Міністерство освіти і науки України
Національний Університет «Києво-Могилянська Академія»
Кафедра інформатики факультету інформатики

РОЗРОБКА ВЕБ-ЗАСТОСУВАННЯ ДЛЯ ВІДСТЕЖЕННЯ ЧАСУ ТА ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ

Текстова частина до курсової роботи

за спеціальністю 122 “Комп’ютерні науки”

Керівниця курсової роботи

Кандидатка фізико-математичних наук

Гречко А. В.

Виконала студентка 3 курсу

Гоменюк К. О.

Київ 2025

Зміст

Вступ

В умовах, де кожен день поділений на десятки завдань, а пріоритети постійно змінюються, підхід до організації власного часу та звичок виходить на перший план. Багатьом людям не вдається досягти своїх цілей саме через відсутність систематичного підходу до планування. Ефективне управління власними ресурсами та часом є важливим елементом особистісного розвитку та підтримки ментального здоров'я. Неорганізованість, перевантаження і відсутність системи у виконанні справ часто стають причинами хронічного стресу, що погіршує якість життя та продуктивність [1]. Іноді люди використовують паперові аналоги трекінгу звичок або ж календарі. Та на відміну від паперових планувальників, цифрові рішення впроваджують гнучкість, мобільність та адаптацію під індивідуальні потреби користувача. Наразі дуже актуальні веб-застосування, саме через їх зручність і можливість їх використання на власному мобільному засобі.

З наукової точки зору, тема поєднує декілька дисциплін: психологію (мотивація, формування звичок), тайм-менеджмент та інформаційні технології (розробка технічного рішення). Важливу роль у формуванні звичок відіграє створення системи, яка б підтримувала мотивацію та систематичність їх виконання [2], де і стає в нагоді веб-застосування з трекером. Тайм-менеджмент дозволяє позбутися стресу через забуті чи пропущені справи, а це однозначно позитивний вплив на здоров'я, потрібний кожному [3].

Отже розробка веб-застосування для управління часом та підвищення продуктивності має практичну значимість, оскільки передбачає створення доступного для кожної пересічної особи інструменту, що покращує якість організації та планування життєдіяльності.

Метою цієї курсової роботи є створення сучасного веб-рішення для оптимізації особистої продуктивності через комплексний підхід до управління часом та формування корисних звичок. Акцент робиться на розробку інтуїтивного інструменту, який поєднує функціонал трекінгу звичок, планувальника завдань та календаря.

Основними завданнями даної курсової роботи є:

1. аналіз існуючих систем-аналогів для формування переліку функціональніх вимог до системи;
2. проектування архітектури застосування;
3. реалізація функціональних можливостей системи;

Розроблений інструмент може знайти застосування у:

- навчальному процесі для студентів та школярів - розподіл навчального матеріалу на мікро-завдання, контроль дедлайнів;
- відстежуванні власної рутини для людей старшого віку - спрощення щоденних ритуалів, підтримка самостійності;
- для оптимізації роботи працівників у робочій та професійній сферах - працівники часто не відчувають прогресу в довготривалих проектах, а дрібні відмічені завдання створюють ілюзію легкості;
- загальному розвитку себе як особистості, завдяки впровадженню нових корисних звичок.

Це відкриває широкі перспективи для подальшого дослідження та адаптації системи під специфічні потреби різних груп користувачів.

Об’єктом дослідження є процеси планування часу та управління звичками за допомогою цифрових інструментів.

Предметом дослідження виступає веб-застосування для підвищення продуктивності, яке дозволяє користувачам керувати своїми звичками, планувати завдання чи дедлайни та загалом аналізувати свою активність.

Для реалізації було використано такі сучасні веб-технології як **Django** на серверній частині та HTML + CSS на клієнтській. Для дослідження і розробки також було використано джерела та продукти з Open source [4]. Особливу увагу приділено візуалізації та забезпеченню простого і зрозумілого інтерфейсу і розглянуто важливість UI/UX дизайну в таких додатках.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглядаються аналіз предметної області, дослідження подібних застосувань та власне постановка задачі. Другий розділ присвячений теоретичним відомостям про саму задачу, методи і підходи до її розв’язку, тощо. У третьому розділі описується процес розробки та впровадження застосування.

Розділ 1 : Аналіз предметної області

1.1 Аналіз сучасного стану питання та його рішення

Зараз багато людей, команд, навіть компаній стикаються з проблемою організації власних завдань, звичок, тощо. Це все спонукало до розвитку такого

напряму, як тайм-менеджмент. Десятиліття тому й дотепер різні спеціалісти придумують якомога ефективніші способи як підвищити продуктивність та мотивацію. Адже людство часто зіштовхується з недостатньою організацією власного часу, відсутністю мотивації. Одним з рішень для таких проблем було організовувати власні рутинні звички/завдання шляхом примітивної таблички, з днями та списком потрібних звичок. Відповідно після кожного виконаного пункту зафарбовується клітинка навпроти дати.

Звичка[5] - це автоматична повторювана поведінка, що викликається певними сигналами та виконується з невеликим усвідомленням. Звички формуються через так званий цикл звички - процес з трьох компонентів: підказки, рутини та винагороди(рис.1.1). Указівка - такий тригер, який спонукає до певної поведінки, рутина/шаблон — це сама поведінка, а винагорода - це позитивний результат, який зміцнює звичку[5].

Рисунок 1.1 - Цикл звички

І причиною чому трекери звичок так ефективно працюють є висвітлювання цього циклу як щось вимірюване. Записуючи вказівки, процедури та винагороди,

люди мають змогу визначати шаблони та свідомо змінювати свою паттерни власної поведінки.

Це свідоме залучення до циклу звичок допомагає як з формуванням нових звичок, так і з відмовою від старих. Видимість, яку забезпечує трекер звичок, перетворює абстрактні цілі на конкретні дії, що полегшує збереження відданості та відповідальності перед самим собою[5].

Робиться сильний акцент на психології, а саме впровадження механіки "маленьких перемог" – розбиття великих цілей на мікро-звички (наприклад, "читати декілька сторінок на день" замість "прочитати цілу книгу"). Людський мозок еволюційно запрограмований уникати великих, невизначених цілей через страх невдачі. Розбиття на мікро-дії робить процес більш керованим, зменшує когнітивне навантаження. Маленькі успіхи на шляху до великої мети дозволяють рухатися поступово та зберігати мотивацію[6].

Спершу, трекери реалізовувались як паперові роздруківки або ж саморобні таблиці, але з розвитком сучасних технологій почали з’являтись і додатки, сайти.

Розробка веб-застосунку для відстеження часу та підвищення продуктивності зараз актуальна через:

- візуалізацію прогресу (список що вже виконано, а що ще залишилось) допомагає користувачам краще розуміти свої плани та розбивати день на маленькі цілі;
- популярність самоорганізації, звідусіль можна почути яким важливим є персональний розвиток, здоровий спосіб життя та влаштування ефективного власного тайм-менеджменту;
- зростання інформаційного навантаження - тепер наш мозок відстежує більше даних, ніж будь-коли в історії людства. Ми переживаємо новий тип вигорання

в сучасному комп’ютеризованому світі: емоційне і когнітивне перевантаження через таку велику кількість додатків[7]. Зібравши декілька функцій в одному місці полегшить навантаження;

- культуру "завжди онлайн" - середній користувач перевіряє смартфон 58 разів на день, підкреслюючи поширеність використання смартфонів у сучасному суспільстві, що може руйнувати концентрацію[8]. Через це може знадобитись спеціальний інструмент для відслідковування своїх цілей та завдань.

На ринку існують різні рішення, але жодне з них не поєднує повноцінний календар і трекінг звичок в одному інтерфейсі. Інтеграція цих функцій в один застосунок значно спростить планування.

1.2 Дослідження аналогів

Було досліджено декілька подібних аналогів для організації та планування часу. Важливо зазначити, що у всіх були свої переваги та недоліки, але найголовніше, чого не було знайдено - це відсутність декількох опцій одночасно (тобто мати і календар, і трекер звичок замість чогось одного). Огляньмо їх.

Спершу розглянемо **Google Calendar**[9] - популярний хмарний сервіс для планування часу, який інтегрується з іншими сервісами Гугл(рис.1.2.1).

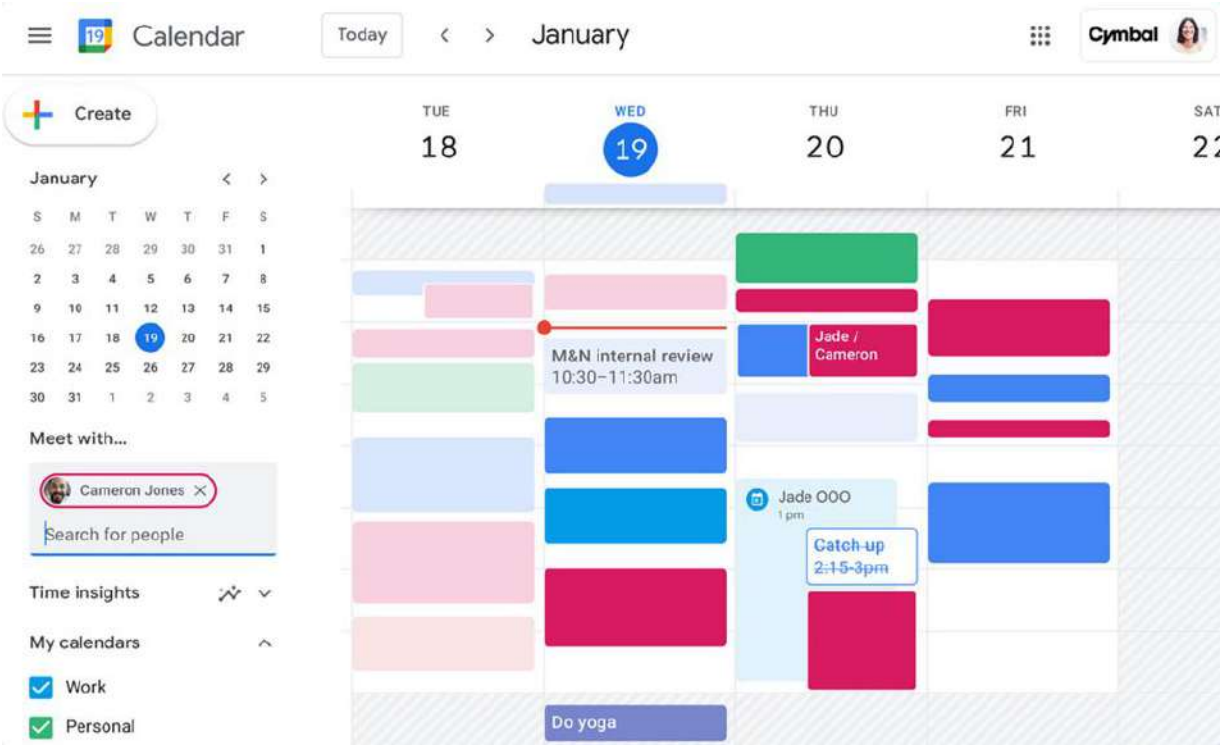

Рисунок 1.2.1 - Приклад вигляду головної сторінки Google Calendar

Його плюси:

- безкоштовний і доступний інструмент для кожного з браузера, який працює на всіх пристроях (телефон, ПК) та автоматично оновлює події;
- інтеграція з іншими сервісами від Гугл, такі як з Gmail, Meet, Google диск, тощо;
- синхронізація з іншими людьми (зручно для колег, друзів, родини)(рис.1.2.2). Поділитись своїм календарем можна змінивши статус на загальнодоступний або ж відкрити його для конкретних людей, вписавши їх пошту;

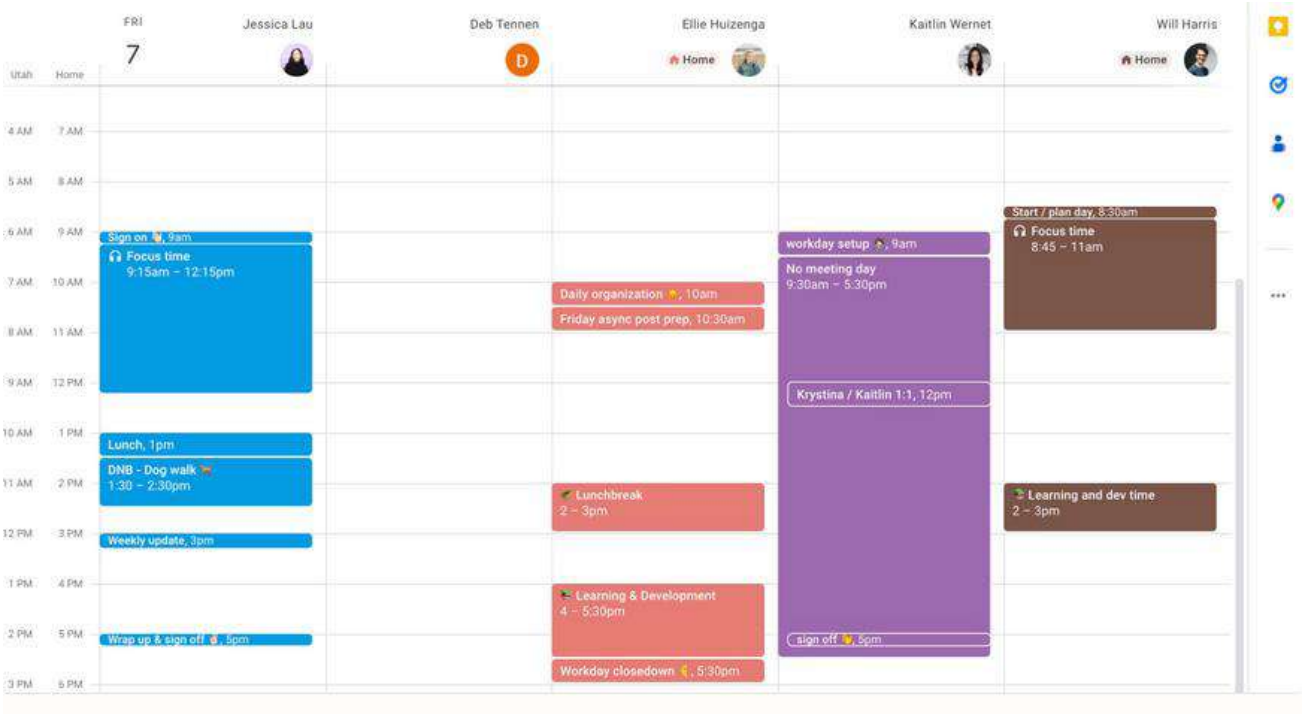

Рисунок 1.2.2 - Синхронізація Google Calendar з іншими користувачами

- багато інших додаткових функцій: нумерація тижнів для планування спринтів Agile, створення повторюваних подій, різні види перегляду (день/тиждень/місяць), налаштування сповіщень, можливість додавання різних часових поясів [9].

Та мінуси:

- деякі функції недоступні офлайн, через що - залежність від інтернету;
- реклама у безкоштовній версії;
- немає вбудованих шаблонів як от для ведення щоденника, а найголовніше - відстеження прогресу звичок.

Google Календар – універсальний інструмент, який добре підходить для базового планування, але для складніших завдань (як глибокий аналіз звичок чи Agile-дошки) знадобляться додаткові сервіси.

Наступний на розгляді **Loop** (рис.1.2.3) - безкоштовний додаток для Android з відкритим кодом на Github[10].

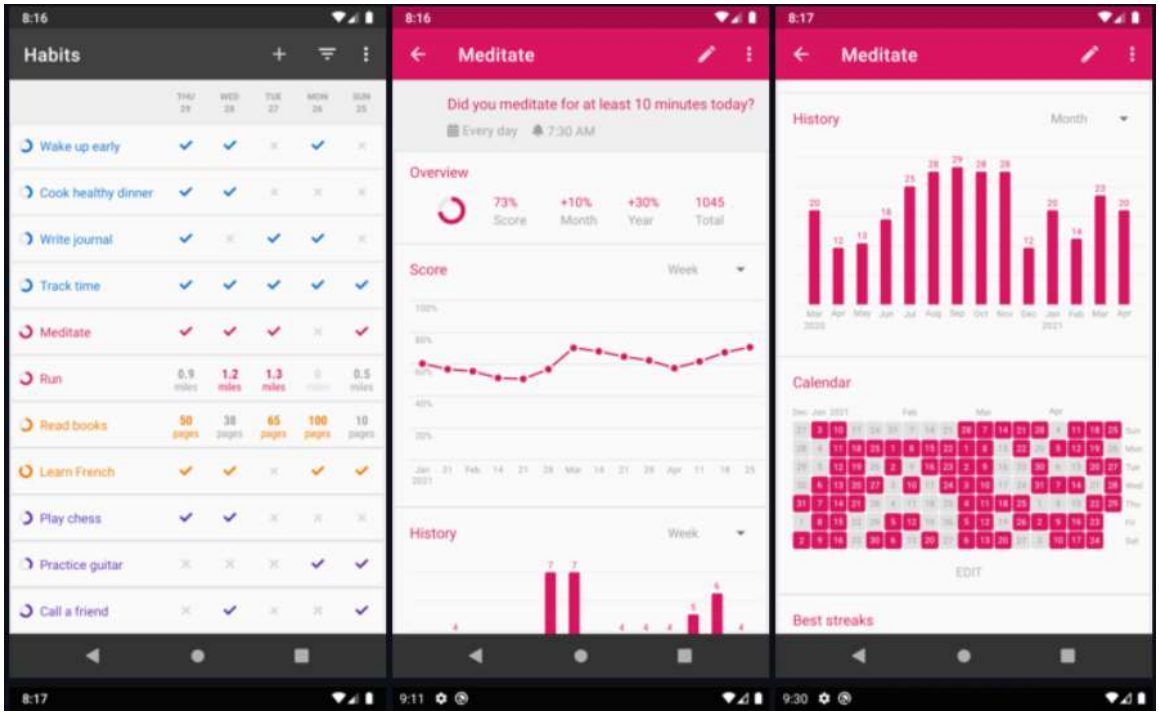

Рисунок 1.2.3 - Функціонал трекінгового додатку Loop

Плюси:

- додаток також має чудову аналітику, яка дозволяє візуалізувати прогрес з часом;
- немає реклами;
- перегляд календаря, який одразу показує, у які дні місяця ви виконали або пропустили свою звичку;
- усі дані в програмі можна легко експортувати як CSV або файл бази даних SQLite, користувач повністю володіє своїми даними;
- інтуїтивний інтерфейс: подібний до саморобних паперових трекерів, але з автоматизованим підрахунком.

Мінуси:

- ця програма недоступна для настільних ПК з iOS, Web, MacOS або Windows;
- відсутня інтеграція, лише базові сповіщення;
- якщо змінити телефон – потрібно вручну переносити резервну копію.

Підсумовуючи, Loop - дуже якісний додаток, особливо для Android-користувачів. В ньому є всі необхідні критерії для ефективного трекінгу і відслідковування звичок: можливість контролю, мінімалізм, офлайн функціонування, повна прозорість та аналітика.

Ще одним цікавим аналогом є **Habitify** (рис.1.2.4) **-** одна з найкращих міжплатформних програм для відстеження звичок[11]. Чудовою перевагою є те, що інтрумент доступний на iOS, macOS, Android і онлайн, однак це одна з небагатьох програм, яка потребує постійної підписки, щоб отримати від неї максимум можливостей.

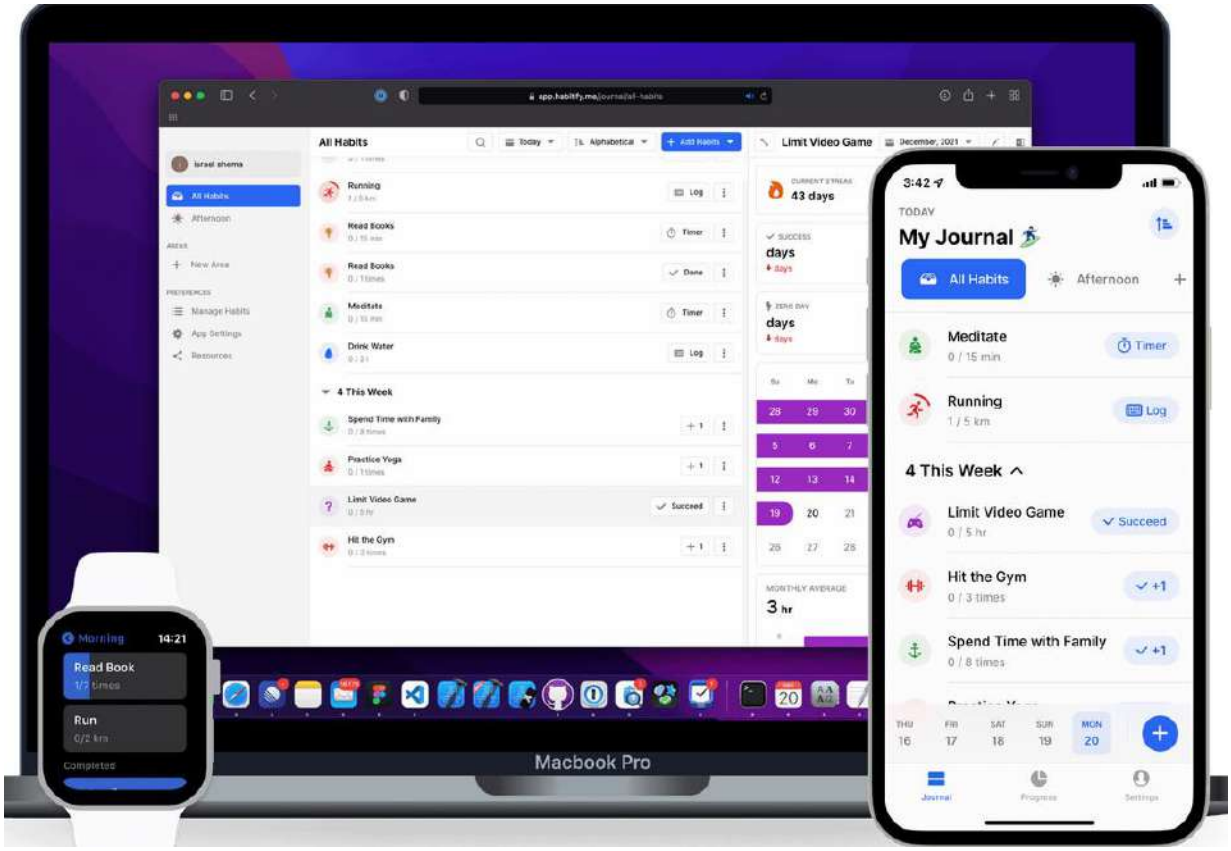

Рисунок 1.2.4 - Функціонал Habitify на різних девайсах

Плюси:

- підтримка кількох платформ (на відміну від Loop);
- календарний вигляд з кольоровим кодуванням (пропущені/виконані дні);
- також пропонує соціальні виклики (змагання). В застосунку можливо приєднатися до вже створеного челенджу, наприклад, випивати 2 літри води щодня, або ж організувати свій власний. Це надихає не здаватися та продовжувати формувати нову звичку;
- аналітика звичок: відсоток успішності, найдовші стрічки;
- синхронізація з мобільними додатками (може отримувати дані з Apple Health або Google Fit), щоб звички автоматично відслідковувалися.

Мінуси:

- напевно єдиним вагомим недоліком буде вище згадане обмеження для безкоштовної версії: максимум 5 звичок у безкоштовному режимі, наявність реклами, та інші урізання функціоналу.

Habitify підходить для тих, хто готовий інвестувати у свій інструмент для саморозвитку, однак безкоштовна версія суттєво обмежена.

Дослідження трьох інструментів для організації часу та відстеження звичок демонструє, що на ринку немає ідеального універсального рішення, яке поєднувало б повноцінний календар із потужним трекером звичок. Кожен з розглянутих додатків має свої унікальні переваги та серйозні обмеження.

1.3 Характеристики предметної області та постановка задачі

Предметна область даного веб-застосування охоплює формування звичок, управління часом та підвищення продуктивності. Вона поєднує елементи психології поведінки, тайм-менеджменту та цифрових інструментів для особистого розвитку, як було зазначено раніше.

Ключовим у виконанні звичок є ефект "галочки" – відмічені виконані завдання викликають викид дофаміну. Відмітка виконаних завдань (наприклад, зафарбовування або викреслювання в to-do списку) активізує викид дофаміну, що створює відчуття задоволення. Мозок сприймає відмічену "галочку" як завершення циклу (також про це у 1.1 підрозділі). Це простий, але потужний інструмент для продуктивності, він перетворює рутину на гру, де кожна відмітка — це маленька перемога.

У сфері управління часом та формування звичок можна виділити такі ключові сутності:

- користувач – особа, яка використовує застосунок для планування завдань або відстеження звичок;

- звичка – повторювана дія, яку користувач хоче закріпити;
- завдання – разова або періодична подія ("зустріч із друзями о 17:00");
- календар – інструмент для візуалізації подій у часі.

Одним з найголовнішим процесом у предметній області є створення звички: користувач визначає мету (наприклад, "бігати щоранку"); налаштовує параметри: час початку й кінця, частоту виконання, тощо; система генерує "цикл звички" (підказка → виконання → візуальна винагорода)

Очікування користувачів від веб-застосунку:

- простий інтерфейс – зрозумілий навіть для технічно ненавчених людей, текст чітко написаний, користувачу легко орієнтуватись у додатку;
- автоматизація – повторювані події, які й власне виробляють звички.

Мета розробки створити веб-застосунок, який поєднує календар подій і трекер звичок в єдиному інтерфейсі, а також не містить реклами та платних обмежень.

Предметна область ґрунтується на двох ключових елементах:

1. об’єкти (користувачі, звички, календар);
2. взаємозв’язки (наприклад, користувач позначає звичку як виконану).

Ця структура дозволить розробити застосунок, який не лише автоматизує рутинні дії, але й надає користувачам глибоке розуміння власної продуктивності.

Як було вже зазначено вище, застосунок буде корисний кожному, від звичайних студентів з багатьма дедлайнами і до пенсіонерів, яким наприклад, потрібно не забувати приймати регулярні ліки.

Сучасний ринок пропонує багато інструментів для планування, але жоден не вирішує всіх потреб користувачів. Запропонований веб-застосунок покликаний

заповнити цю нішу, поєднуючи трекінг звичок, календарний планувальник завдань та аналітику виконуваності. Його унікальність – в поєднанні календаря та трекеру звичок, що спростить життя багатьом людям.

Розділ 2 : Теоретичні відомості

2.1 Фреймворк Django

Створення веб-сайтів виключно на Python загалом є можливим без сторонніх інструментів, але такий спосіб вимагає значно більше часу та зусиль. Саме тому для спрощення розробки були створені спеціальні фреймворки – готові набори інструментів для швидкої побудови веб-проєктів. Одним з них є Django, фреймворк розроблений на Python, який дозволяє ефективно реалізовувати сайти будь-якої складності від простих лендінгів до масштабних платформ [12].

Django - високорівневий веб-фреймворк Python, який втілює швидку розробку та чистий, зрозумілий дизайн. Створений досвідченими розробниками, даний фреймворк абстрагує низку технічних деталей веб-розробки, оптимізуючи процес розробки та дозволяючи зосередитись на унікальній функціональності застосунку. Також фреймворк є безкоштовним і з відкритим вихідним кодом [12].

Головною перевагою Django є його вбудовані можливості, які значно прискорюють розробку. Наприклад, можливість швидко створити систему авторизації, додати форум або пошук, не розробляючи їх з нуля.

Крім того, Django працює за принципом MVC (Model-View-Controller), що спрощує організацію коду: HTML-шаблони відповідають за відображення, моделі - за взаємодію з базою даних, а контролери — за логіку зв’язку між ними (рис. 2.1).

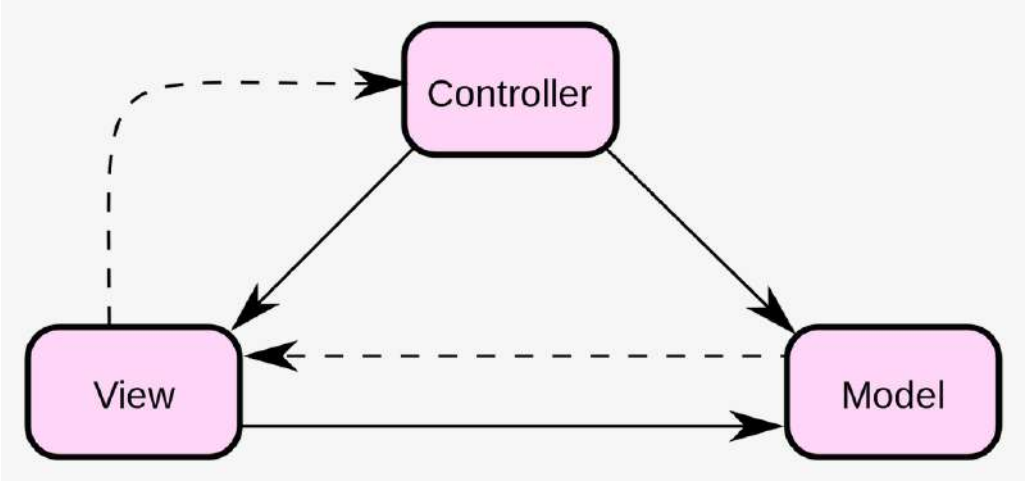

Рисунок 2.1 - Діаграма взаємодії між компонентами шаблону MVC

Django наголошує на можливості уникати дублювання коду за рахунок готових рішень та модульності, що також має назву принцип DRY (Don't Repeat Yourself) та забезпечується завдяки готових до використання функцій, такими як система входу в систему, підключення до бази даних та CRUD-операції (Create, Read, Update, Delete) [13].

Дуже багато компаній, організацій та навіть урядів використовують Django для створення різноманітних рішень, від систем управління контентом, соціальних мереж до наукових обчислювальних платформ.

Деякі з найбільш завантажених сайтів на планеті використовують здатність Django до швидкого і гнучкого масштабування, щоб задовольнити найвищі вимоги до трафіку. Наприклад:

- Google - найбільший сайт на Django, величезна пошукова система;
- YouTube - найбільший відеохостинг;
- Instagram - один з найвідоміших соц мереж;
- Pinterest - найбільший соціальний інтернет-сервіс.

Сайти, створені на Django, витримували сплески трафіку понад 50 тисяч відвідувань на секунду [14].

Коли користувач відкриває сторінку Django-застосування, система виконує наступну чітку послідовність дій:

1. при отриманні запиту Django аналізує URL через конфігурацію у файлі urls.py, після чого ініціює відповідний view-обробник, який пов’язаний із цим шляхом;
2. у поданні, розташованому в views.py, перевіряється наявність відповідних моделей;
3. моделі імпортуються з файлу models.py;
4. обробник (view) формує контекст даних та перенаправляє його до відповідного HTML-шаблону, розташованого у папці шаблонів проекту;
5. шаблон, що містить теги HTML і Django, разом з даними повертає браузеру готовий HTML-контент.

Django включає в себе десятки додаткових функцій, які можна використовувати для вирішення типових завдань веб-розробки: автентифікації користувачів, адміністрування контенту, створення мапи сайту, RSS-каналів та багатьох інших.

Django реалізує комплексний підхід до веб-безпеки, автоматично захищаючи додатки від типових загроз, серед яких SQL-ін'єкції, міжсайтовий скриптинг (XSS), підробка міжсайтових запитів (CSRF) та інші вектори атак. Власна система автентифікації забезпечує безпечне зберігання та перевірку облікових даних користувачів[12].

Наприклад, набори запитів Django захищені від SQL-ін'єкцій (такий тип атаки, коли зловмисник може виконати довільний SQL-код у базі даних), шляхом побудови з використанням параметризації запитів.

Django також надає розробникам можливість писати необроблені запити або виконувати власні sql. Однак ці можливості слід використовувати з обережністю, щоб правильно виводити будь-які параметри, які користувач може контролювати.

Суттєва перевага Django - автоматичне створення професійної адмін-панелі [15]. Не потрібно писати жодного додаткового коду - достатньо просто зареєструвати свої моделі в admin.py і система сама згенерує зручний інтерфейс для:

- додавання нових даних;
- редагування існуючих записів;
- видалення або фільтрації об’єктів.

Це ідеальне рішення для швидкого управління контентом, тестування даних або навіть повноцінної роботи адміністраторів сайту. Як результат, є готовий до продакшену інструмент, що заощаджує години рутинної розробки.

Саме завдяки таким перевагам Django залишається одним із найвідоміших фреймворків для веб-розробки.

2.2 Мова шаблонів Django

Як веб-фреймворк, Django потребує зручного способу динамічної генерації HTML. Найпоширенішим підходом є використання шаблонів. Шаблон містить статичні частини бажаного HTML-виводу, а також деякий спеціальний синтаксис, що описує, як буде вставлено динамічний вміст.

У Django передбачена можливість використання одного, декількох шаблонізаторів або повна відмова від них за відсутності потреби у шаблонах.

Django постачається з вбудованим бекендом для власної системи шаблонів, яку називають мовою шаблонів Django (DTL).

До версії Django 1.8 це був єдиний доступний вбудований варіант. Це хороша бібліотека шаблонів, хоча і має деякі особливості [16]. Якщо немає нагальних

причин обирати інший бекенд, варто використовувати DTL, особливо при написанні застосування, що підтримує підключення та розповсюдження своїх шаблонів. Django contrib-застосування, які містять шаблони, наприклад, django.contrib.admin, використовують DTL.

Бекенди для інших мов шаблонів можуть бути доступні від сторонніх розробників. Власний бекенд шаблонів у Django реалізує стандартний API, який забезпечує завантаження та рендеринг шаблонів незалежно від конкретного бекенду. Процес завантаження передбачає пошук шаблону за певним ідентифікатором і його попередню обробку, зазвичай компіляцію у внутрішнє представлення в пам'яті. Рендеринг полягає у підстановці (інтерполяції) даних із контексту в шаблон та поверненні готового рядка.

Django використовує механізм пошуку шаблонів, що дозволяє уникнути дублювання. Система послідовно перевіряє каталоги, вказані в параметрі DIRS: якщо шаблон відсутній у першій папці, пошук продовжується в наступних, поки файл не буде знайдено [15].

Під час рендерингу шаблону система підставляє змінні значеннями з переданого контексту та виконує спеціальні теги шаблонізатора, тоді як решта контенту залишається без змін.

Сам шаблон у Django — це текстовий файл або рядок коду на Python, який містить спеціальні позначки мови шаблонів Django. Деякі конструкції розпізнаються та інтерпретуються механізмом шаблонів.

Під час рендерингу шаблону система поєднує його з контекстом - підставляє значення змінних з контексту, виконує теги, а всі інші елементи залишає незмінними.

Синтаксис мови шаблонів Django включає чотири конструкції [16]:

1. **Змінні**

Щоб вивести дані в шаблоні, Django використовує змінні. Він бере їх із спеціального словника даних, який передається під час рендерингу шаблону. Змінні оточені подвійними фігурними дужками{{ }} (рис. 2.2.1)

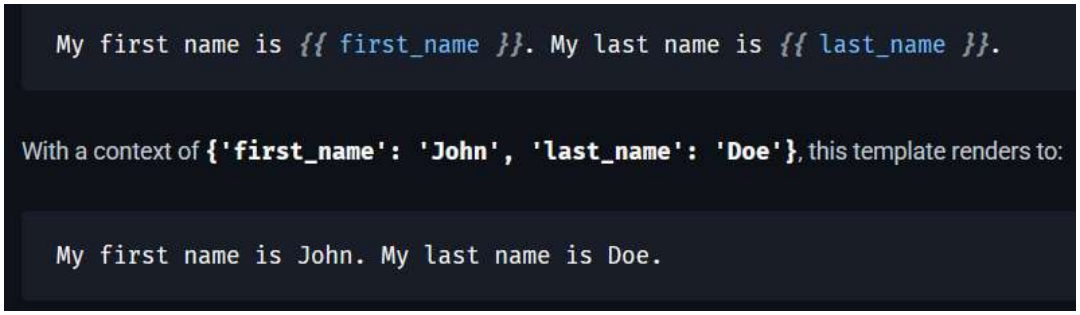

Рисунок 2.2.1 - Синтаксис змінних у мові шаблонів Django

2. Теги

Теги забезпечують довільну логіку в процесі рендерингу. Визначення не є сильно точним, адже тег може виводити вміст, слугувати керуючою структурою, наприклад, оператором «if» або циклом «for», отримувати вміст з бази даних або навіть відкривати доступ до інших тегів шаблону. Теги оточені символами {% і %} (рис.2.2.2). Більшість тегів приймають аргументи, а для деяких потрібні початковий та кінцевий теги (рис.2.2.3).

```
{% cycle 'odd' 'even' %}
```

Рисунок 2.2.2 - Синтаксис тегів у мові шаблонів Django

```
{% if user.is_authenticated %}Hello, {{ user.username }}.{% endif %}
```

Рисунок 2.2.3 - Приклад початкового і кінцевого тегів

3. Фільтри

Фільтри перетворюють значення змінних та аргументів тегів і будуються за принципом {{ змінна|фільтр }} (рис.2.2.4). Деякі з них приймають аргумент (рис.2.2.5).

```
{{ django|title }}
```

```
{{ my_date|date:"Y-m-d" }}
```

Рисунок 2.2.4 та 2.2.5 - Синтаксис фільтрів у мові шаблонів Django

4. Коментарі

Коментарі в мові шаблонів Django обмежені символами {# і #} (рис.2.2.6)

```
{# this won't be rendered #}
```

Рисунок 2.2.6 - Синтаксис коментарів у мові шаблонів Django

Тег {% коментар %} забезпечує багаторядкові коментарі.

Кожна з цих конструкцій виконує чітко визначену функцію в системі шаблонів Django, забезпечуючи гнучкість у відокремленні логіки від представлення даних. Така архітектура дозволяє ефективно комбінувати HTML-розмітку з динамічним контентом, зберігаючи при цьому чистоту коду.

2.3 Особливості архітектури Django-застосунку

Більшість сучасних веб-застосувань використовують трирівневу архітектуру. У Django-застосунку зазвичай реалізовано наступний базовий потік запитів та взаємодія з клієнтом і зовнішніми сервісами (рис.2.3).

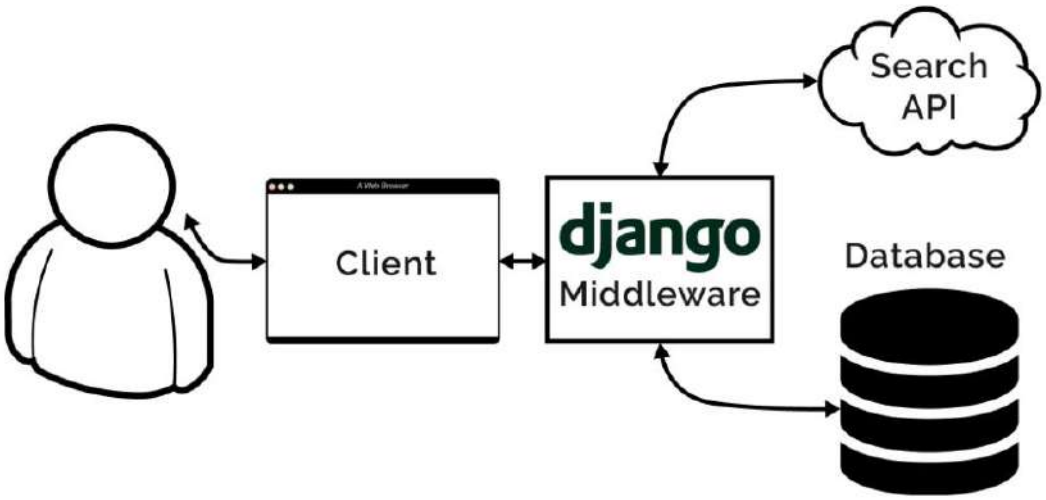

Рисунок 2.3 - Потік запитів у Django-застосунку

Мережева взаємодія в Django ґрунтується на обміні даними між клієнтом та сервером з використанням протоколів HTTP, WebSocket або інших. Фреймворк забезпечує механізми для обробки API-запитів, формування відповідей, а також реалізацію аутентифікації та захисту даних.

Обробка HTTP-запитів на Django відбувається наступним чином:

1. маршрутизація - Django URLconf визначає, яка функція обробить запит;
2. View - відповідний обробник запиту (функція чи клас) виконує логіку;
3. Model - View взаємодіє з базою даних через ORM (Object-Relational Mapping);
4. Template - дані передаються шаблон для формування відповіді;
5. сформована HTTP-відповідь надсилається клієнту.

Також критично важлива частина розробки, яка безпосередньо впливає на структуру проекту та його стійкість є Virtual Environment (віртуальне середовище)[15]. Це таке ізольоване середовище для Python-проектів, де встановлюються залежності (пакети), специфічні для поточного застосунку.

Іншими словами це аналог "контейнера" для бібліотек, який унеможливлює конфлікти версій між різними проектами, адже різні проекти можуть потребувати різних версій Django чи бібліотек. Додатково це допомагає уникнути конфліктів з

глобальним Python-середовищем, оновлення бібліотек у одному проекті не зламають інші. Це гарантує, що програмне забезпечення, встановлене в одному середовищі, не впливає на програмне забезпечення, встановлене в іншому.

Система маршрутизації є ключовою частиною архітектури будь-якого веб-застосунку. У Django це views - функції Python, які приймають http-запити і повертають http-відповіді, як HTML-документи.

Зазвичай вони зберігаються у файлі views.py, розташованому в папці додатку. Це "карта" застосунку, яка пов’язує URL-адреси з функціями чи класами. Також визначає, яка логіка (View) виконується для кожного запиту і забезпечує чистоту та модульність коду (наприклад, розділення URL між додатками).

Розділ 3 : Опис реалізації програмного продукта

3.1 Обґрунтування алгоритму й структури програми

Тільки зареєстровані користувачі зможуть користуватись застосунком, а відвідувачі сайту повинні мати можливість зареєструвати обліковий запис. Після успішної автентифікації надається доступ до особистого кабінету.

Користувач може додавати нові звички, вказуючи їхні параметри (назва, періодичність, ціль тощо). Програма автоматично генерує завдання (tasks) на основі вказаної періодичності. Користувач може переглядати або видаляти створені звички.

Відстеження виконання відбувається наступним чином: користувач відмічає виконання завдань, а програма оновлює дані про поточний страйк (серію виконаних завдань поспіль) та загальну статистику. Користувач також може додавати події до календаря або видаляти їх. Події відображаються у вигляді інтерактивного календаря.

Дана структура програми та алгоритм є логічними і достатніми для вирішення поставлених завдань. Розбиття на модулі дозволяє легко масштабувати проект, додавати нові функції. Використання Django забезпечує надійність і безпеку, а інтуїтивний інтерфейс робить програму зручною для користувачів.

Програма розділена на декілька модулів, де кожен відповідає за певні функціональні частини:

- модуль habit, в якому знаходяться моделі: Habit, TaskTracker, Streak. Вони відповідальні за зберігання даних про звички, завдання, страйки;
- модуль Users відповідає за автентифікацію та управління користувачами;
- модуль Habit_Tracker є ядром проєкту. Він об’єднує всі частини системи через файл settings.py (де прописані підключені додатки через INSTALLED_APPS) та керує глобальною маршрутизацією (urls.py). У цьому модулі також розміщено конфігураційні файли для запуску (asgi.py, wsgi.py) та обробки запитів (views.py).

Проєкт використовує SQLite як базу даних за замовчуванням, а також підтримує структуру шаблонів і статичних файлів. Такий поділ коду підвищує читабельність, повторне використання компонентів та забезпечує зручну інтеграцію нових функцій.

Схема бази даних системи застосування (рис.3.1, таблиця 1) має наступні зв’язки:

- один користувач має багато звичок;
- кожна звичка має багато записів про виконання;
- один користувач має багато подій у календарі.

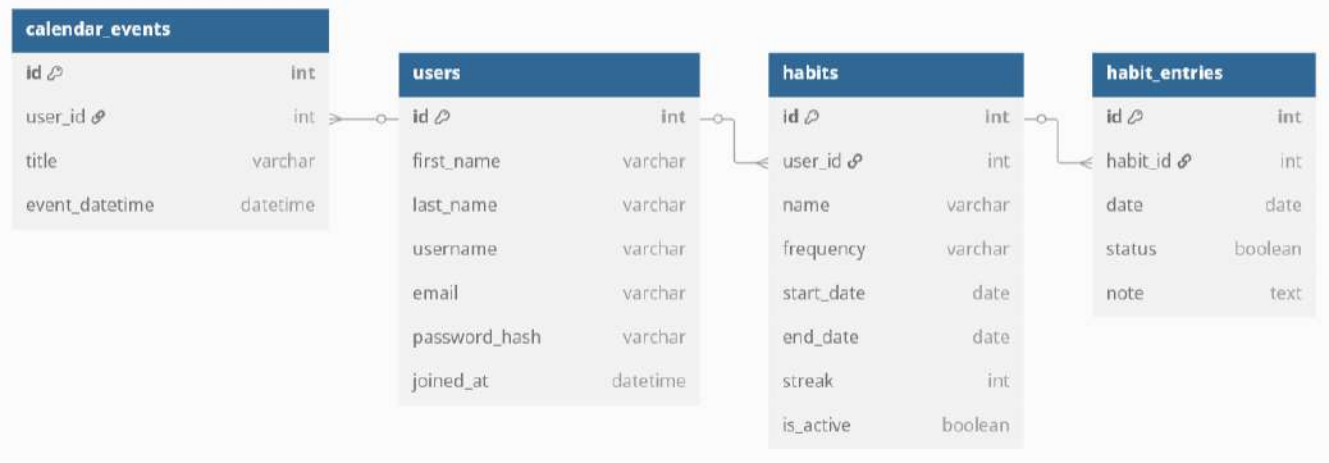

Рисунок 3.1 - Схема БД системи

Сутність	Поле	Тип даних	Опис
Користувачі (users)	id	int	Ідентифікатор користувача
	first_name	varchar	Ім’я користувача
	last_name	varchar	Прізвище користувача
	username	varchar	Логін
	email	varchar	Електронна пошта
	password_hash	varchar	Пароль
	joined_at	datetime	Дата реєстрації
Звички (habits)	id	int	Ідентифікатор звички
	user_id	int	Зовнішній ключ на таблицю users
	name	varchar	Назва звички
	frequency	varchar	Частота виконання (наприклад: тричі на тиждень)
	start_date	datetime	Дата початку
	end_date	datetime	Дата дедлайну
	streak	int	Кількість днів безперервного виконання
	is_active	boolean	Активна/неактивна звичка (тобто чи вже час її виконати)
Записи звичок (habit_entries)	id	int	Ідентифікатор запису
	habit_id	int	Зовнішній ключ на таблицю habits
	date	datetime	Дата запису
	status	boolean	Статус (виконано або ні)

Календар подій (calendar_events)	id	int	Ідентифікатор події
	user_id	int	Зовнішній ключ на таблицю users
	title	varchar	Назва події
	event_datetime	datetime	Дата і час події

Таблиця 1 - Обгрунтування схеми бази даних з описом українською мовою

3.2 Налаштування програми

Файл settings.py є основним конфігураційним файлом Django-проекту, який визначає глобальні параметри роботи додатку, підключення до бази даних, безпеку, маршрутизацію статичних файлів та інші критичні налаштування. У проекті цей файл відіграє ключову роль у забезпеченні функціонування всіх компонентів системи.

В ньому знаходяться шаблони, статика, медіа, наприклад:

- TEMPLATES: Шляхи до HTML-шаблонів;
- STATIC_URL/MEDIA_URL: Для CSS/JS та завантажених файлів.

А також ще багато конфігурацій Django-проекту:

- AUTH_PASSWORD_VALIDATORS: Валідація паролів;
- LOGIN_REDIRECT_URL: Перенаправлення після входу;
- BASE_DIR: Шлях до кореня проекту;
- INSTALLED_APPS: завантажені додатки, бібліотеки, розширення.

3.3 Опис розробки моделей програми

Моделі програми розписані у файлі models.py та є основою бази даних у Django-застосуванні. Файл визначає структуру даних, зв’язки між сутностями та бізнес-логіку, пов’язану зі збереженням інформації. У проекті цей файл містить усі

моделі, необхідні для роботи зі звичками, завданнями, серію виконаних завдань поспіль, тощо.

У models.py загалом реалізовано 4 класи-моделі, розгляньмо їх:

1. Модель Habit (рис.3.3.1)

   Відповідає за звички користувача. Ключовими атрибутами є:

   name – назва звички. frequency – кількість повторень за період (наприклад, "2 рази на тиждень"). goal – ціль у днях (наприклад, "90 днів"). completion_date – розрахункова дата завершення звички.

   А також метод: save() – автоматично розраховує num_of_tasks (кількість завдань) та completion_date при збереженні

```python
class Habit(models.Model):

    name = models.CharField(max_length=255)
    frequency = models.IntegerField(default= 1)
    period = models.CharField(max_length=255)
    goal = models.IntegerField(default=90)
    num_of_tasks = models.IntegerField()
    notes = models.CharField(max_length=255, default=None)
    creation_time = models.DateTimeField(auto_now_add=True)
    start_date = models.DateTimeField(null=True, blank=True)
    completion_date = models.DateTimeField(null=True, blank=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE, null = True)


    def save(self, *args, **kwargs):
        """
        Overrides the save method to perform calculations before saving the object.
        Calculates the number of tasks required to achieve the habit's goal based on the specified period
        and frequency. If the completion date is not provided, it calculates it based on the goal.
        """

        self.name = self.name.lower()

        # Calculate the completion date based on the goal
        if not self.completion_date:
            self.completion_date = self.start_date + timedelta(days=self.goal)

        # Calculate the number of tasks needed to achieve the habit goal
        num_of_period = convert_period_to_days(self.period)
        if not self.num_of_tasks:
            self.num_of_tasks = (self.goal // num_of_period) * self.frequency


        super().save(*args, **kwargs)
```

Рисунок 3.3.1 - Реаліація моделі Habit

2. Модель TaskTracker

   Відстежує завдання для кожної звички. Ключові атрибути:

   due_date – дедлайн завдання.
   task_status – статус ("In progress", "Completed" або "Failed").

   Методи:

   create_tasks() – автоматично генерує завдання при створенні звички.
   update_failed_tasks() – позначає прострочені завдання як "Failed".

3. Модель Streak фіксує серії успішних виконань звички.

Ключові атрибути:

current_streak – поточна серія.

longest_streak – найдовша серія.

Важливий метод: save() – оновлює longest_streak, у випадку, якщо поточний страйк більший.

4. Модель Event (рис.3.3.2)

   Використовується для календаря подій. Ключові атрибути:

   start, end – час події.

```python
class Event(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    start = models.DateTimeField()
    end = models.DateTimeField()
    color = models.CharField(max_length=7, default='#378006')
    all_day = models.BooleanField(default=False)

    def __str__(self):
        return f"{self.title} ({self.start.date()})"
```

Рисунок 3.3.2 - Реалізація моделі Event

3.4 Календар

Календар у застосуванні для відстеження звичок служить ключовим інструментом для візуалізації власних планів користувачів, для відслідковування дедлайнів та різноманітних подій.

Для реалізації календаря було обрано бібліотеку FullCalendar – інструмент з відкритим кодом [17], який забезпечує:

- інтерактивність (drag-and-drop, клік для створення подій);
- підтримку локалізації (в даному випадку українська мова);

- зручний API для роботи з подіями.

За замовчуванням календар має місячний вигляд (dayGridMonth), але також доступні тижневий (timeGridWeek) та денний (timeGridDay) види. Разом з цим чудова підтримка навігації - перемикання місяців/тижнів та кнопка "Сьогодні" (рис.3.4).

Події підвантажуються динамічно при завантаженні сторінки за допомогою fetch GET-запиту з URL /habit/api/events/. Видалення реалізується через eventClick + fetch DELETE.

Django виконує чудову роль як бекенд, він використовується для зберігання подій у моделі Event (поля: title, start, end, user). Синхронізація фронтенду та бекенду через використання JsonResponse для обміну даними у форматі JSON.

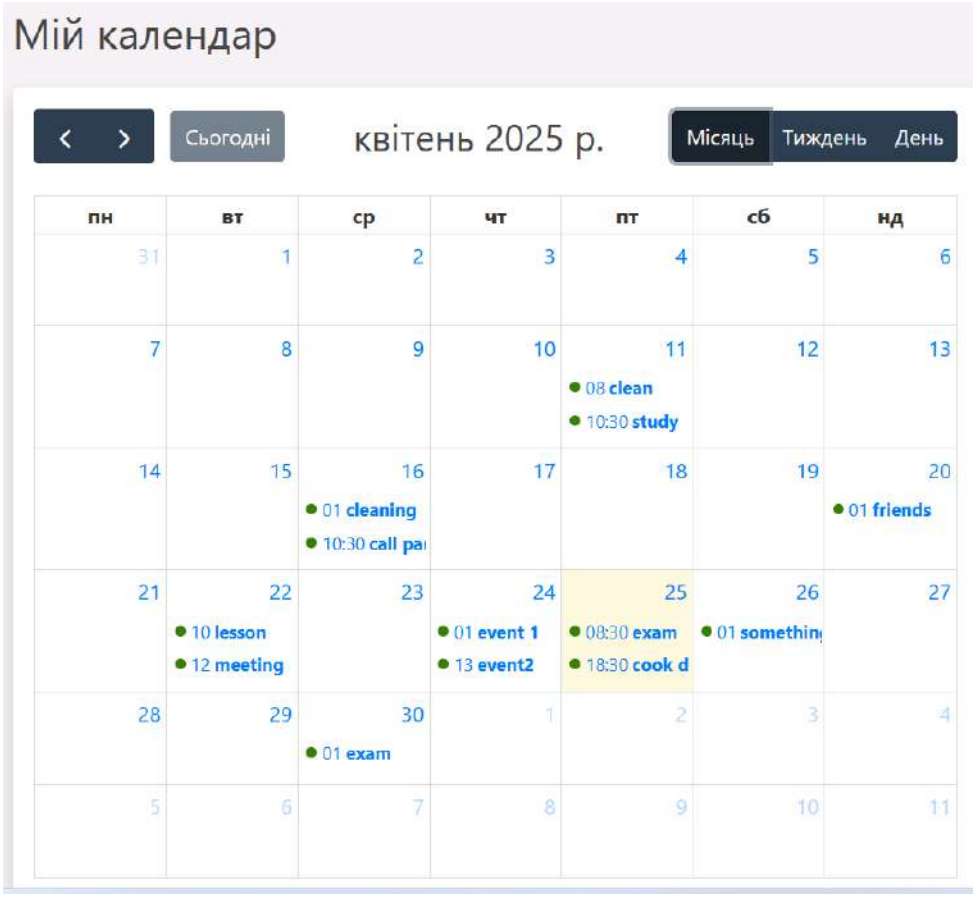

Рисунок 3.4 - Вигляд запланованих подій в календарі

Переваги такої реалізації:

- легке управління звичками/подіями;
- асинхронність: не потрібно перезавантажувати сторінку;
- можливість для розширення – можна додати повторення подій, сповіщення, категорії та ще багато чого.

3.5 Опис інтерфейсу програми

Інтерфейс розроблений з використанням Django шаблонів та Bootstrap для забезпечення зручності та адаптивності.

Після автентифікації користувач бачить головну сторінку додатку, яка складається з привітання, основного меню навігації звичок (додавання, управління та аналіз), а також посилання на сторінку з календарем (рис.3.5.1). Ім’я користувача в заголовку створює ефект індивідуального підходу та дуже зручно угруповані задачі за статусом (сьогодні/активні/майбутні).

Рисунок 3.5.1 - Головна сторінка застосування

У розділі Habits Manager (рис. 3.5.2) кожна звичка представлена у вигляді картки з ключовою інформацією про тип, дати термінів, страйку, прогрес та ще багато іншого.

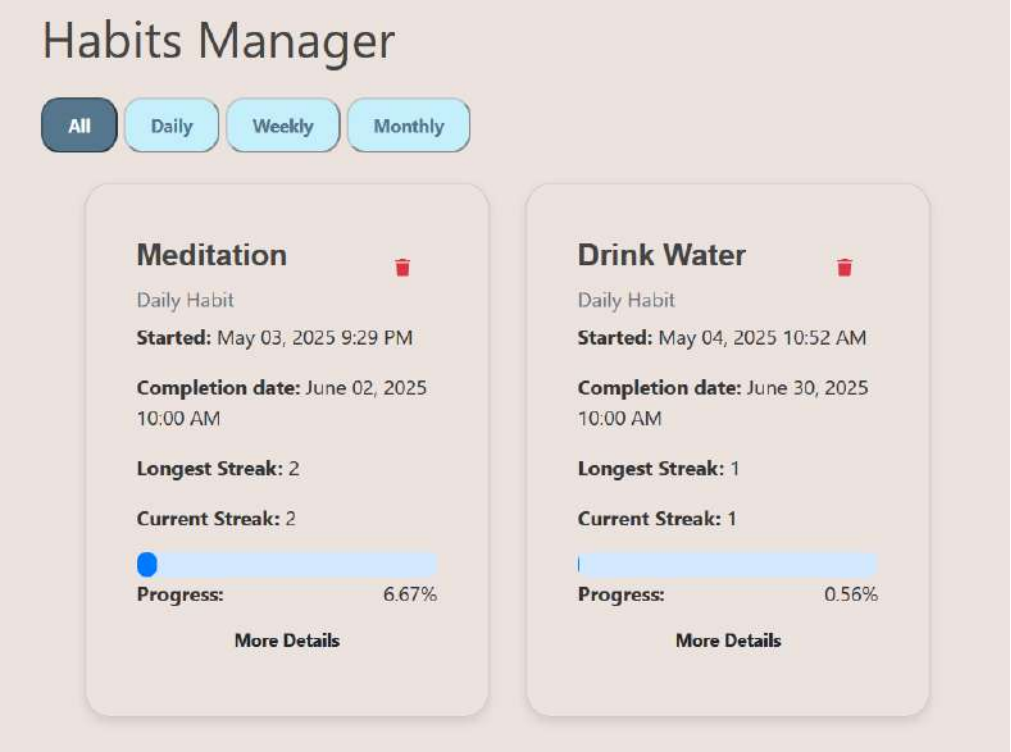

Рисунок 3.5.2 - Вигляд звичок у Habit manager

Коли з’являється звичка, яку потрібно відмітити, вона випливає таким блоком на головному екрані (рис.5.3.3). Виділена кнопка відразу кидається в очі, чітко прописана назва і зрозуміла постановка задачі. Також бачимо дедлайн та вид самої звички (наприклад, денна).

Рисунок 5.3.3 - Вигляд блок завдання/звички

Підсумовуючи, цей інтерфейс можна назвати інтуїтивним, адже присутні чітка навігація, контрастні кнопки, читабельний текст та простір. Усі елементи (форми, таблиці, календар) оптимізовані для зручного використання.

Висновки

У ході виконання курсової роботи було розроблено Django-застосування для відстеження звичок та планів, яке дозволяє користувачам ставити цілі, фіксувати прогрес та допомагає дотримуватися регулярності. Крім того, було досліджено та проведено аналіз різноманітних інструментів для розробки веб-застосувань.

У першому розділі ретельно досліджено сучасні підходи до формування звичок, проаналізовано аналоги подібних застосувань та визначено ключові функціоальні можливості, необхідні для ефективного трекера. На основі цього сформовано вимоги до системи.

Другий розділ присвячений ознайомленню з описом основних можливостей використаних інструментів, обраних для реалізації системи. Досліджено всі нюанси фреймворку Django, які стали у нагоді для розробки даного застосування.

У третьому розділі описано логіку та функціональні можливості розробленого застосування. Пояснюється як влаштований бекенд та фронтенд проєкту, які є файли, модулі, методи та за що вони відповідальні.

Робота демонструє ефективне застосування Django для створення повноцінного веб-застосування зі зручним інтерфейсом та корисним функціоналом. Результати можуть бути використані як основа для подальшого вдосконалення або ж інтеграції в більші системи.

Список використаної літератури

1. Психічне здоров'я на роботі: як його зберегти [Електронний ресурс] // Центр громадського здоров'я МОЗ України. – URL: https://phc.org.ua/news/psikhichne-zdorovya-na-roboti-yak-yogo-zberegti.
2. Гальмаков Ю. Коли трекер звичок стає непотрібним. URL: https://galmakov.com/treker-nepotriben/.
3. Тайм-менеджмент: що таке тайм-менеджмент та його важливість під час роботи. Worksection [Електронний ресурс] URL: https://worksection.com/ua/blog/importance-of-time-management-in-the-workplace.html.
4. Django Habit Tracker. URL: https://github.com/UsfZA/Habit-Tracker.
5. Самодисципліна та звичка [Електронний ресурс] // Chytay.ua. – URL: https://chytay-ua.com/blog.php?id=464.
6. Мистецтво встановлення цілей – шлях до успіху [Електронний ресурс] // Drukarnia.com.ua. – URL: https://drukarnia.com.ua/articles/mistectvo-vstanovlennya-cilei-shlyakh-do-uspikhu-seTRZ.
7. Rutkowski, A.-F., Saunders, C. S. Growing Pains with Information Overload [Електронний ресурс] // Communications of the ACM. — 2010. – URL: https://repository.tilburguniversity.edu/server/api/core/bitstreams/993aa3a0-2f56-4d35-a32a-0d3094afc93e/content.
8. Smartphone Usage Statistics [Електронний ресурс] // Priori Data. – URL: https://prioridata.com/data/smartphone-usage-statistics/.
9. Все можуть календарі: 12 функцій Google Календаря [Електронний ресурс] // Laba. – 2023. – URL: https://laba.ua/blog/2954-vse-mozhut-kalendari-12-funkciy-google-kalendarya.
10. Loop Habit Tracker. URL: https://github.com/iSoron/uhabits/
11. Best Habit Tracker Apps [Електронний ресурс] // Zapier. – URL: https://zapier.com/blog/best-habit-tracker-app/#habitify.

12. Django Software Foundation. Django Project [Електронний ресурс]. – URL: https://www.djangoproject.com/.
13. Вступ до Django [Електронний ресурс] // W3Schools. – URL: https://www.w3schools.com/django/django_intro.php.
14. Курс Django (українською) [Електронний ресурс] // ITProger. – URL: https://itproger.com/ua/course/django.
15. Azzopardi L., Maxwell D. Tango With Django 2. Leanpub, 2020.
16. Django Templates [Електронний ресурс] // Django documentation. – URL: https://docs.djangoproject.com/en/5.2/topics/templates/.
17. FullCalendar: JavaScript Event Calendar [Електронний ресурс]. – URL: https://fullcalendar.io/.

Додаток А - частина коду, що стосується бекенду

Зміст файлу Habit_Tracker\urls.py, де знаходяться всі URL-маршрути:

```
from .views import calendar_view
from django.contrib import admin
from django.contrib.auth import views as auth_views
from django.urls import path
from Users import views as user_views
from habit import views as habit_views
from django.conf import settings
from django.conf.urls.static import static
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from habit.views import EventViewSet
from django.contrib.staticfiles.storage import staticfiles_storage
from django.views.generic.base import RedirectView
from habit import views
from habit import views as habit_views

router = DefaultRouter()
router.register(r'events', EventViewSet)

urlpatterns = [
    path('admin/', admin.site.urls),
    path('Login', auth_views.LoginView.as_view(template_name='Users/login.html'), name='login'),
    path('Register/', user_views.register, name='register'),
```

```
    path('Profile/', user_views.profile, name='profile'),
    path('Logout/', auth_views.LogoutView.as_view(template_name='Users/logout.html'), name='logout'),

    path('', habit_views.HabitView.as_view(), name='habit-home'),

    path('Add-Habit/', habit_views.HabitManagerView.add_habit, name='habit_creation'),
    path('delete-habit/<int:habit_id>/', habit_views.HabitManagerView.delete_habit, name='habit_deletion'),
    path('Habit-Manager/', habit_views.HabitManagerView.active_habits, name = 'active_habits'),
    path('Habit-Infos/<int:habit_id>/', habit_views.HabitManagerView.habit_detail, name='habit_detail'),

    path('Habits-Analysis/', habit_views.HabitAnalysis.as_view(), name = 'HabitsAnalysis'),
    path('api/', include(router.urls)),
    path('calendar/', calendar_view, name='calendar'),
    path('favicon.ico', RedirectView.as_view(url=staticfiles_storage.url('img/favicon.ico'))),
    path('habit/api/events/', views.calendar_events, name='calendar_events'),
    path('habit/api/events/<int:event_id>/', views.calendar_events, name='calendar_event_detail'),
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Додаток Б - частина коду, що стосується фронтенду

Вигляд \templates\base.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">
    <link rel="stylesheet" type="text/css" href="{% static 'css/main.css' %}">
    <script src="{% static 'fontawesomefree/js/all.min.js' %}"></script>
    <link href="{% static 'fontawesomefree/css/all.min.css' %}" rel="stylesheet" type="text/css">
    {% if title %}
        <title>Habit Tracker - {{ title }}</title>
    {% else %}
        <title>Habit Tracker</title>
    {% endif %}
</head>
<body>
    <header class="site-header">
        <nav class="navbar navbar-expand-md navbar-dark bg-steel fixed-top">
            <div class="container">
                <a class="navbar-brand mr-4" href="{% url 'habit-home' %}"><i class="fa-solid fa-house"></i>    Habit Tracker
</a>
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarToggle"
                aria-controls="navbarToggle" aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
```

```
            <div class="collapse navbar-collapse" id="navbarToggle">
              <div class="navbar-nav ml-auto">
                <div class="nav-item dropdown">
                  <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-
toggle="dropdown"
                  aria-haspopup="true" aria-expanded="false">
                    Menu
                  </a>
                  <div class="dropdown-menu" aria-labelledby="navbarDropdown">
                    <a class="dropdown-item" href="{% url 'profile' %}">Profile</a>
                    <div class="dropdown-divider"></div>
                    <a class="dropdown-item" href="{% url 'logout' %}">Logout</a>
                  </div>
                </div>
              </div>
            </div>
          </nav>
        </header>
        <main role="main" class="container">
          <div class="row">
            <div class="col-md-8">
              {% if messages %}
                {% for message in messages %}
                  <div class="alert alert-{{ message.tags }}">
                    {{ message }}
                  </div>
                {% endfor %}
              {% endif %}
              {% block content %}{% endblock %}
            </div>
          </div>
        </main>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datepicker/1.9.0/js/bootstrap-datepicker.min.js"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>
</body>
</html>
```