** Users can create accounts, sign in, and access personalized content. The registration form includes:

   - **Username:** A unique identifier for the user.
   - **Password:** To secure the account.
   - **patient info:** contacts, background health of patient.

   Returning users can log in using their credentials.

2. **Activity Logging Page:** Users can select daily activities, log their completion, and add notes. Predefined activities include:

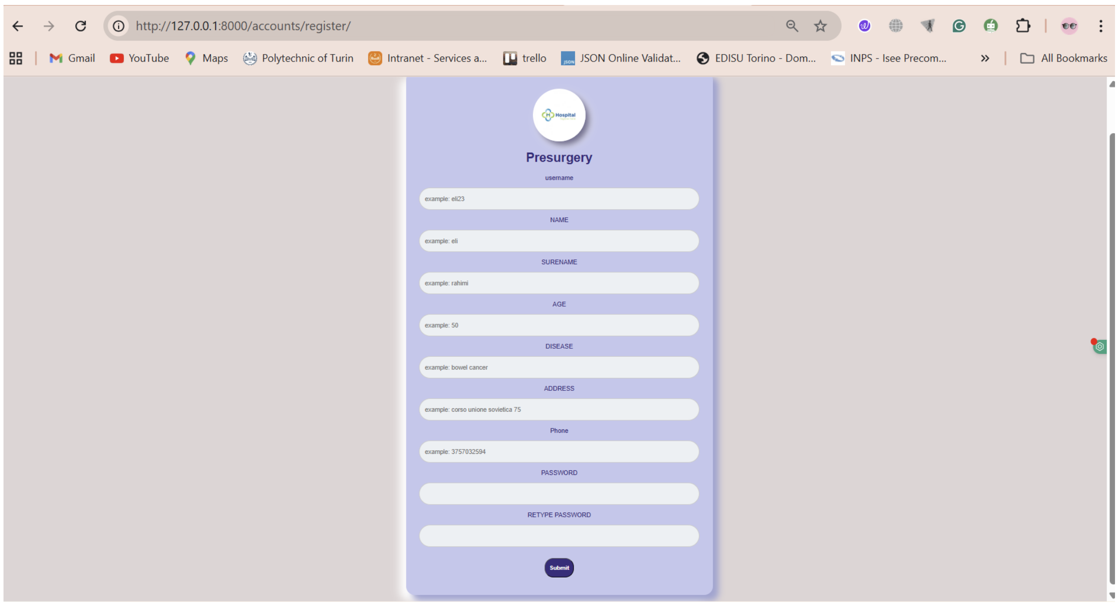
- Exercise
- relaxation
- Meal

Users can also select the date of the activity and add optional notes for additional details.

3. **Instructional Content Page:** This page provides multimedia content to guide users in performing activities. Based on user selection, it can display: Pictures illustrating each step

4.1.1 User Registration Form

Django forms and templates have been used to create the registration page. The registration HTML form is shown in the figure 4.1, 4.2 Input fields are validated on the client-side (HTML required) and server-side with Django forms. The form data is handed to a Django view for handling the user save operation in the database. There is a separate login page through which users can log in after registration



Presurgery

username
example: eli23

NAME
example: eli

SURENAME
example: rahimi

AGE
example: 50

DISEASE
example: bowel cancer

ADDRESS
example: corso unione sovietica 75

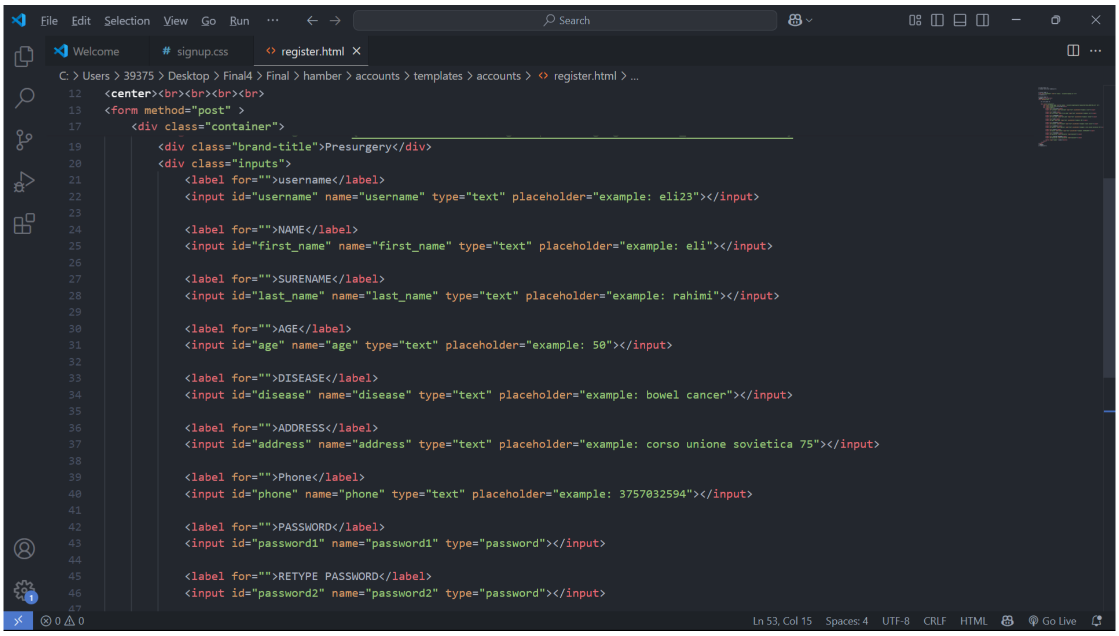
Phone
example: 3757032594

PASSWORD

RETYPE PASSWORD

Submit

Figure 4.1: Registration Page UI.

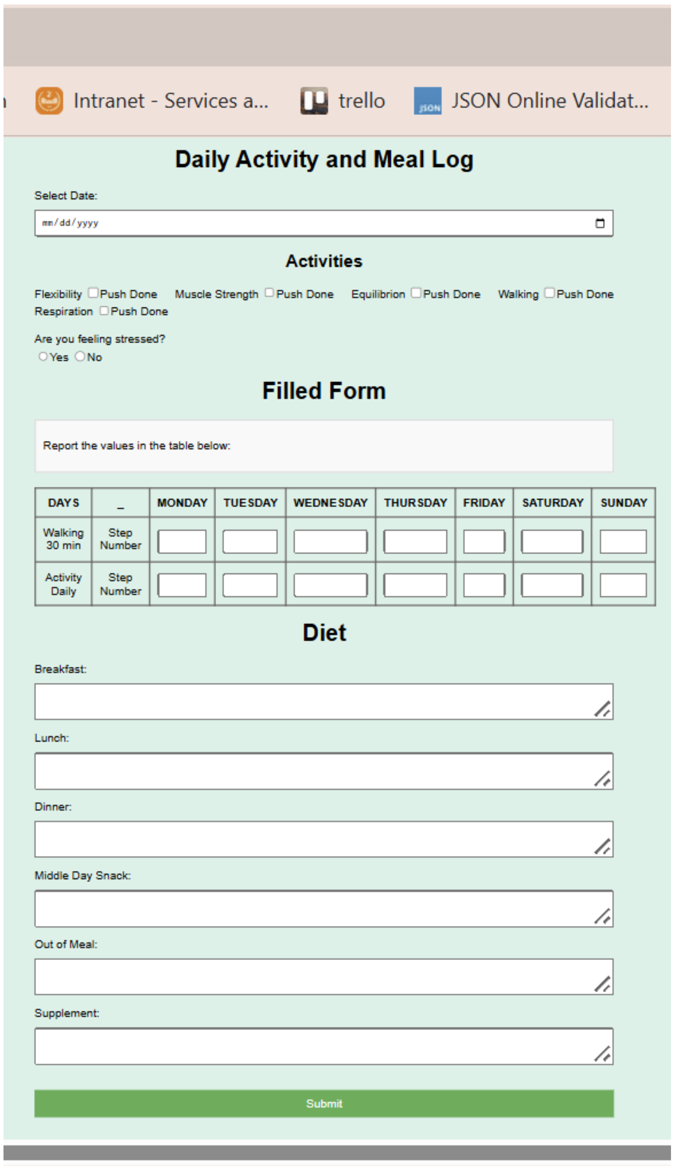

File
Edit
Selection
View
Go
Run
Search
Welcome
signup.css
register.html
C: > Users > 39375 > Desktop > Final4 > Final > hamber > accounts > templates > accounts > register.html > ...
Ln 53, Col 15
Spaces: 4
UTF-8
CRLF
HTML
Go Live

```html
<center><br><br><br><br>
<form method="post" >
    <div class="container">
        <div class="brand-title">Presurgery</div>
        <div class="inputs">
            <label for="">username</label>
            <input id="username" name="username" type="text" placeholder="example: eli23"></input>

            <label for="">NAME</label>
            <input id="first_name" name="first_name" type="text" placeholder="example: eli"></input>

            <label for="">SURENAME</label>
            <input id="last_name" name="last_name" type="text" placeholder="example: rahimi"></input>

            <label for="">AGE</label>
            <input id="age" name="age" type="text" placeholder="example: 50"></input>

            <label for="">DISEASE</label>
            <input id="disease" name="disease" type="text" placeholder="example: bowel cancer"></input>

            <label for="">ADDRESS</label>
            <input id="address" name="address" type="text" placeholder="example: corso unione sovietica 75"></input>

            <label for="">Phone</label>
            <input id="phone" name="phone" type="text" placeholder="example: 3757032594"></input>

            <label for="">PASSWORD</label>
            <input id="password1" name="password1" type="password"></input>

            <label for="">RETYPE PASSWORD</label>
            <input id="password2" name="password2" type="password"></input>
```

Figure 4.2: Registration HTML Code.

4.1.2 Activity Logging Interface

An activity logging page enables users to record their daily activities. Users pick the date and what they have done. Notes provide additional details for an activity, for instance, length, intensity, or food consumed. A form submission turns on a Django view, which validates the input and, with the help of the Activity model, saves it in the database. Figure 4.3



Daily Activity and Meal Log

Select Date:

mm/dd/yyyy

Activities

Flexibility ☐ Push Done Muscle Strength ☐ Push Done Equilibrion ☐ Push Done Walking ☐ Push Done
Respiration ☐ Push Done

Are you feeling stressed?
○ Yes ○ No

Filled Form

Report the values in the table below:

DAYS	_	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY
Walking 30 min	Step Number							
Activity Daily	Step Number							

Diet

Breakfast:

Lunch:

Dinner:

Middle Day Snack:

Out of Meal:

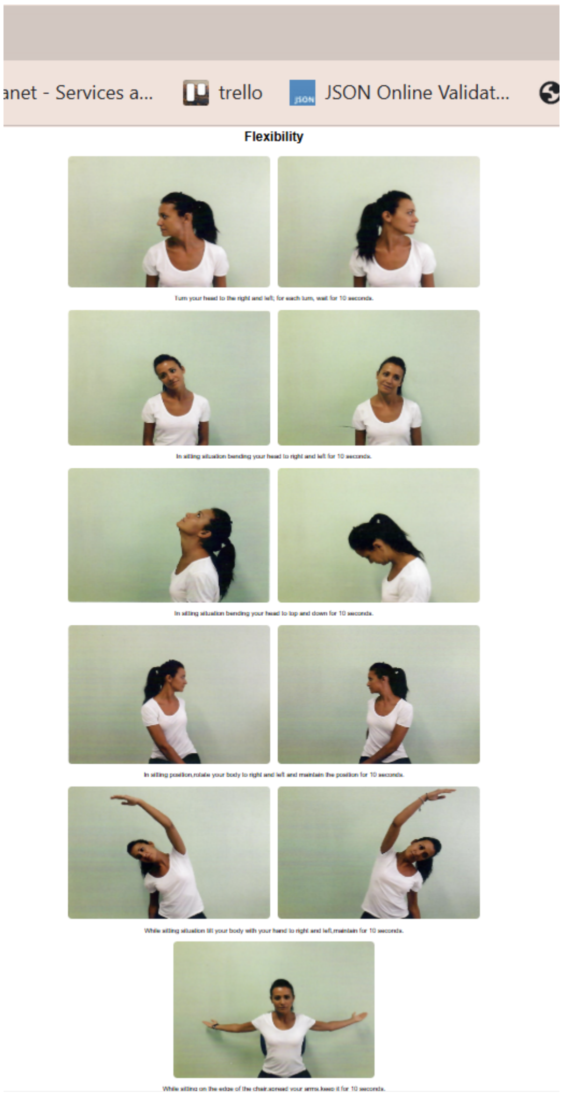
Supplement:

Submit

Figure 4.3: Filling up Form.

4.1.3 Dynamic Instructional Content

Users of the home page can view step-by-step instructional materials that guide them through the correct execution of a task. The content is dynamically loaded based on the selected activity. Pictures are kept in the media folder and shown via Django templates. Before users carry out the tasks, they can visualize and grasp the procedures through the instructions given to them. In addition to that, they are also able to view the map, the dates for hospital visits, and the contact information of the related hospitals. Figure 4.4, 4.5



Flexibility

Turn your head to the right and left; for each turn, wait for 10 seconds.

In sitting situation bending your head to right and left for 10 seconds.

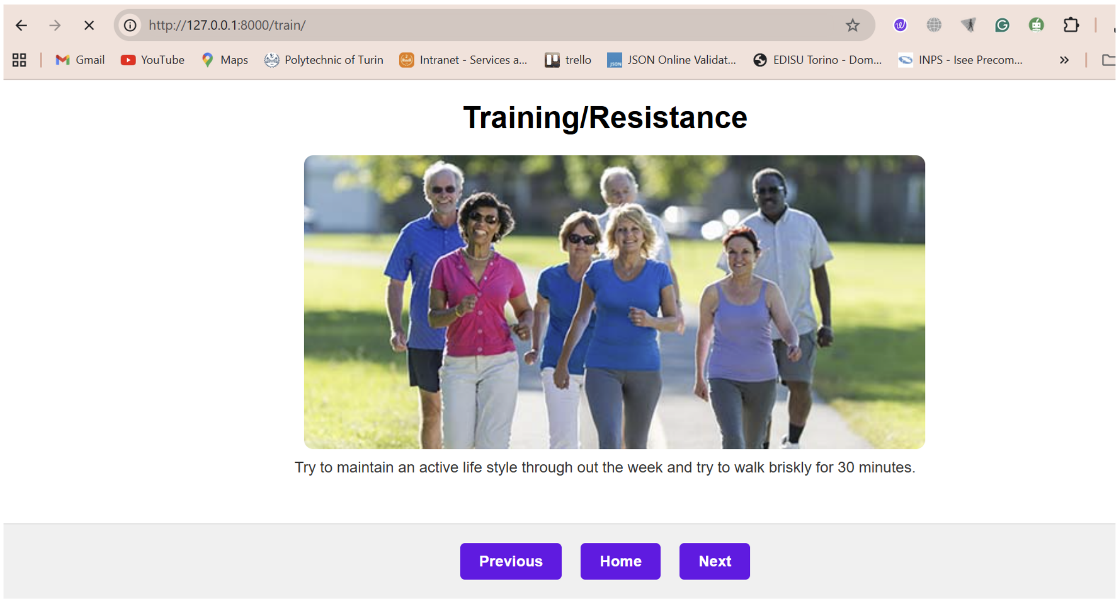
In sitting situation bending your head to top and down for 10 seconds.

In sitting position,rotate your body to right and left and maintain the position for 10 seconds.

While sitting situation tilt your body with your hand to right and left,maintain for 10 seconds.

While sitting on the edge of the chair,spread your arms,keep it for 10 seconds.

Figure 4.4: Example of Exercise.



Training/Resistance

Try to maintain an active life style through out the week and try to walk briskly for 30 minutes.

Figure 4.5: Example of Training.

4.1.4 Navigation and User Experience

- Navbar: Allows users to move between the dashboard, activity logging, and instructional content. Figure 4.6
- Responsive Design: CSS and media queries make sure that pages work on both desktop and mobile devices.
- Feedback Messages: The Django messages framework provides feedback, such as "Activity logged successfully."

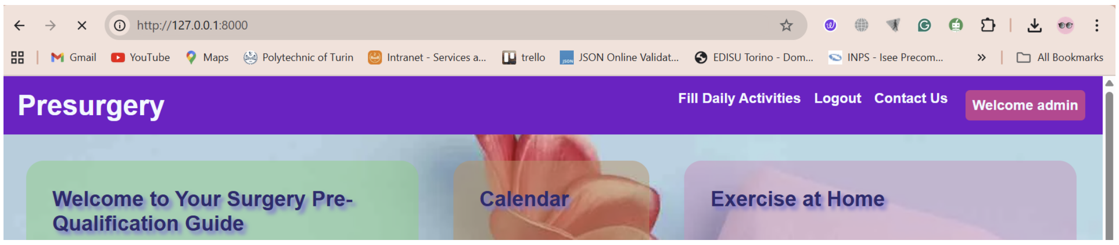
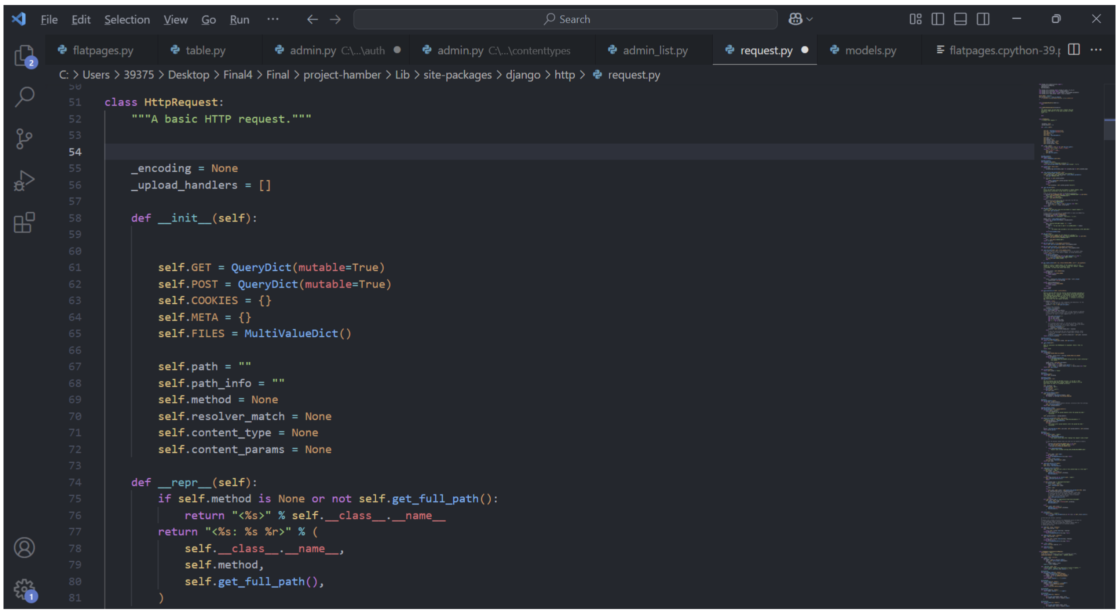


Figure 4.6: Navbar.

4.1.5 Integration with Backend

Every form on the web interface is linked to a Django view that takes in user input through HTTP requests. Thus, the Activity Logging form in the interface sends data to a Django view for data storage. The view here is the handler for **GET** requests, to load the page, and **POST** requests, to update the database with the given data. Additionally, it maintains the record of the user performing each action. As such, this is an example of the application's full-stack integration, which combines user interface, backend logic, and database. Figure 4.7

- `login_required` ensures only authenticated users can log activities.
- `request.method` distinguishes between GET (rendering the form) and POST (processing submission).
- `form.save(commit=False)` allows modifying the object before saving, here to associate it with `request.user`.
- `redirect()` sends the user back to the activity page after successful submission.


File
Edit
Selection
View
Go
Run
Search
flatpages.py
table.py
admin.py C:\...\auth
admin.py C:\...\contenttypes
admin_list.py
request.py
models.py
flatpages.cpython-39.
C: > Users > 39375 > Desktop > Final4 > Final > project-hamber > Lib > site-packages > django > http > request.py

```python
class HttpRequest:
    """A basic HTTP request."""


    _encoding = None
    _upload_handlers = []

    def __init__(self):


        self.GET = QueryDict(mutable=True)
        self.POST = QueryDict(mutable=True)
        self.COOKIES = {}
        self.META = {}
        self.FILES = MultiValueDict()

        self.path = ""
        self.path_info = ""
        self.method = None
        self.resolver_match = None
        self.content_type = None
        self.content_params = None

    def __repr__(self):
        if self.method is None or not self.get_full_path():
            return "<%s>" % self.__class__.__name__
        return "<%s: %s %r>" % (
            self.__class__.__name__,
            self.method,
            self.get_full_path(),
        )
```

Figure 4.7: Request Method.

4.2 Backend Development with Django

The backend system is the mind of the project that implements all the business logic, deals with the user interface and data storage communication, and also provides security, scalability, and consistency. If the frontend is the face of the hospital patients and medical staff, the backend is still working for them to manage their requests, take care of authentication, ensure security, and make database interaction possible. We used Django mainly for the reasons that we have mentioned prior i.e. Fast development, Security, Scalability, ORM compatibility, and a large open-source community that facilitates by offering packages, solutions, and documentation.

The directory structure of the Django project is as follows:

```
project_root/
|
|-- manage.py
|-- hamber/      # Main project configuration
|   |-- __init__.py
|   |-- settings.py
|   |-- urls.py
|   |-- asgi.py
|   `-- wsgi.py
|
|-- home/        # Custom application (patients, activities, etc)
|   |-- models.py
|   |-- views.py
|   |-- urls.py
|   |-- admin.py
|   |-- forms.py
|   `-- templates/
|
`-- db.sqlite3   # SQLite3 database
```

About Core Files Explanation: When working with a Django project, there are a few files that show up right away, and each plays a very specific role:

- **manage.py :** It is the project's control panel. From here, you can start the development server, apply or roll back database migrations, or even create a superuser account to access the admin panel. It's the file you'll often reach for when you want to run quick commands.
- **settings.py :** Here is the core of the configuration. The file describes to Django the details of the database to be connected, what apps are currently

active, and the general settings of middleware, templates, and static files. Practically any significant setting would be changed here.

- **urls.py :**The file is a controller of some sort. If a request (for example, a visit to /login/ or /dashboard/) is made, then this file finds the appropriate view to manage the request. Simply put, the program would be unable to provide different outputs for different web addresses without this file.

- **wsgi.py and asgi.py :** In daily work, these two files are hardly noticed, but they have an essential role when the project goes live. WSGI works with standard server configurations, whereas ASGI is designed for asynchronous operations that require instant communication. In simple terms, they are the “access point” from the web server to the application.

- **apps (for example, home/) :** Django encourages breaking things down into smaller, independent apps. Each app has its own purpose: one might take care of patient information, another might schedule appointments, and another might manage nutrition logs. This modular design keeps the overall project organized and easier to extend later.

In Django, there is middleware. The request is first passed through these middleware layers when someone clicks on your site; then, it is handled. They are able to do various things like checking, changing, or even blocking the request if something looks suspicious. Only after that, the request goes to the view where your main logic is executed. And it doesn’t stop here. The response from the view, which has to go through the same middlewares as before, is now on its way back. This function is like some helpers who are checking if everything is okay before the user finally gets the page. It’s not very visible, but without it, the whole process of authentication, logging, or other background checks would be a mess. As shown in Figure 4.8, a Django request passes sequentially through middleware, reaches the view for processing, and then returns a response back through middleware to the browser.

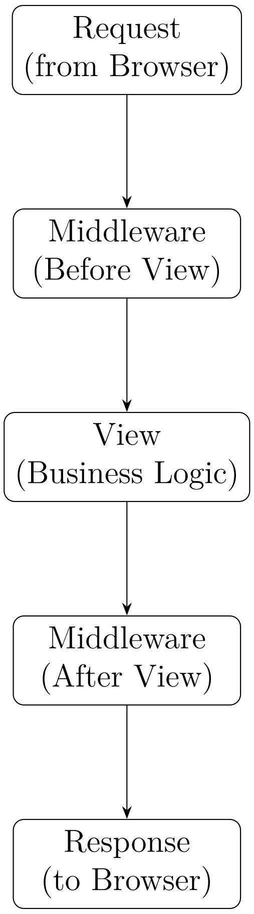
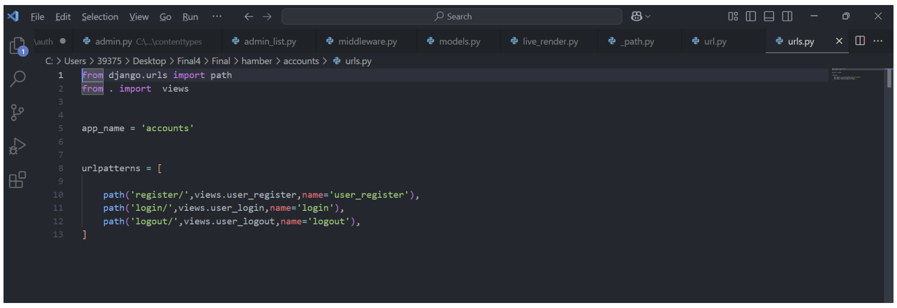


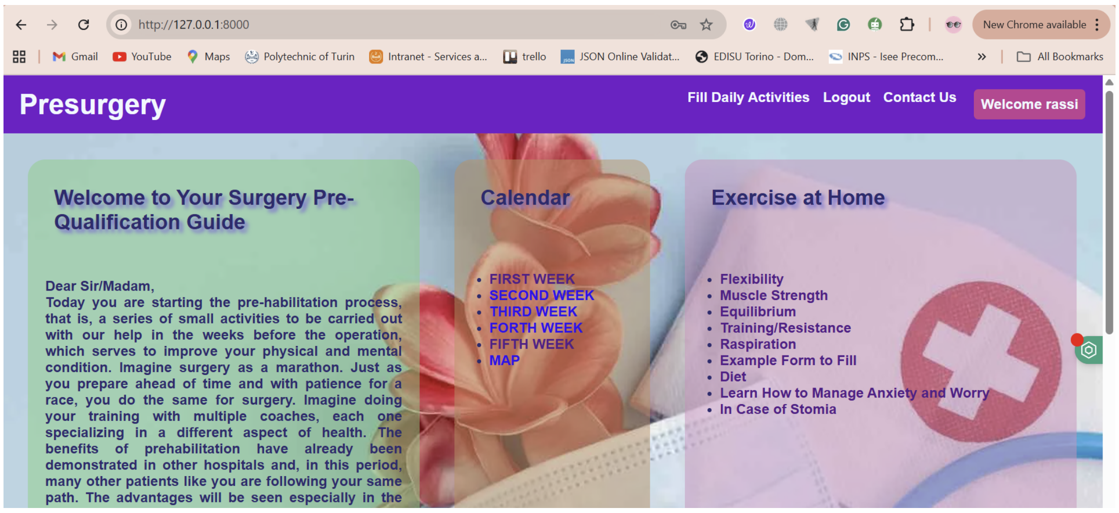
Figure 4.8: Django Request/Response Cycle with Middleware

About URLs and Views determine how data appears to the user and the path to the main goal. Django offers function-based views (FBV) and class-based views (CBV).Figure 4.9 Here, /home/ goes to the dashboard view. This view gets user activities and displays them in HTML.Figure 4.10


File Edit Selection View Go Run
Search
\auth
admin.py C:\...\contenttypes
admin_list.py
middleware.py
models.py
live_render.py
_path.py
url.py
urls.py
C: > Users > 39375 > Desktop > Final4 > Final > hamber > accounts > urls.py

```python
from django.urls import path
from . import  views


app_name = 'accounts'


urlpatterns = [

    path('register/',views.user_register,name='user_register'),
    path('login/',views.user_login,name='login'),
    path('logout/',views.user_logout,name='logout'),
]
```

Figure 4.9: URL.



Presurgery

Fill Daily Activities Logout Contact Us Welcome rassi

Welcome to Your Surgery Pre-Qualification Guide

Dear Sir/Madam,
Today you are starting the pre-habilitation process, that is, a series of small activities to be carried out with our help in the weeks before the operation, which serves to improve your physical and mental condition. Imagine surgery as a marathon. Just as you prepare ahead of time and with patience for a race, you do the same for surgery. Imagine doing your training with multiple coaches, each one specializing in a different aspect of health. The benefits of prehabilitation have already been demonstrated in other hospitals and, in this period, many other patients like you are following your same path. The advantages will be seen especially in the

Calendar

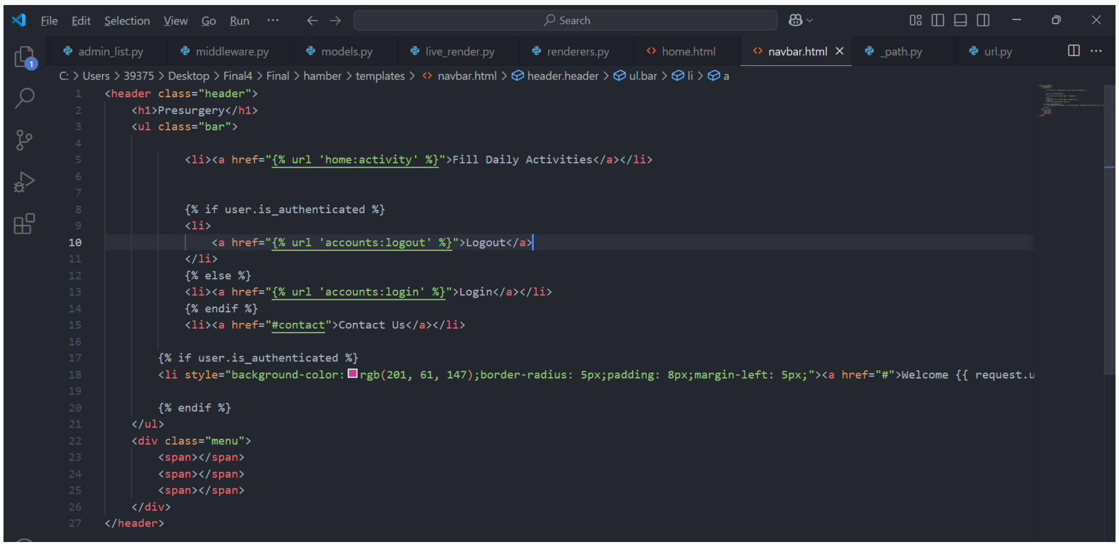
- FIRST WEEK
- SECOND WEEK
- THIRD WEEK
- FORTH WEEK
- FIFTH WEEK
- MAP

Exercise at Home

- Flexibility
- Muscle Strength
- Equilibrium
- Training/Resistance
- Raspiration
- Example Form to Fill
- Diet
- Learn How to Manage Anxiety and Worry
- In Case of Stomia

Figure 4.10: View HTML.

Django uses a template engine to combine HTML with dynamic data. This code displays personalized activities for each logged-in user. Figures 4.11, 4.12


File Edit Selection View Go Run ...
Search
admin_list.py
middleware.py
models.py
live_render.py
renderers.py
home.html
navbar.html
_path.py
url.py
C: > Users > 39375 > Desktop > Final4 > Final > hamber > templates > navbar.html > header.header > ul.bar > li > a

```
<header class="header">
    <h1>Presurgery</h1>
    <ul class="bar">

            <li><a href="{% url 'home:activity' %}">Fill Daily Activities</a></li>


            {% if user.is_authenticated %}
            <li>
                <a href="{% url 'accounts:logout' %}">Logout</a>
            </li>
            {% else %}
            <li><a href="{% url 'accounts:login' %}">Login</a></li>
            {% endif %}
            <li><a href="#contact">Contact Us</a></li>

        {% if user.is_authenticated %}
        <li style="background-color: rgb(201, 61, 147);border-radius: 5px;padding: 8px;margin-left: 5px;"><a href="#">Welcome {{ request.u

        {% endif %}
    </ul>
    <div class="menu">
        <span></span>
        <span></span>
        <span></span>
    </div>
</header>
```

Figure 4.11: Combine HTML with Dynamic Data.

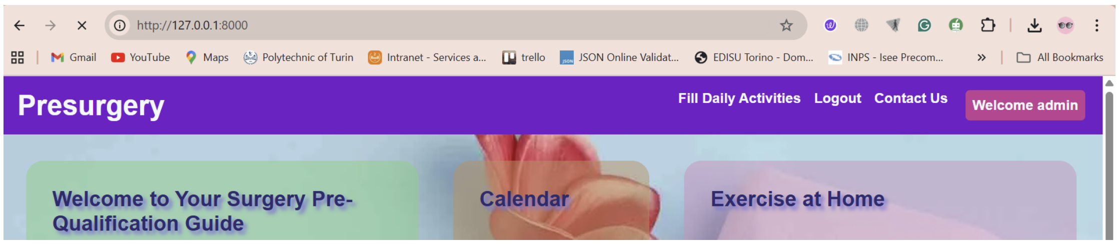
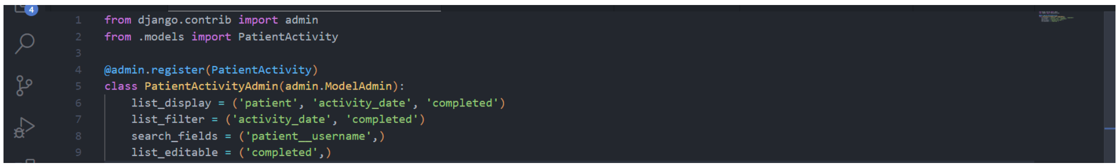


Figure 4.12: Screenshot of Dashboard Page Rendering.

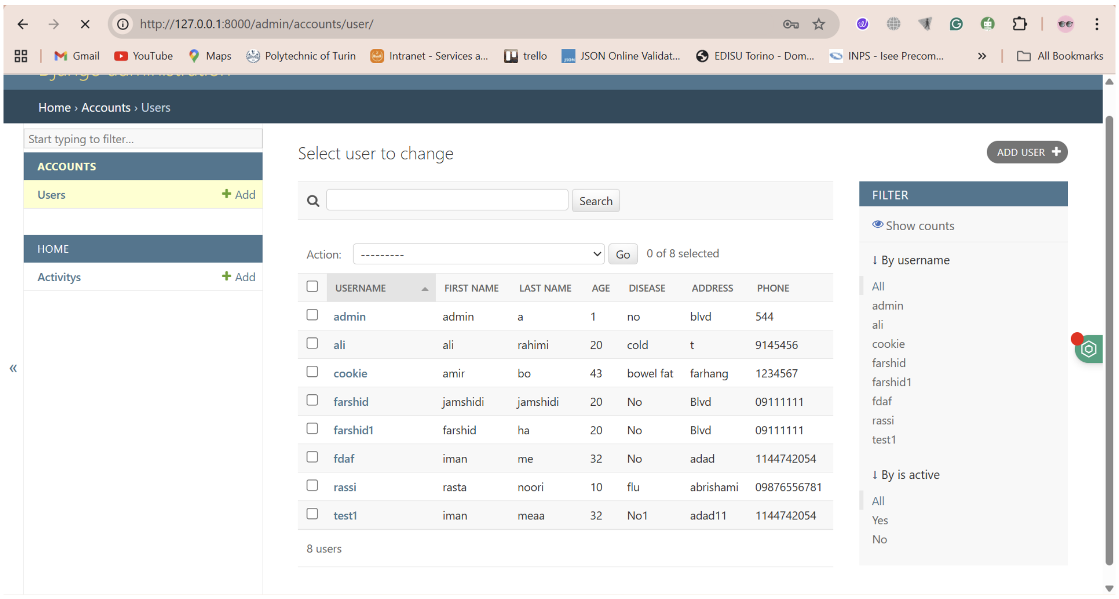
Django has an admin interface by default. The admin interface enables users to carry out CRUD operations on models without the need to develop a separate dashboard. By means of this modification, the management team is able to control, locate, and maintain the patient's activities through the backend without any hassle.

Figures 4.13, 4.14



```python
from django.contrib import admin
from .models import PatientActivity

@admin.register(PatientActivity)
class PatientActivityAdmin(admin.ModelAdmin):
    list_display = ('patient', 'activity_date', 'completed')
    list_filter = ('activity_date', 'completed')
    search_fields = ('patient__username',)
    list_editable = ('completed',)
```

Figure 4.13: Admin Panel.

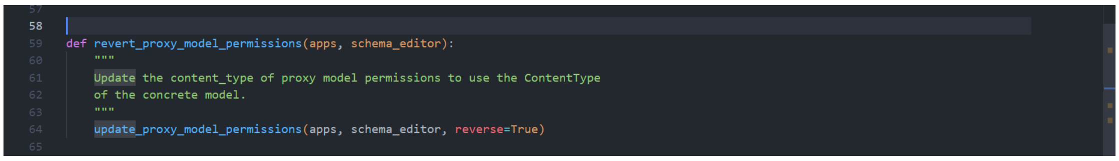


USERNAME	FIRST NAME	LAST NAME	AGE	DISEASE	ADDRESS	PHONE
admin	admin	a	1	no	blvd	544
ali	ali	rahimi	20	cold	t	9145456
cookie	amir	bo	43	bowel fat	farhang	1234567
farshid	jamshidi	jamshidi	20	No	Blvd	09111111
farshid1	farshid	ha	20	No	Blvd	09111111
fdaf	iman	me	32	No	adad	1144742054
rassi	rasta	noori	10	flu	abrishami	09876556781
test1	iman	meaa	32	No1	adad11	1144742054

8 users

Figure 4.14: Screenshot of Customized Admin Panel.

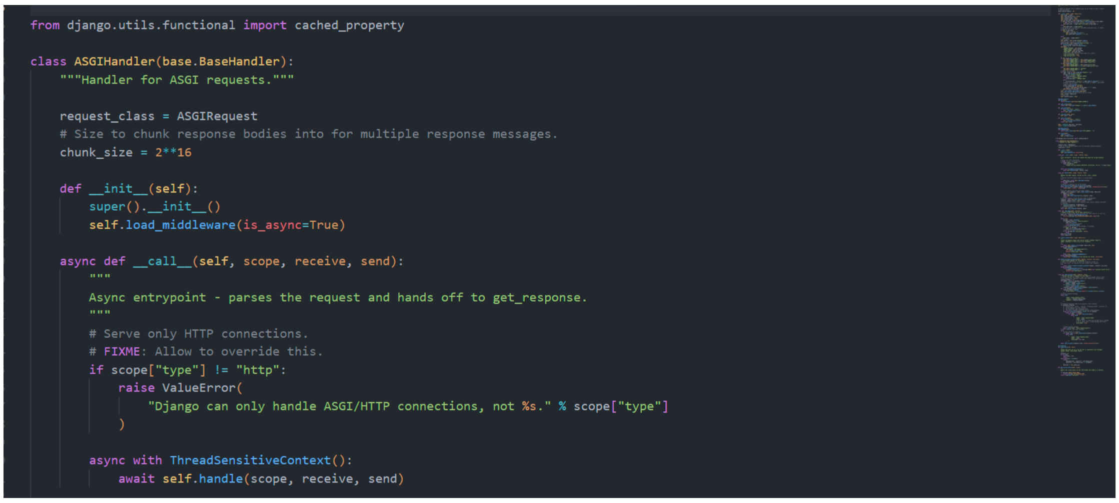
Security is a top priority, undoubtedly, in any healthcare-related software system or application. The good news is that Django has its own built-in facilities to offer a positive outcome in this sphere. User accounts are unique, while different user categories like doctors, nurses, and admins can be assigned different sets of permissions. Moreover, Django does it automatically for each data model; it provides default permission operations such as adding, changing, deleting, and viewing. Additionally, you can define your own permission if required. So to illustrate, assuming a model named PatientRecord, staff members can be allowed to update the information with the help of permissions, but not necessarily delete the records. All this information is stored by Django in the database. Thus, the actions are only accessible to users with sufficient privileges. Django has a method of making sure that every username is unique and valid. It can come up with a default username and check if it doesn't conflict with any other existing usernames whenever new users are being created. This method keeps accounts in order and avoids errors. Figure 4.15 illustrates it clearly.



```python

def revert_proxy_model_permissions(apps, schema_editor):
    """
    Update the content_type of proxy model permissions to use the ContentType
    of the concrete model.
    """
    update_proxy_model_permissions(apps, schema_editor, reverse=True)

```

Figure 4.15: Update permissions for proxy models.

Django normally runs in a regular synchronous mode (WSGI), but newer features need it to handle things in real-time, like WebSockets, which is asynchronous (ASGI). To make sure everything still works smoothly, the project used some built-in Django helpers like guarantee-single-callable, CurrentThreadExecutor, and Local. For example, guarantee-single-callable just wraps a function so it can run safely whether the app is using old-style synchronous calls or the newer async setup. Figure 4.16

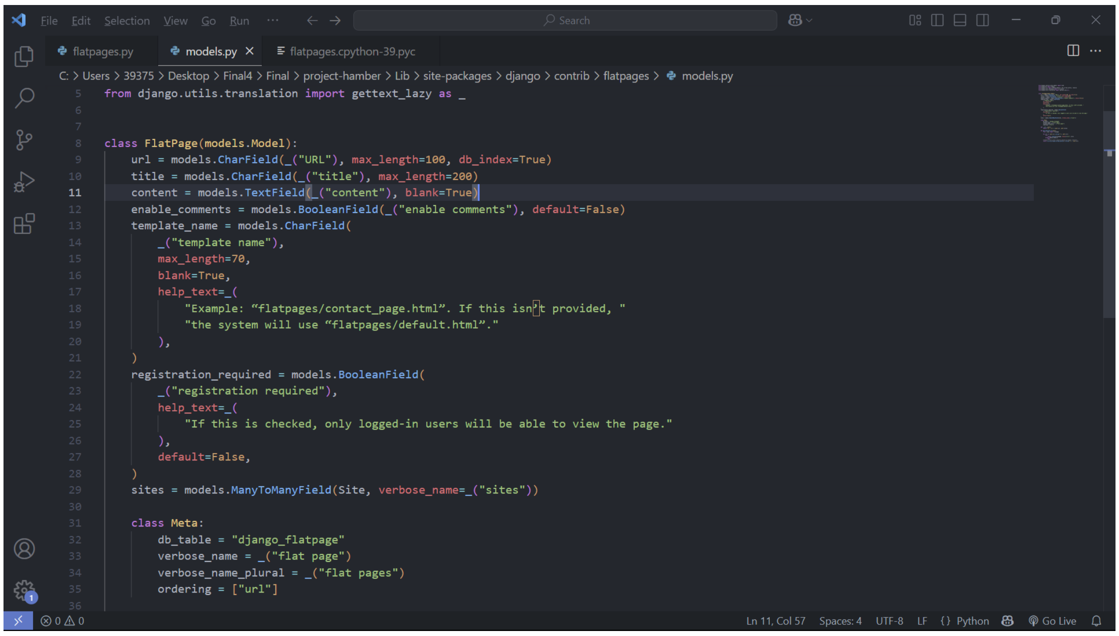


```python
from django.utils.functional import cached_property

class ASGIHandler(base.BaseHandler):
    """Handler for ASGI requests."""

    request_class = ASGIRequest
    # Size to chunk response bodies into for multiple response messages.
    chunk_size = 2**16

    def __init__(self):
        super().__init__()
        self.load_middleware(is_async=True)

    async def __call__(self, scope, receive, send):
        """
        Async entrypoint - parses the request and hands off to get_response.
        """
        # Serve only HTTP connections.
        # FIXME: Allow to override this.
        if scope["type"] != "http":
            raise ValueError(
                "Django can only handle ASGI/HTTP connections, not %s." % scope["type"]
            )

        async with ThreadSensitiveContext():
            await self.handle(scope, receive, send)
```

Figure 4.16: Handle Asinc Requests.

4.3 Database Design

Every web application needs some kind of storage for its data. In this particular case, we were using Django's ORM, which allows the database to be accessed in Python rather than in SQL. For our development, we chose to use SQLite3 as our database, which is basically a simple and easy to set up one. In Django, models are nothing more than Python classes that specify the ways data will be stored. Each class attribute gets converted into a field of the database. Django takes care of all the changes, like creating and updating the tables, with migrations that are automated. Besides that, we relied on the FlatPage model from Django's flatpages. It is a handy tool, in fact, for getting up fast small pages such as "About Us" or "Help," without the need for your intervention in the writing of the HTML code. The only thing you have to do is save the content in the database, and thus editing will be easier. . Figure 4.17



```python
from django.utils.translation import gettext_lazy as _


class FlatPage(models.Model):
    url = models.CharField(_("URL"), max_length=100, db_index=True)
    title = models.CharField(_("title"), max_length=200)
    content = models.TextField(_("content"), blank=True)
    enable_comments = models.BooleanField(_("enable comments"), default=False)
    template_name = models.CharField(
        _("template name"),
        max_length=70,
        blank=True,
        help_text=_(
            "Example: “flatpages/contact_page.html”. If this isn’t provided, "
            "the system will use “flatpages/default.html”."
        ),
    )
    registration_required = models.BooleanField(
        _("registration required"),
        help_text=_(
            "If this is checked, only logged-in users will be able to view the page."
        ),
        default=False,
    )
    sites = models.ManyToManyField(Site, verbose_name=_("sites"))

    class Meta:
        db_table = "django_flatpage"
        verbose_name = _("flat page")
        verbose_name_plural = _("flat pages")
        ordering = ["url"]

```

Figure 4.17: Database Structure.

When Django handles this model, it sets up the database table automatically. The FlatPage model ends up as a table called django-flatpage, and it comes with a bunch of fields that match the model's attributes: Table 4.1

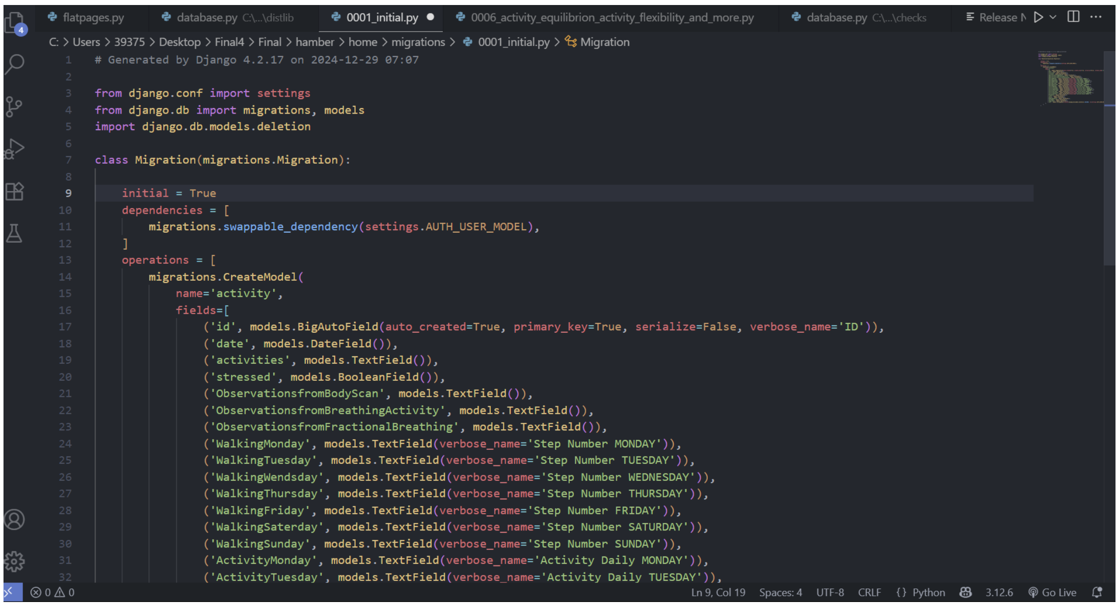
Field Name	Type	Description
`id (auto)`	Primary Key	Auto-increment unique identifier
`url`	VARCHAR(100)	Relative URL of the page
`title`	VARCHAR(200)	Title of the page
`content`	TEXT	Body content of the page
`enable_comments`	Boolean	Allow comments on the page
`template_name`	VARCHAR(70)	Custom template path
`registration_required`	Boolean	Restrict to logged-in users
`sites`	ManyToMany	Relation to Site model

Table 4.1: FlatPage model fields and descriptions

This table represents the database in a very simple and user-friendly manner. Each field shows the data that is going to be stored and its purpose. Such a table is also convenient for the team during the development phase when they need to discuss the database.

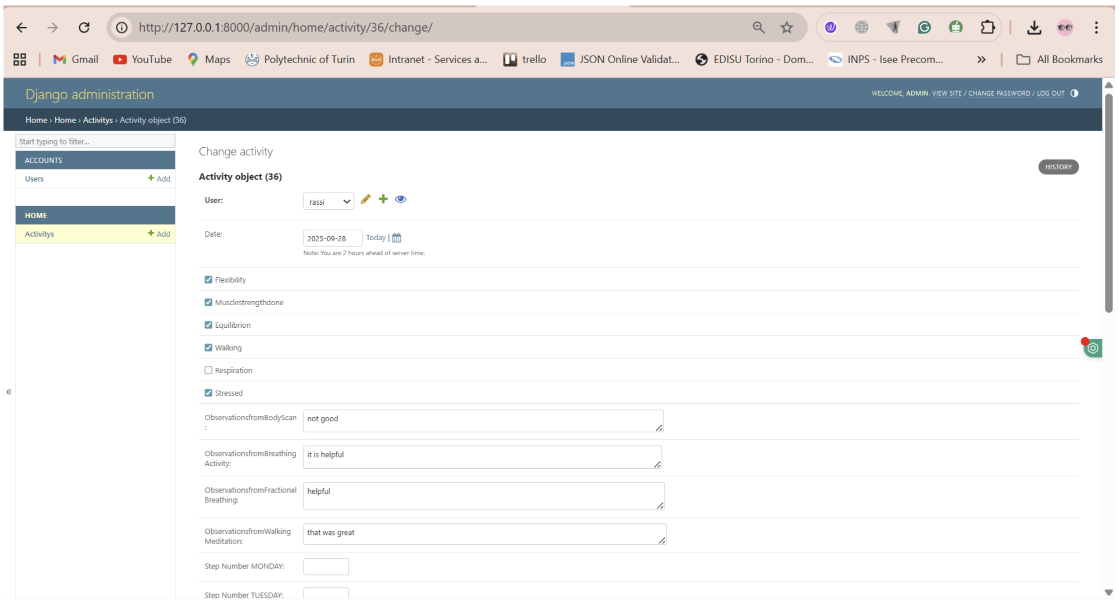
For this project, we mapped out the database using the classic ER diagrams and also made some visual schematics in Python using the Rich library to see how everything connects.

We used the Rich library to make our terminal output look nicer. Its Table feature lets you show database tables in a neat, readable format. For example, we could display the FlatPage table with all its fields, types, and explanations, all lined up. This made it easy to peek at the database structure and make sure everything looked right without opening extra tools or diagrams. Figures 4.18, 4.19.


flatpages.py
database.py C:\...\distlib
0001_initial.py
0006_activity_equilibrion_activity_flexibility_and_more.py
database.py C:\...\checks
Release N
C: > Users > 39375 > Desktop > Final4 > Final > hamber > home > migrations > 0001_initial.py > Migration
0 0
Ln 9, Col 19
Spaces: 4
UTF-8
CRLF
{} Python
3.12.6
Go Live

```python
# Generated by Django 4.2.17 on 2024-12-29 07:07

from django.conf import settings
from django.db import migrations, models
import django.db.models.deletion

class Migration(migrations.Migration):

    initial = True
    dependencies = [
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
    ]
    operations = [
        migrations.CreateModel(
            name='activity',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('date', models.DateField()),
                ('activities', models.TextField()),
                ('stressed', models.BooleanField()),
                ('ObservationsfromBodyScan', models.TextField()),
                ('ObservationsfromBreathingActivity', models.TextField()),
                ('ObservationsfromFractionalBreathing', models.TextField()),
                ('WalkingMonday', models.TextField(verbose_name='Step Number MONDAY')),
                ('WalkingTuesday', models.TextField(verbose_name='Step Number TUESDAY')),
                ('WalkingWendsday', models.TextField(verbose_name='Step Number WEDNESDAY')),
                ('WalkingThursday', models.TextField(verbose_name='Step Number THURSDAY')),
                ('WalkingFriday', models.TextField(verbose_name='Step Number FRIDAY')),
                ('WalkingSaterday', models.TextField(verbose_name='Step Number SATURDAY')),
                ('WalkingSunday', models.TextField(verbose_name='Step Number SUNDAY')),
                ('ActivityMonday', models.TextField(verbose_name='Activity Daily MONDAY')),
                ('ActivityTuesday', models.TextField(verbose_name='Activity Daily TUESDAY')),
```

Figure 4.18: Using Rich Library.



http://127.0.0.1:8000/admin/home/activity/36/change/

Django administration

Home › Home › Activitys › Activity object (36)

Change activity

Activity object (36)

User:	rassi
Date:	2025-09-28
[✓] Flexibility	
[✓] Musclestrengthdone	
[✓] Equilibrion	
[✓] Walking	
[ ] Respiration	
[✓] Stressed	
ObservationsfromBodyScan :	not good
ObservationsfromBreathing Activity:	it is helpful
ObservationsfromFractional Breathing:	helpful
ObservationsfromWalking Meditation:	that was great
Step Number MONDAY:	
Step Number TUESDAY:	

Figure 4.19: Database Form.

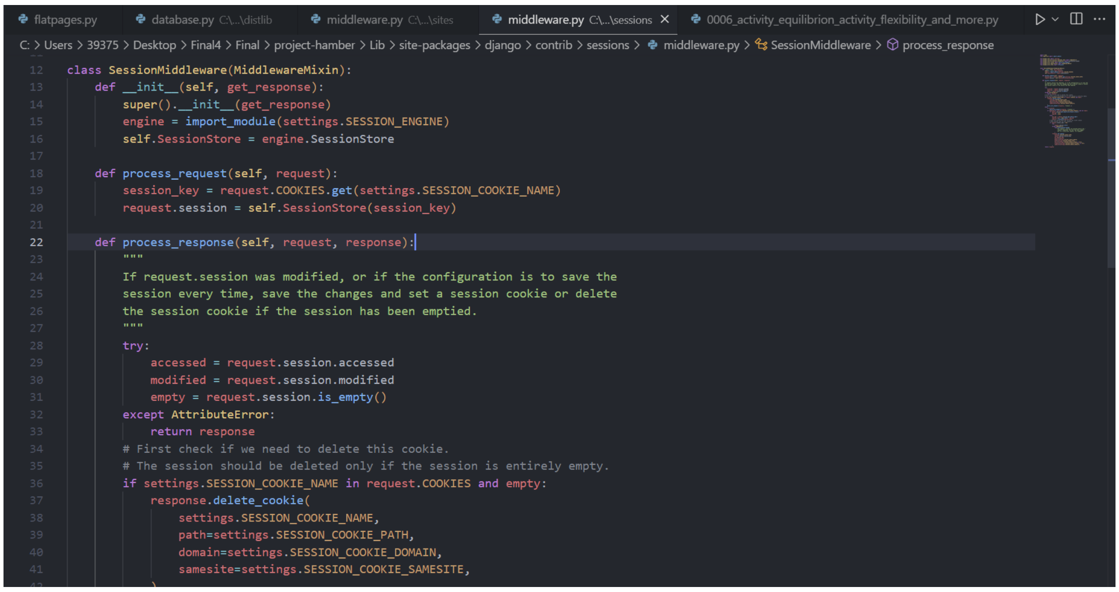
Middleware works like a helper that checks things before and after a page loads. Before showing a page, it can see if someone is logged in. That way, pages meant only for registered users stay private.

When the app finishes making a page, middleware can also update info or keep track of activity in the database.

There are certain pages in a web application that are restricted to users who have logged in. Django managesâ€‹â€‹ these through sessions â€‹â€‹- small records that keep track of who is logged in. The SessionMiddleware acts on behalf of the system to do everything. It gets the user's session cookie upon a page request, checks its validity, and identifies the user. When a page is returned, it also changes or gets rid of the cookie if necessary. This is the mechanism by which users who have logged in can only have access to secure pages.

For instance, if a page requires registration, middleware checks the session first and either lets the user in or blocks access. You can also connect this to the database: the session info links to the auth-user table, which stores user accounts. Including a screenshot of that table shows how the app keeps track of users.

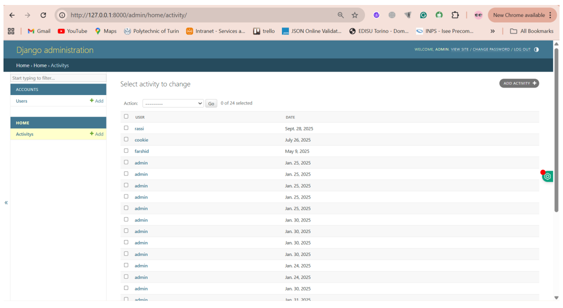
This setup means that database logic isn't handled by models and views alone—middleware helps keep everything secure and consistent. This middleware manages user sessions. It checks the session cookie and keeps track of who is logged in, which is why pages requiring login only show to authenticated users. Figures 4.20, 4.21 show that.



flatpages.py
database.py C:\...\distlib
middleware.py C:\...\sites
middleware.py C:\...\sessions
0006_activity_equilibrion_activity_flexibility_and_more.py
C: > Users > 39375 > Desktop > Final4 > Final > project-hamber > Lib > site-packages > django > contrib > sessions > middleware.py > SessionMiddleware > process_response

```python
class SessionMiddleware(MiddlewareMixin):
    def __init__(self, get_response):
        super().__init__(get_response)
        engine = import_module(settings.SESSION_ENGINE)
        self.SessionStore = engine.SessionStore

    def process_request(self, request):
        session_key = request.COOKIES.get(settings.SESSION_COOKIE_NAME)
        request.session = self.SessionStore(session_key)

    def process_response(self, request, response):
        """
        If request.session was modified, or if the configuration is to save the
        session every time, save the changes and set a session cookie or delete
        the session cookie if the session has been emptied.
        """
        try:
            accessed = request.session.accessed
            modified = request.session.modified
            empty = request.session.is_empty()
        except AttributeError:
            return response
        # First check if we need to delete this cookie.
        # The session should be deleted only if the session is entirely empty.
        if settings.SESSION_COOKIE_NAME in request.COOKIES and empty:
            response.delete_cookie(
                settings.SESSION_COOKIE_NAME,
                path=settings.SESSION_COOKIE_PATH,
                domain=settings.SESSION_COOKIE_DOMAIN,
                samesite=settings.SESSION_COOKIE_SAMESITE,
```

Figure 4.20: Middleware Manages User Sessions.



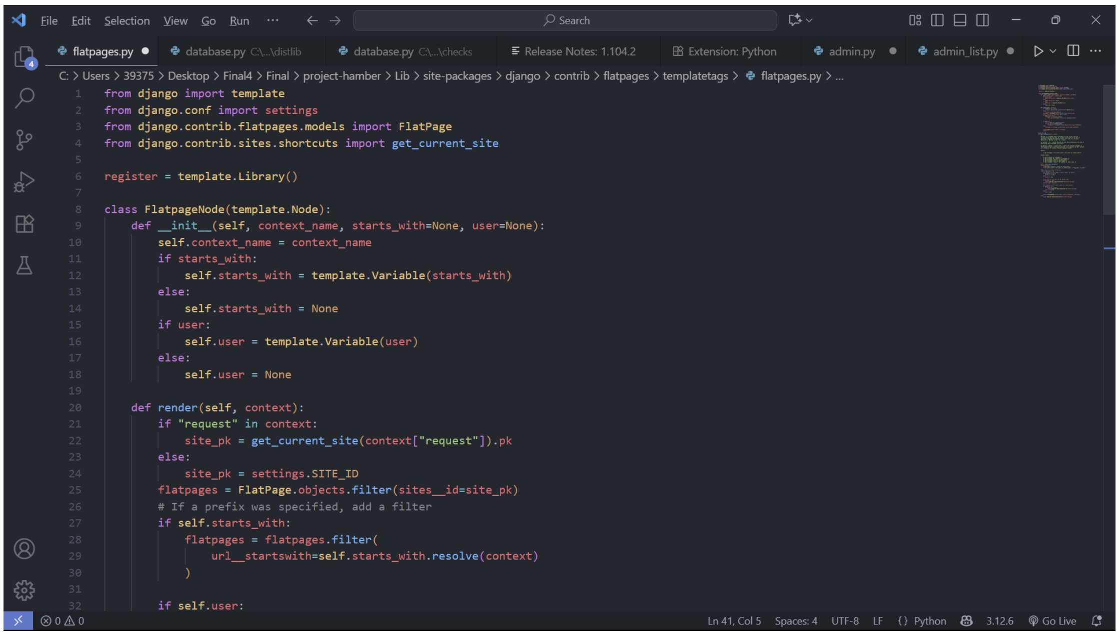
Select activity to change

0 of 24 selected

USER	DATE
rassi	Sept. 28, 2025
cookie	July 26, 2025
farshid	May 9, 2025
admin	Jan. 25, 2025
admin	Jan. 25, 2025
admin	Jan. 25, 2025
admin	Jan. 25, 2025
admin	Jan. 25, 2025
admin	Jan. 30, 2025
admin	Jan. 30, 2025
admin	Jan. 30, 2025
admin	Jan. 30, 2025
admin	Jan. 24, 2025
admin	Jan. 24, 2025
admin	Jan. 30, 2025
admin	Jan. 31, 2025

Figure 4.21: Activity of Users and Admin.

Django's Admin Panel is really the heart of content management on the website. Essentially, through the web interface, users with the right access can add, change, or delete pages; thus, no direct interaction with the database is needed. In order to ease the tasks even more, the project comprises some minor helper functions and template tags that facilitate how pages are displayed in the admin, like sorting, filtering, and paging through multiple pages. Besides that, I put together a special template tag for FlatPages, giving the site the ability to show pages for a current site as well as display them only for users with proper authorization. The tag is also quite handy for filtering pages by URL, which is the easiest way for you to have "About Us" or "Help" pages included into templates without having to do extra work in the view. Figure 4.22

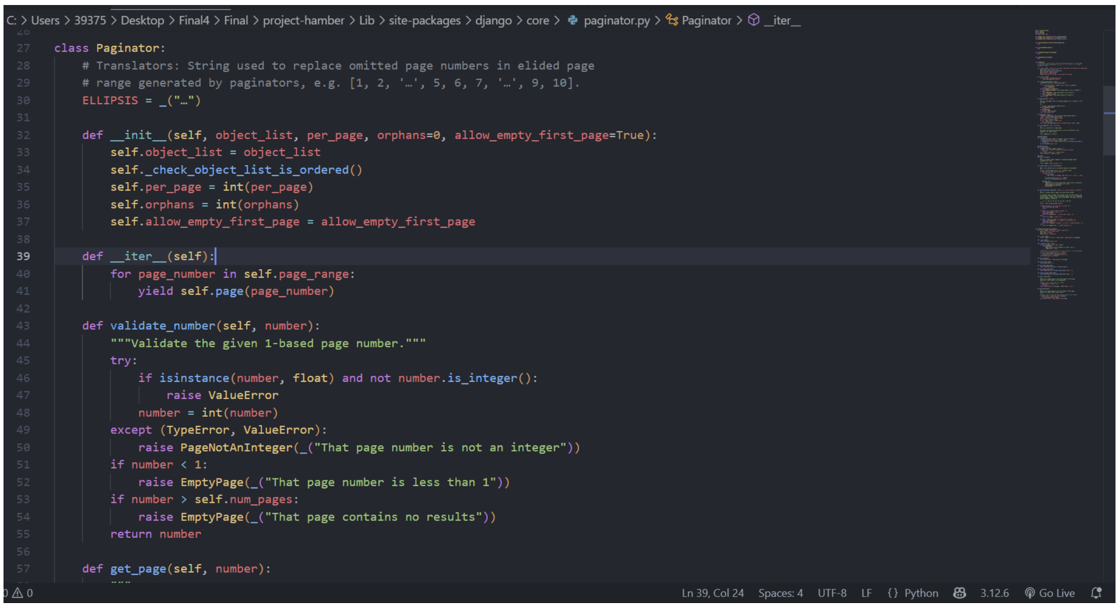

File Edit Selection View Go Run ···
Search
flatpages.py
database.py C:\...\distlib
database.py C:\...\checks
Release Notes: 1.104.2
Extension: Python
admin.py
admin_list.py
C: > Users > 39375 > Desktop > Final4 > Final > project-hamber > Lib > site-packages > django > contrib > flatpages > templatetags > flatpages.py > ...
Ln 41, Col 5 Spaces: 4 UTF-8 LF {} Python 3.12.6 Go Live

```python
from django import template
from django.conf import settings
from django.contrib.flatpages.models import FlatPage
from django.contrib.sites.shortcuts import get_current_site

register = template.Library()

class FlatpageNode(template.Node):
    def __init__(self, context_name, starts_with=None, user=None):
        self.context_name = context_name
        if starts_with:
            self.starts_with = template.Variable(starts_with)
        else:
            self.starts_with = None
        if user:
            self.user = template.Variable(user)
        else:
            self.user = None

    def render(self, context):
        if "request" in context:
            site_pk = get_current_site(context["request"]).pk
        else:
            site_pk = settings.SITE_ID
        flatpages = FlatPage.objects.filter(sites__id=site_pk)
        # If a prefix was specified, add a filter
        if self.starts_with:
            flatpages = flatpages.filter(
                url__startswith=self.starts_with.resolve(context)
            )

        if self.user:
```

Figure 4.22: Admin FlatPage list.

In the admin interface, to simplify the process of managing lists, I have included several features. To make it easy for users to move from one page to another, pagination controls were implemented; they can use page numbers, ellipses, or a "show all" button, for instance. Sorting by various fields was the main idea behind upgrading table headers and, consequently, besides sorting, users can also find easy-to-read results for yes/no queries or related objects in a specially formatted section of the table. If you look at every individual row of data in list view, you'll find that it not only displays the correct information but also takes care of empty fields, foreign keys, and even editable fields if the user has the necessary permission. Data behind the scenes is arranged in such a way that the templates can properly present it.

In addition to that, I incorporated a detailed date breakdown feature that allows administrators to view the records by year, month, or day. Moreover, a search box operates throughout the list, dynamic filters enable precise narrowing of results, and bulk action tracking facilitates the smooth performance of bulk delete or update operations. Additionally, the ease of managing single objects has been enhanced with the provision of tools that can be found at the top of each list and keeping everything simple to access and use. Figure 4.23.



C: > Users > 39375 > Desktop > Final4 > Final > project-hamber > Lib > site-packages > django > core > paginator.py > Paginator > _iter_
Ln 39, Col 24
Spaces: 4
UTF-8
LF
{} Python
3.12.6
Go Live

```python
class Paginator:
    # Translators: String used to replace omitted page numbers in elided page
    # range generated by paginators, e.g. [1, 2, '…', 5, 6, 7, '…', 9, 10].
    ELLIPSIS = _("…")

    def __init__(self, object_list, per_page, orphans=0, allow_empty_first_page=True):
        self.object_list = object_list
        self._check_object_list_is_ordered()
        self.per_page = int(per_page)
        self.orphans = int(orphans)
        self.allow_empty_first_page = allow_empty_first_page

    def __iter__(self):
        for page_number in self.page_range:
            yield self.page(page_number)

    def validate_number(self, number):
        """Validate the given 1-based page number."""
        try:
            if isinstance(number, float) and not number.is_integer():
                raise ValueError
            number = int(number)
        except (TypeError, ValueError):
            raise PageNotAnInteger(_("That page number is not an integer"))
        if number < 1:
            raise EmptyPage(_("That page number is less than 1"))
        if number > self.num_pages:
            raise EmptyPage(_("That page contains no results"))
        return number

    def get_page(self, number):
```

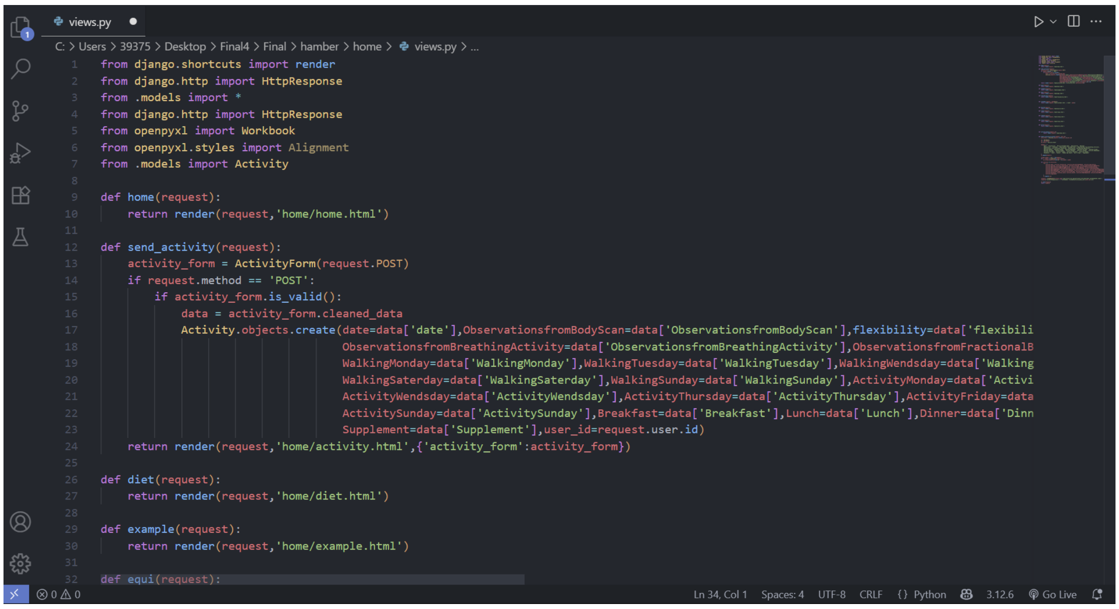
Figure 4.23: How Django Paginate.

This was an additional requirement for the project: the user needed a simple tool to extract patient records into an Excel file. The concept turned out to be a perfect way for doctors and the hospital staff to download patient data quickly, thereby having access to the patient's database offline and not relying on the web system if they wished. The system will extract the data that has been stored (for example, the visit, diagnosis, or treatment notes) and will lay out the data in proper rows and columns using Django's built-in instruments in conjunction with a library such as openpyxl. The generated file can then be saved locally or distributed as any other Excel file. This feature also gives a lot of flexibility to the medical staff as they can check the data, cross-check appointments, or even create reports without having to be logged into the application all the time.

This function lets a user download all of their saved activity records in Excel format. When the export button is clicked, Django looks up all the activities in the database that belong to that user (using their user ID).

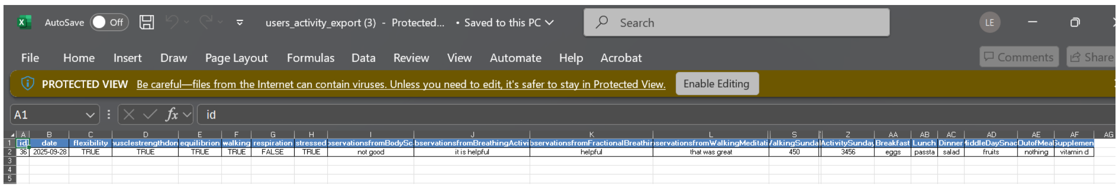
It then creates a new Excel file using the openpyxl library. The file starts with a header row, where each column represents something the patient recorded, like flexibility, walking, meals, or daily notes.

Next, the program goes through each activity entry in the database and writes the values into the Excel sheet, row by row. Finally, the file is sent back to the browser as a download, so the patient (or doctor) can keep it, share it, or review it later. Figures 4.24, 4.25



```
from django.shortcuts import render
from django.http import HttpResponse
from .models import *
from django.http import HttpResponse
from openpyxl import Workbook
from openpyxl.styles import Alignment
from .models import Activity

def home(request):
    return render(request,'home/home.html')

def send_activity(request):
    activity_form = ActivityForm(request.POST)
    if request.method == 'POST':
        if activity_form.is_valid():
            data = activity_form.cleaned_data
            Activity.objects.create(date=data['date'],ObservationsfromBodyScan=data['ObservationsfromBodyScan'],flexibility=data['flexibili
                                    ObservationsfromBreathingActivity=data['ObservationsfromBreathingActivity'],ObservationsfromFractionalB
                                    WalkingMonday=data['WalkingMonday'],WalkingTuesday=data['WalkingTuesday'],WalkingWendsday=data['Walking
                                    WalkingSaterday=data['WalkingSaterday'],WalkingSunday=data['WalkingSunday'],ActivityMonday=data['Activi
                                    ActivityWendsday=data['ActivityWendsday'],ActivityThursday=data['ActivityThursday'],ActivityFriday=data
                                    ActivitySunday=data['ActivitySunday'],Breakfast=data['Breakfast'],Lunch=data['Lunch'],Dinner=data['Dinn
                                    Supplement=data['Supplement'],user_id=request.user.id)
    return render(request,'home/activity.html',{'activity_form':activity_form})

def diet(request):
    return render(request,'home/diet.html')

def example(request):
    return render(request,'home/example.html')

def equi(request):
```

Figure 4.24: Exporting File.



users_activity_export (3) - Protected... • Saved to this PC

PROTECTED VIEW Be careful—files from the Internet can contain viruses. Unless you need to edit, it's safer to stay in Protected View. Enable Editing

A1 id

	A	B	C	D	E	F	G	H	I	J	K	L	S	Z	AA	AB	AC	AD	AE	AF
1	id	date	flexibility	musclestrengthdon	equilibrion	walking	respiration	stressed	bservationsfromBodySc	bservationsfromBreathingActivi	bservationsfromFractionalBreathi	servationsfromWalkingMeditati	/alkingSunda	ActivitySunday	Breakfast	Lunch	Dinner	1iddleDaySnac	OutofMeal	Supplemen
2	36	2025-09-28	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	not good	it is helpful	helpful	that was great	450	3456	eggs	passta	salad	fruits	nothing	vitamin d
3																				
4																				
5																				

Figure 4.25: An Excel File.

Chapter 5

Conclusion and Future Works

Looking back on this project, it has been some journey to witness an idea grow from concept through to fully operational web application. Coming into this project, I had some idea that I wanted something that was both useful as well as usable but the specifics of how it all was going to hang together from frontend right through backend, through design all the way through database were not clear. Working on this thesis gave me an understanding not just of the technical details of building websites but also the value of planning, iterative development as well as attention to detail in the production of the seamless user experience.

It's one of the big lessons of this project how interdependently all the components of a web application depend on each other. Frontend is not only about making something beautiful; it should also be user-friendly, responsive, and converse properly with the backend. Backend is not only about saving data; it is the heart that makes the system robust, scalable, and secure. Carefully combining all the components, selecting technologies such as Django, SQLite, JavaScript, and Bootstrap, I was able to develop an application meeting the objectives set at the project outset. Each of the diagrams, each of the pieces of code, each design choice added up to a cohesive whole demonstrating how small decisions during development may affect the application on the whole but greatly affect the application as per the application on the whole.

One significant thing I learned was the learning-by-doing principle. Although I had some familiarity with Python and introductory web technologies beforehand, creating this project needed further understanding and experimentation by doing. To initialize the database and properly set up communication between the models and the views in the context of Django was hard at the beginning but made me realize the logic behind web frameworks as well as the patterns making large applications manageable. In the same way, the process of planning the user interface and its usability testing made me realize the users' requirements and expectation should forever direct the building protocol.

I also saw the value of clear organization and documentation. Addition of screenshots, coding examples, and diagrams not only assisted me in finding track of the work but will also assist the future developers or users who will come in contact with the system. Proper documentation is never given the credit it deserves but it plays an important part in maintaining as well as enhancing software in the long run. It also made me more careful about the logic behind all decisions from the database structure up to the way the interface was laid out, thereby enhancing the overall project quality.

It was not only an exhibition of technical know-how but also an indication of problem-solving ability, creativity, and determination. Its completon instilled in me the belief that I am capable of taking up challenging software projects and accomplishing them end-to-end. It also emphasized the value of learning without stopping, as for every problem encountered, I had to go researching for newer solutions, investigate different methods of approaching them, as well as know best practices for building websites.

Future Work

From the clinical point of view, the system is backing up the three major stages of quality cancer care which are: prehabilitation, perioperative care, and long-term rehabilitation with continuous follow-up. The platform, by facilitating an active preoperative training and a diligent postoperative monitoring, intends to be a part of the recovery that is faster and also of the recurrence that can be detected at an early stage.

Although the system created in this project fulfills its purposes well, there is certainly potential for future improvement. In one potential area for improvement could the user interface continue by making the system more intuitive as well as aesthetically pleasing, perhaps through the use of newer interactive features or more dynamic designs. In another area for improvement could the functionality of the system's functionality be increased, perhaps by incorporating more modules, enhancing data analyzes, or providing for more individualized uses.

Optimization for security and performance for the future also involves areas. While the existing system runs well and reliably, future releases may integrate more robust authentication procedures, encryption, as well as performance enhancements for supporting larger volumes of data or numbers of users more efficiently. Also, shifting the application over to a more powerful database system or incorporating cloud functionality could provide the application for being more scalable as well as adaptable for practical use.

It also presents an opportunity for incorporating more sophisticated features, like notifications, auto-flows, or even mobility, so the system could access an larger number of people and supply more worth. User responses could also be gathered and studied as part of improving the app continuously so it grows according to their requirements. Lastly, the project may become the starting point for larger web applications or later research projects. What was learned here—design choices to an implementation plan—can apply directly to any subsequent development work so that hard problems may be attacked using a well-structured, user-focused mentality. In summary, this thesis demonstrates the challenges as well as the benefits of constructing a web application entirely on our own. It demonstrates how planning, experimentation, and reflection may collaborate to develop an operative system that happens to run well, be maintainable, as well as easy to use. While doing so, it also leaves the field open for improvement, thereby evidencing software development as an ongoing learning as well as refinement process. I am pleased at what has been achieved as well as eager at the prospects ahead of me, considering the fact that each step in this process has fortified my abilities as well as enriched the understanding of how web applications may be developed.

Bibliography

[1] World Health Organization. «Cancer Control: Knowledge Into Action – WHO Guide for Effective Programmes». In: (2008). URL: `https://www.who.int/publications/i/item/9789241547406` (cit. on pp. 4, 5).

[2] H. S. Richards, J. M. Blazeby, A. Portal, and et al. «A real-time electronic symptom monitoring system for patients after discharge following surgery: a pilot study in cancer-related surgery». In: *BMC Cancer* 20 (2020), p. 543. DOI: `10.1186/s12885-020-07027-5`. URL: `https://doi.org/10.1186/s12885-020-07027-5` (cit. on p. 5).

[3] Samer Sawesi, Mohamed Rashrash, Krit Phalakornkule, Janine S. Carpenter, and Jennifer F. Jones. «The Impact of Information Technology on Patient Engagement and Health Behavior Change: A Systematic Review of the Literature». In: *JMIR Medical Informatics* 4.1 (Jan. 2016), e1. DOI: `10.2196/medinform.4514`. URL: `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4742621/` (cit. on p. 5).

[4] Pho, K. K. et al. *Mobile Device Applications for Electronic Patient Portals in Oncology.* Journal of Oncology Practice, 15(5), e467-e474. Accessed: 2025-10-02. 2019. URL: `https://ascopubs.org/doi/abs/10.1200/CCI.18.00094` (cit. on p. 6).

[5] Carevive. *Carevive: EHR-integrated Cancer Care Management Software.* Accessed: 2025-10-02. n.d. URL: `https://www.carevive.com/` (cit. on p. 6).

[6] Post, A. R. et al. *Electronic Health Record Data in Cancer Learning Health Systems: Challenges and Opportunities.* Journal of Oncology Practice, 18(9), e1227-e1234. Accessed: 2025-10-02. 2022. URL: `https://ascopubs.org/doi/abs/10.1200/CCI.21.00300` (cit. on p. 6).

[7] Lazarou, I. et al. *Cancer Patients' Perspectives and Requirements of Digital Health Technologies: A Scoping Review.* Cancers, 16(13), 2293. Accessed: 2025-10-02. 2024. URL: `https://www.mdpi.com/2072-6694/16/13/2293` (cit. on p. 6).

[8] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann, 1994 (cit. on pp. 6, 7).

[9] John Brooke. «SUS: a 'quick and dirty' usability scale». In: *Usability Evaluation in Industry* (1996), pp. 189–194 (cit. on pp. 6, 7).

[10] *Web Content Accessibility Guidelines (WCAG) 2.2*. `https://www.w3.org/TR/WCAG22/`. 2023 (cit. on pp. 6, 7).

[11] European Union. *General Data Protection Regulation (EU) 2016/679*. 2016. URL: `https://gdpr-info.eu/` (cit. on p. 7).

[12] OWASP Foundation. *OWASP Top Ten 2021*. 2021. URL: `https://owasp.org/Top10/` (cit. on p. 7).

[13] J. L. Baldwin, H. Singh, D. F. Sittig, and T. D. Giardina. «Patient portals and health apps: Pitfalls, promises, and what one might learn from the other». In: *Healthcare (Amsterdam, Netherlands)* 5.3 (2017), pp. 101–107. DOI: `10.1016/j.hjdsi.2016.08.006` (cit. on pp. 8, 9).

[14] E. Carini et al. «The impact of digital patient portals on health outcomes: A systematic review». In: *Journal of Medical Internet Research* 23.7 (2021), e24568. URL: `https://www.jmir.org/2021/7/e24568/` (cit. on pp. 8, 9).

[15] N. El Yaman et al. «Utilization of patient portals: A cross-sectional study». In: *BMC Medical Informatics and Decision Making* 23 (2023), pp. 1–10. URL: `https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-023-02252-x` (cit. on pp. 8, 9).

[16] Y. N. Ong. «Usage of patient portals among primary healthcare patients: A review». In: *Saudi Medical Journal* 40.4 (2019), pp. 353–359. URL: `https://journals.lww.com/smj/fulltext/9900/usage_of_patient_portals_among_primary_healthcare.144.aspx` (cit. on pp. 8, 9).

[17] C. Strawley et al. *Individuals' access and use of patient portals: A systematic review*. National Library of Medicine, 2023. URL: `https://www.ncbi.nlm.nih.gov/books/NBK606032/` (cit. on pp. 8, 9).

[18] Guido van Rossum et al. *PEP 405 – Python Virtual Environments*. `https://www.python.org/dev/peps/pep-0405/`. Accessed: 2025-10-02. 2012 (cit. on p. 11).

[19] Real Python. *Python Virtual Environments: A Primer*. `https://realpython.com/python-virtual-environments-a-primer/`. Accessed: 2025-10-02. 2024 (cit. on p. 11).

[20] W3Schools. *Python Virtual Environments*. `https://www.w3schools.com/python/python_virtualenv.asp`. Accessed: 2025-10-02. 2024 (cit. on p. 11).

[21] Django Software Foundation. *Django Web Framework*. https://www.djangoproject.com. Accessed: 2025-10-02. 2025 (cit. on pp. 11, 13).

[22] Mozilla Developer Network. *Introduction to Django*. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction. Accessed: 2025-10-02. 2024 (cit. on pp. 11, 13).

[23] TechVariable. *Rapid Development and Security in Django*. https://techvariable.com/blogs/django-rapid-development-without-compromising-security. Accessed: 2025-10-02. 2023 (cit. on pp. 11, 13).

[24] Amazon Web Services. *What is Django?* https://aws.amazon.com/what-is/django/. Accessed: 2025-10-02. 2023 (cit. on pp. 12, 13).

[25] Wikipedia Contributors. *Django (web framework)*. https://en.wikipedia.org/wiki/Django_(web_framework). Accessed: 2025-10-02. 2024 (cit. on pp. 12, 13).

[26] MDN Web Docs. *JavaScript First Steps*. Accessed: 2025-10-02. 2023. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps (cit. on p. 13).

[27] MDN Web Docs. *Introduction to the DOM*. Accessed: 2025-10-02. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction (cit. on p. 13).

[28] MDN Web Docs. *Client-side web frameworks*. Accessed: 2025-10-02. 2023. URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction (cit. on p. 13).

[29] Ecma International. *ECMAScript Language Specification (ECMA-262)*. Accessed: 2025-10-02. 2023. URL: https://262.ecma-international.org/ (cit. on p. 13).

[30] Node.js Foundation. *About Node.js*. Accessed: 2025-10-02. 2023. URL: https://nodejs.org/en/about (cit. on p. 14).

[31] MDN Web Docs. *HTML: HyperText Markup Language*. Accessed 2025-10-01. 2025. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (cit. on p. 14).

[32] WHATWG. *HTML Living Standard*. Accessed 2025-10-01. 2025. URL: https://html.spec.whatwg.org/ (cit. on p. 15).

[33] MDN Web Docs. *Structuring content with HTML*. Accessed 2025-10-01. 2025. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Structuring_content (cit. on p. 15).

[34] MDN Web Docs. *HTML: A good basis for accessibility.* Accessed 2025-10-01. 2025. URL: `https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Accessibility/HTML` (cit. on p. 15).

[35] W3C. *Web Content Accessibility Guidelines (WCAG) 2.1.* Accessed 2025-10-01. 2018. URL: `https://www.w3.org/TR/WCAG21/` (cit. on p. 15).

[36] Mark Otto and Jacob Thornton. «Bootstrap». In: First released as an open-source front-end framework by Twitter engineers. 2011. URL: `https://getbootstrap.com` (cit. on p. 16).

[37] Jake Spurlock. *Bootstrap: Responsive Web Development.* O'Reilly Media, 2013. ISBN: 978-1449335176 (cit. on p. 16).

[38] S. Shah and P. Patel. «Responsive Web Design Using Bootstrap Framework». In: *International Journal of Advance Research in Computer Science and Management Studies* 7.5 (2019), pp. 45–52. URL: `https://www.ijarcsms.com/docs/paper/Responsive-Web-Design-Using-Bootstrap-Framework.pdf` (cit. on p. 16).

[39] Yoshiki Matsuo and Hiroshi Sakamoto. «A Study on Front-End Frameworks: Comparison of Bootstrap and Others». In: *2020 International Conference on Software Engineering Research and Practice (SERP).* CSREA Press, 2020, pp. 112–118 (cit. on p. 17).

[40] *About SQLite.* `https://sqlite.org/about.html`. Accessed: 2025-10-03 (cit. on p. 17).

[41] *Features Of SQLite.* `https://sqlite.org/features.html`. Accessed: 2025-10-03 (cit. on p. 17).

[42] *SQLite: Single File Database.* `https://sqlite.org/onefile.html`. Accessed: 2025-10-03 (cit. on p. 17).

[43] K. P. Gaffney et al. «SQLite: Past, Present, and Future». In: *Proceedings of the VLDB Endowment* 15 (2022), pp. 3535–3547. DOI: `10.14778/3554821.3554842`. URL: `https://www.vldb.org/pvldb/vol15/p3535-gaffney.pdf` (cit. on p. 17).

[44] *Appropriate Uses For SQLite.* `https://sqlite.org/whentouse.html`. Accessed: 2025-10-03 (cit. on p. 17).

[45] *SQLite Frequently Asked Questions (FAQ).* `https://sqlite.org/faq.html`. Accessed: 2025-10-03 (cit. on p. 17).