

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Вычислительные машины, комплексы, системы и
сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

ВЕБ-ПРИЛОЖЕНИЕ УПРАВЛЕНИЯ
ЗАДАЧАМИ С ПОВЫШЕННОЙ
СИСТЕМОЙ БЕЗОПАСНОСТИ

Студент Левин 23.05.26 Левин Артём Дмитриевич
(подпись, дата) (ФИО полностью)
Руководитель Токарев 23.05.26 Токарев А.Н.
(подпись, дата) (фамилия, инициалы)
Нормоконтролёр Бычков 23.05.26 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой Митрохин Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от
Никитин 23.05.26 № 4)

Секретарь ГЭК Гурин Гурин Е.И.
(подпись) (фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 07 » 04 20 26 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Лёвин Артём Дмитриевич гр. 22BBB1 факультета ФВТ

2. Тема работы: Веб-приложение управления задачами с повышенной системой безопасности

Тема утверждена приказом ПГУ № 347/0 от " 07 " 04 20 26 г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Токарев А.Н.

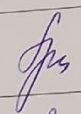
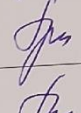
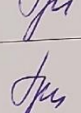
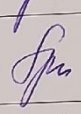
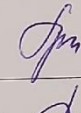
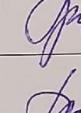
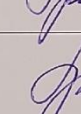
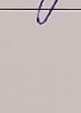
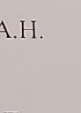
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать веб-приложение управления задачами с повышенной системой безопасности. Приложение должно обеспечивать регистрацию и аутентификацию пользователей, двухфакторную аутентификацию на основе TOTP, генерацию одноразовых резервных кодов, защиту от перебора паролей с временной блокировкой учётной записи, безопасное управление сессиями, а также разграничение ролей. Функционал управления задачами должен включать создание, редактирование, удаление, просмотр и фильтрацию задач по статусу. Администратор дополнительно должен иметь возможность просматривать все задачи всех пользователей, создавать задачи для других пользователей, сбрасывать пароль и отключать двухфакторную аутентификацию любому пользователю.

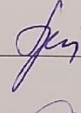
5. Перечень подлежащих разработке вопросов

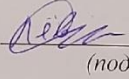
Произвести анализ существующих систем управления задачами и выявить их недостатки с точки зрения безопасности учётных записей, обосновать актуальность разработки собственного решения. Определить назначение, функциональные и нефункциональные требования, а также общий состав модулей веб-приложения. Выполнить выбор инструментов разработки. Разработать архитектуру приложения. Разработать программное обеспечение: модуль аутентификации с двухфакторной защитой, модуль управления задачами с ролевым разграничением, механизмы защиты от перебора паролей и безопасного управления сессиями. Произвести проверку работы приложения, включая тестирование устойчивости к атакам перебора, корректности работы 2FA и резервных кодов, изоляции данных между пользователями, и оценить соответствие разработанного решения поставленным требованиям безопасности.

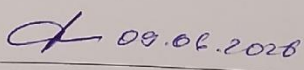
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	30	07.04.2026 – 10.04.2026	
2	Анализ аналогов, обоснование актуальности, постановка задачи	24	10.04.2026 – 13.04.2026	
3	Выбор и настройка инструментов разработки	8	13.04.2026 – 14.04.2026	
4	Проектирование архитектуры приложения	40	14.04.2026 – 20.04.2026	
5	Разработки модуля аутентификации и двухфакторной защиты	60	20.04.2026 – 28.04.2026	
6	Разработка модуля управления задачами и ролевого разграничения	65	28.04.2026 – 07.05.2026	
7	Реализация дополнительных механизмов безопасности	55	07.05.2026 – 14.05.2026	
8	Тестирование и отладка	20	14.05.2026 – 18.05.2026	
9	Оформление пояснительной записки	20	18.05.2026 – 21.05.2026	

Дата выдачи задания " 07 " 04 20 26 г.

Руководитель бакалаврской работы  Токарев А.Н.
(подпись, дата)

Задание к исполнению принял студент  Лёвин А.Д.
(подпись, дата)

Работу к защите допустить
Декан факультета  08.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 77 листов, 39 рисунков, 1 таблицу, 7 источников, 3 приложения.

АУТЕНТИФИКАЦИЯ, ВЕБ-ПРИЛОЖЕНИЕ, ЗАЩИТА ДАННЫХ, КОНТРОЛЬ ДОСТУПА, УПРАВЛЕНИЕ ЗАДАЧАМИ, DJANGO, PYTHON

Цель работы – разработать веб-приложение управления задачами с повышенной системой безопасности.

Объектом разработки является веб-приложение управления задачами с повышенной системой безопасности, включающей в себя: двухфакторную аутентификацию, защиту от перебора паролей, резервные коды и ролевую модель доступа.

Разработка проводилась в среде программирования PyCharm Community Edition. В качестве языка программирования выбран был Python и веб-фреймворк Django.

В результате проведённой работы разработано веб-приложение, удовлетворяющее поставленной задаче, а именно: удобный интерфейс, возможность создания, редактирования и фильтрации задач, разграничение прав доступа между пользователем и администратором, а также реализованы механизмы защиты учётных данных.

Созданное приложение может быть использовано в учебных целях для изучения современных методов безопасности веб-приложений, а также в качестве основы для доработки в небольших организациях.

					ПГУ 1.090301.07.001 ПЗ			
Изм.	Лист	№ докум.	Подпи	Дата	Веб-приложение управления задачами с повышенной системой безопасности Пояснительная записка	Лит.	Лист	Листов
Разраб.		Лёвин А.Д.	<i>Лёвин</i>	26.05.26			2	77
Провер.		Токарев А.Н.	<i>Токарев</i>	26.05.26				
Н. Контр.		Бычков А.С.	<i>Бычков</i>	26.05.26				
Утверд.		Митрохин М.А.	<i>Митрохин</i>	26.05.26				
						ФВТ, каф. ВТ гр. 22BVB1		

Содержание

Перечень сокращений и обозначений.....	5
Введение	6
1 Анализ предметной области.....	8
1.1 Анализ систем управления задачами	8
1.2 Анализ актуальности обеспечения безопасности веб-приложений	10
1.3 Анализ существующих решений	13
1.4 Постановка задачи разработки	16
2 Реализация.....	21
2.1 Описание архитектуры проекта	21
2.2 Выбор инструментов разработки	24
2.3 Установка и настройка инструментов разработки	26
2.4 Описание структуры программы	33
2.5 Разработка программы.....	37
2.5.1 Реализация аутентификации и двухфакторной защиты	38
2.5.2 Реализация управления задачами.....	38
2.5.3 Реализация механизмов безопасности	39
2.5.4 Реализация вспомогательных модулей.....	39
2.6 Описание доступных функций.....	40
2.6.1 Домашняя страница	41
2.6.2 Регистрация и аутентификация	41
2.6.3 Двухфакторная аутентификация (2FA)	42
2.6.4 Навигация и общий вид интерфейса.....	43
2.6.5 Управление задачами (CRUD)	43
2.6.6 Разграничение прав доступа.....	45
2.6.7 Защита от перебора паролей	46
2.6.8 Смена пароля.....	47
2.6.9 Сброс двухфакторной аутентификации.....	47
2.6.10 Выход из системы и завершение сессии.....	47

3 Анализ результатов	48
3.1 Результаты разработки приложения	48
3.2 Рекомендации к дальнейшей разработке	50
4 Безопасность жизнедеятельности	53
4.1 Правила безопасности при использовании ПК	53
4.2 Пожарная безопасность	53
Заключение	55
Список использованных источников	56
Приложение А – UML-Диаграммы	57
Приложение Б – Листинг программы	59
Приложение В – Презентация	75

Перечень сокращений и обозначений

В настоящей работе применяются следующие сокращения и обозначения

2FA – двухфакторная аутентификация (Two-Factor Authentication)

Brute force – метод атаки полным перебором паролей (брутфорс)

Cookie – небольшой фрагмент данных, сохраняемый браузером и используемый, в частности, для поддержки сессий

CRUD – набор базовых операций с данными: создание, чтение, обновление, удаление (Create, Read, Update, Delete)

CSRF – межсайтовая подделка запроса (Cross-Site Request Forgery), вид атаки на веб-приложения

HTTP – протокол передачи гипертекста (HyperText Transfer Protocol), лежащий в основе обмена данными в веб-приложениях

MVT – архитектурный шаблон «модель-представление-шаблон» (Model-View-Template), применяемый, в частности, в Django

Open-source – программное обеспечение с открытым исходным кодом

ORM – объектно-реляционное отображение (Object-Relational Mapping), технология связи между объектной моделью и реляционной базой данных

OWASP Top 10 – список десяти наиболее критических угроз безопасности веб-приложений, публикуемый Open Web Application Security Project

PBKDF2 – функция генерации ключа на основе пароля с солью и многократным хешированием (Password-Based Key Derivation Function 2)

RBAC – управление доступом на основе ролей (Role-Based Access Control)

RFC 6238 – стандарт IETF, описывающий алгоритм TOTP

SSO-провайдер – провайдер технологии единого входа (Single Sign-On), позволяющий однократно аутентифицироваться для доступа к нескольким системам

TOTP – алгоритм генерации одноразовых паролей на основе текущего времени (Time-based One-Time Password)

Введение

Веб-приложения для управления задачами стали неотъемлемым инструментом в профессиональной и личной деятельности, позволяя эффективно планировать, распределять и контролировать выполнение работ. Однако широкое распространение таких сервисов сопровождается ростом кибератак, направленных на компрометацию учётных записей пользователей. Согласно статистике, атаки перебора паролей (brute force) и фишинг остаются одними из самых распространённых способов несанкционированного доступа. Большинство популярных систем (Trello, Asana, Jira) предоставляют расширенные механизмы безопасности только в платных версиях, либо не реализуют их в полном объёме. Open-source аналоги (Vikunja) часто ограничиваются базовой аутентификацией и не предлагают встроенных средств защиты от подбора паролей или резервных кодов для восстановления доступа. В связи с этим актуальной является разработка веб-приложения для управления задачами, которое с самого начала проектируется с учётом повышенных требований к безопасности учётных записей.

Существующие открытые системы управления задачами предоставляют базовые функции: создание задач, назначение исполнителей, отслеживание статусов. Однако механизмы двухфакторной аутентификации (TOTP по RFC 6238), автоматической временной блокировки при превышении числа неудачных попыток входа, инвалидации сессий при смене пароля или 2FA, а также разграничение прав в них отсутствуют, или же реализованы частично. Коммерческие продукты с полным набором функций имеют закрытый исходный код и требуют платной подписки, что затрудняет их изучение и адаптацию. Поэтому создание собственного приложения с открытой реализацией указанных механизмов является теоретически и практически обоснованным.

Разработанное веб-приложение может быть использовано в учебных целях для изучения современных механизмов защиты веб-приложений. Кроме того, проект может служить основой для доработки или внедрения в небольших

					ПГУ 1.090301.07.001 ПЗ	Лист
						6
Изм.	Лист	№докум.	Подпись	Дата		

организациях, где требуется надёжный инструмент управления задачами с повышенными требованиями к безопасности учётных записей.

Для достижения цели поставлены следующие задачи:

- провести анализ предметной области систем управления задачами и выявить основные угрозы безопасности веб-приложений;
- выполнить обзор существующих open-source и коммерческих решений, оценить их соответствие требованиям повышенной безопасности;
- спроектировать архитектуру веб-приложения и структуру базы данных с учётом хеширования паролей средствами Django, хранения TOTP-секретов, резервных кодов;
- реализовать модуль аутентификации, включающий: стандартный вход по паролю (хеширование с солью), двухфакторную аутентификацию на основе TOTP, генерацию и проверку одноразовых резервных кодов, защиту от перебора паролей;
- реализовать модуль управления задачами с базовыми операциями (создание, редактирование, удаление, просмотр) для пользователя, а также расширенными возможностями для администратора (просмотр всех задач всех пользователей, создание задач для других пользователей, фильтрация по пользователям и статусам задач, сброс пароля и 2FA у конкретного пользователя);
- провести тестирование разработанного приложения, включая проверку устойчивости к атакам перебора, корректности работы 2FA и резервных кодов, изоляции данных между пользователями и правильности разграничения ролей;
- оценить полученные результаты и определить направления дальнейшего развития проекта.

1 Анализ предметной области

1.1 Анализ систем управления задачами

Системы управления задачами представляют собой программные решения, предназначенные для создания, хранения, отслеживания и контроля выполнения задач. Основная цель таких систем – упорядочить рабочие процессы, повысить прозрачность выполнения работ и обеспечить соблюдение сроков. В зависимости от области применения они могут использоваться как отдельными пользователями, так и командами.

По типу использования выделяют три основные категории:

- индивидуальные системы (Todoist, Any.do) ориентированы на одного пользователя, не содержат механизмов разграничения доступа или ролевой модели. Основная функция – ведение личного списка дел;
- корпоративные системы (Trello, Asana, Jira, Vikunja) поддерживают многопользовательскую работу, назначение исполнителей, общие доски проектов, комментарии, уведомления. В таких системах уже присутствует базовая ролевая модель (частник, наблюдатель, администратор);
- специализированные системы – это коммерческие продукты с закрытым кодом, либо разрабатываемые решения, которые с самого начала включают механизмы защиты учётных записей. Именно к такому типу относится система, разрабатываемая в рамках данной работы.

Независимо от типа, любая система управления задачами реализует базовый набор операций с задачами: создание, чтение, редактирование, удаление (CRUD).

Каждая задача обычно содержит следующие атрибуты (сущности):

- название – краткий заголовок задачи;
- описание – детальная информация;
- статус – например, «к выполнению», «в работе», «выполнено», «отложено»;

- дедлайн– дата и время, к которым задача должна быть выполнена.

При разработке подобных систем необходимо учитывать три группы требований: функциональные, нефункциональные и требования безопасности.

Функциональные требования:

- создание, редактирование, удаление, просмотр задач;
- контроль статуса;
- фильтрация задач по статусу, исполнителю, дедлайну;
- разграничение доступа: обычный пользователь работает только со своими задачами, администратор – со всеми.

Нефункциональные требования:

- удобство и интуитивность интерфейса;
- надёжность;
- поддержка многопользовательской работы.

Требования повышенной безопасности:

- двухфакторная аутентификация (2FA) по временному одноразовому паролю (TOTP, RFC 6238) – дополнительный фактор помимо логина и пароля;
- защита от атак перебора паролей (brute force) – ограничение количества неудачных попыток входа с временной блокировкой учётной записи;
- резервные коды – одноразовые коды для восстановления доступа при утере второго фактора или смене пароля;
- безопасное управление сессиями – автоматическое завершение всех сессий пользователя при смене пароля, смене 2FA или при сбросе учётных данных администратором;
- хеширование паролей с солью;
- ролевая модель доступа (RBAC) с разделением на «пользователя» и «администратора», где администратор может сбросить пароль или 2FA другому пользователю.

При разработке системы принимаются следующие допущения:

					ПГУ 1.090301.07.001 ПЗ	Лист
						9
Изм.	Лист	№докум.	Подпись	Дата		

- в качестве метода двухфакторной аутентификации используется только TOTP;
- не рассматриваются вопросы сетевой безопасности – предполагается, что приложение разворачивается в защищённой среде;
- журнал аудита в данной версии не реализуется, но может быть добавлен в будущем.

Проведённый анализ показывает, что традиционные системы управления задачами (индивидуальные и большинство командных) вообще не содержат механизмов защиты учётных записей, а также ограничиваются базовой аутентификацией по паролю. Однако современные угрозы требуют внедрения дополнительных мер безопасности. Следовательно, для обеспечения действительно надёжного решения необходимо разработать систему, которая с самого начала проектируется с учётом этих требований. В следующем подразделе будут подробно рассмотрены актуальные угрозы безопасности веб-приложений и обоснована необходимость применения указанных механизмов.

1.2 Анализ актуальности обеспечения безопасности веб-приложений

Развитие информационных технологий сопровождается не только расширением возможностей программных систем, но и увеличением количества угроз, связанных с их использованием. Веб-приложения являются одной из наиболее распространённых форм программных решений и при этом становятся частыми объектами атак злоумышленников. Это обусловлено тем, что такие системы обеспечивают доступ к пользовательским данным через сеть Интернет, что делает их потенциально уязвимыми.

Одной из ключевых проблем является рост числа инцидентов, связанных с утечкой данных. Согласно отчёту компании Positive Technologies за 2023 год, количество утечек персональных данных в России увеличилось более чем на 40% по сравнению с предыдущим годом, причём более 60% инцидентов приходится на веб-приложения и онлайн-сервисы. Как отмечается в том же отчёте,

«...самыми желанными объектами для атак киберпреступников остаются госучреждения, компании из промышленной и финансовой сферы. 20% пострадавших от действий киберпреступников входят в рейтинг крупнейших компаний России по объёму реализации продукции RAEX-600 за 2022 год.»[1]. Злоумышленники похищают учётные записи, пароли, адреса электронной почты, а иногда и содержимое закрытых документов.

Для систем управления задачами утечка данных особенно опасна, поскольку такие системы хранят не только логины и пароли, но и описание рабочих задач, дедлайны, комментарии, а иногда и коммерческую или личную информацию. Утечка этих сведений может привести к нарушению конфиденциальности проектов, шантажу или несанкционированному изменению планов.

Среди наиболее распространённых угроз выделяются:

- атаки перебора паролей – при отсутствии ограничения числа попыток злоумышленник может автоматически перебирать комбинации символов, пока не подберёт пароль. Этот тип атак входит в OWASP Top 10 как одна из основных уязвимостей аутентификации;

- фишинговые атаки – пользователь самостоятельно передаёт свои учётные данные на поддельной странице входа. В этом случае даже сложный пароль не защищает;

- компрометация баз данных – при взломе сервера злоумышленник получает хеши паролей, и если в системе нет защиты от перебора, он может восстановить исходные пароли.

Несколько случаев, когда недостаток двухфакторной аутентификации или защиты от перебора привёл к серьёзным последствиям:

- в июле 2020 года злоумышленники взломали аккаунты в Twitter известных политиков и бизнесменов. Атака была совершена через сотрудников – злоумышленники получили доступ к внутренним инструментам администрирования, на которых отсутствовала двухфакторная аутентификация. В результате были опубликованы мошеннические сообщения с просьбой

перевести биткойны. Этот инцидент показал, что даже крупные IT-компании уязвимы без дополнительной защиты учётных записей;

– в апреле 2021 года в открытом доступе оказались данные 533 миллионов пользователей Facebook: номера телефонов, имена, даты рождения, а в некоторых случаях и хеши паролей. Хотя пароли были захешированы, часть из них удалось подобрать с помощью атак перебора из-за отсутствия ограничения попыток входа на момент утечки. Этот случай демонстрирует, что даже хеширование не спасает, если система не защищена от автоматизированного подбора;

– в 2022 году произошла утечка базы данных сервиса с хешами паролей пользователей «Яндекс.Еда». После этого злоумышленники предприняли массовые атаки перебора, и многие учётные записи были украдены. Система не имела встроенной защиты от многократных неудачных попыток входа. Пользователям пришлось в срочном порядке менять пароли, а сервис понёс репутационные потери.

Приведённые примеры подтверждают, что одной парольной защиты явно недостаточно.

По данным Verizon Data Breach Investigations Report (2023), более 80% взломов, связанных с веб-приложениями, используют скомпрометированные учётные данные. Как прямо указано в том же отчёте, «86% of the breaches... involve the Use of stolen credentials. And where better to use those credentials than against the various web servers that contain our sensitive information» [2]. Внедрение двухфакторной аутентификации снижает риск несанкционированного доступа, но не всегда является спасением.

Важным аспектом является соблюдение требований законодательства в области защиты персональных данных. В Российской Федерации действует Федеральный закон №152-ФЗ «О персональных данных», который устанавливает требования к обработке и защите информации, относящейся к пользователям. В соответствии с законом организации и разработчики программных систем обязаны обеспечивать защиту персональных данных от несанкционированного

доступа, утраты и иных угроз. Хотя разрабатываемое приложение носит учебный характер, учёт данных требований позволяет сформировать правильный подход к разработке безопасных систем.

На основании вышеизложенного, обеспечение безопасности веб-приложений является актуальной и важной задачей. Рост числа утечек данных, реальные инциденты и фундаментальные недостатки парольной защиты доказывают необходимость внедрения дополнительных механизмов.

1.3 Анализ существующих решений

Целью данного подраздела является оценка существующих систем управления задачами с точки зрения реализованных в них механизмов безопасности, а также выявление их недостатков, которые могут быть устранены в рамках собственной разработки. Анализ проводится по следующим критериям: поддержка двухфакторной аутентификации, способ хранения паролей, защита от перебора, наличие резервных кодов для восстановления доступа, шифрование данных в базе данных, аудит действий пользователей, а также открытость кода для изучения.

Для анализа выбраны четыре системы, представляющие разные сегменты: закрытые массовые решения (Trello, Asana, Jira) и open-source система с фокусом на безопасность (Vikunja).

Trello – веб-приложение на основе канбан-досок. Отличается простотой и наглядностью, но нет защиты от перебора паролей и резервных кодов. Код закрыт, что не позволяет изучать внутреннюю реализацию безопасности. Обзор на рисунке 1.1



Рисунок 1.1 – Обзор аналога «Trello»

Asana — система для управления проектами с расширенными возможностями (списки, доски, календарь). Поддерживает 2FA только через сторонние SSO-провайдеры (Single Sign-On), не предоставляет резервных кодов. Аудит действий доступен только в платной версии. Обзор на рисунке 1.2.

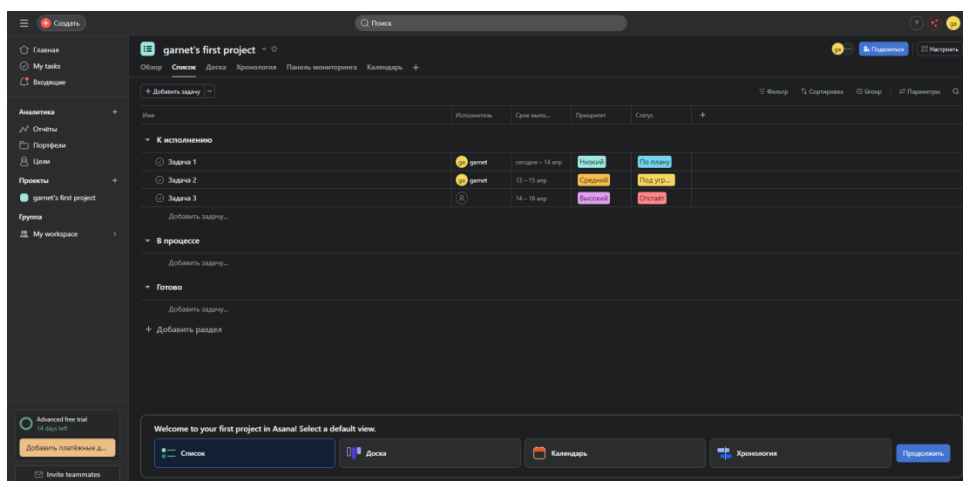


Рисунок 1.2 – Обзор аналога «Asana»

Jira — мощное решение для разработки ПО с гибкой настройкой рабочих процессов. Поддерживает 2FA через Atlassian Access. Система сложна в настройке и имеет закрытый код. Обзор на рисунке 1.3.

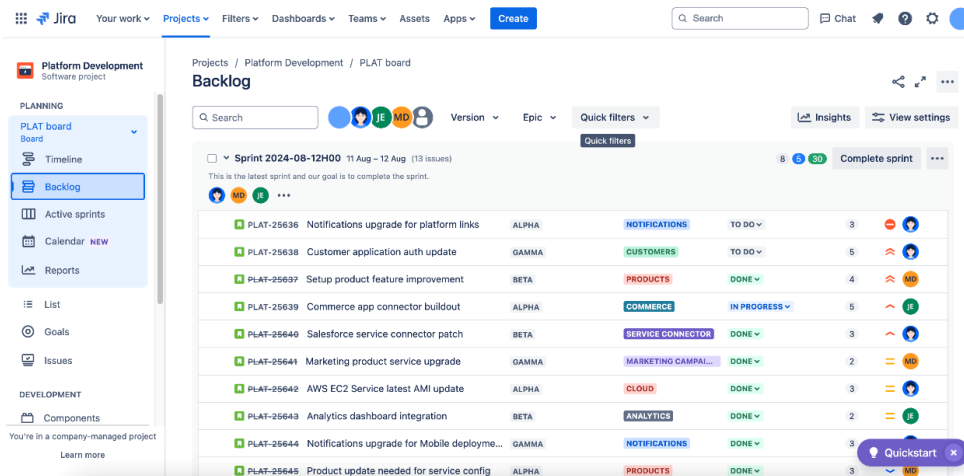


Рисунок 1.3 – Обзор аналога «Jira»

Vikunja – open-source система управления задачами с поддержкой 2FA, ролевой моделью и возможностью самостоятельного хостинга. Однако не имеет шифрования отдельных полей задач и защиты от перебора паролей. Обзор на рисунке 1.4.

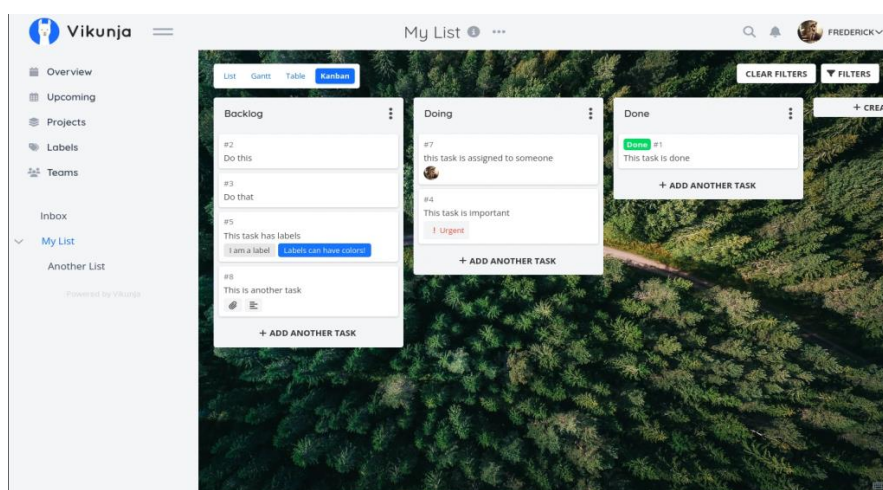


Рисунок 1.4 – Обзор аналога «Vikunja»

В таблице 1.1 приведены основные характеристики систем. Для каждой указаны ключевые преимущества и недостатки.

Таблица 1.1 – Сравнение существующих систем управления задачами.

Название аналога	Преимущества аналога	Недостатки аналога
Trello	Наглядность (канбан-	Закрытый код, аудит в

	доски), бесплатная базовая версия	платной версии, нет 2FA и защиты от перебора
Asana	Расширенный функционал(списки, доски, зависимости задач), поддержка командной работы	Поддерживает 2FA только через сторонние SSO-провайдеры, сложный интерфейс для новичков
Jira	Встроенный аудит, гибкая настройка, есть защиты от перебора	Избыточность для малых проектов, система сложна в настройке
Vikunja	Открытый код, самостоятельный хостинг, присутствует ролевая модель, есть 2FA	Отсутствует защита от перебора пароля, нет шифрования полей задач

Как видно из таблицы, ни одна из рассмотренных систем не удовлетворяет всем требованиям повышенной безопасности в рамках бесплатного и доступного для изучения решения.

Проведённый анализ показал, что существующие популярные системы управления задачами не обладают полным набором необходимых механизмов безопасности, предоставляют их только в платных версиях с закрытым кодом. Open-source альтернативы имеют ограниченный функционал безопасности.

Следовательно, целесообразна разработка собственного веб-приложения для управления задачами, которое с самого начала проектируется с учётом повышенных требований безопасности. Такое решение позволит не только получить рабочий инструмент, но и на практике изучить современные методы защиты веб-приложений.

1.4 Постановка задачи разработки

На основе проведённого анализа систем управления задачами и сравнения существующих аналогов можно сделать вывод, что современные системы управления задачами не обеспечивают необходимого уровня безопасности учётных записей в бесплатных версиях или имеют закрытый исходный код, что не

позволяет изучать и адаптировать механизмы защиты. Открытые аналоги поддерживают двухфакторную аутентификацию, но не реализуют защиту от перебора паролей, резервные коды и шифрование данных.

Таким образом, актуальной является разработка собственного веб-приложения для управления задачами, которое с самого начала проектируется с учётом повышенных требований безопасности и одновременно служит практической платформой для изучения современных методов защиты веб-приложений.

Целью разработки является создание веб-приложения для управления задачами, обеспечивающего:

- базовый функционал работы с задачами (создание, редактирование, удаление, просмотр, фильтрация);
- разграничение ролей (обычный пользователь, администратор);
- повышенную безопасность учётных записей за счёт двухфакторной аутентификации, резервных кодов, защиты от перебора паролей, безопасного управления сессиями и шифрования конфиденциальных полей.

Требования разделены на три группы: функциональные, требования безопасности и нефункциональные.

Функциональные требования:

- регистрация новых пользователей с проверкой уникальности логина;
- аутентификация пользователя по логину и паролю с последующим запросом одноразового кода 2FA, если он включён;
- управление задачами для аутентифицированного пользователя: создание задачи (название, описание, статус, дедлайн), редактирование задачи, удаление задачи, просмотр списка своих задач с возможностью фильтрации по статусам;
- разграничение прав доступа на основе ролей: обычный пользователь видит, создаёт, редактирует и удаляет только свои задачи, а администратор дополнительно может: просматривать все задачи всех пользователей, создавать задачу для любого пользователя, фильтровать задачи по пользователям и

статусам, сбросить пароль любому пользователю, сбросить двухфакторную аутентификацию пользователя;

- возможность смены пароля пользователем с обязательным завершением всех его сессий;

- возможность сброса и повторной регистрации двухфакторной аутентификации пользователем, а также повторной генерации резервных кодов.

Для наглядного представления функциональных требований и разграничения ролей разработана диаграмма прецедентов, показанная на рисунке 1.5. На диаграмме выделены два актора: «Пользователь» и «Администратор». Администратор наследует все возможности обычного пользователя и так же дополнительно получает права на пользование прочим функционалом.



Рисунок 1.5 – Диаграмма прецедентов

Требования безопасности:

- хранение паролей – хеширование с солью с использованием алгоритма PBKDF2;
- двухфакторная аутентификация – реализация стандарта TOTP (RFC 6238) с генерацией 6-значных одноразовых кодов, обновляемых каждые 30 секунд. Для генерации и проверки кодов используется библиотека pyotp;

– резервные коды – при включении 2FA генерируются 8 одноразовых кодов. Каждый код может быть использован только один раз. Резервные коды также применяются при смене пароля;

– защита от перебора паролей – ограничение числа неудачных попыток входа: после 5 неудачных попыток учётная запись блокируется на 10 минут. Счётчик попыток сбрасывается после успешного входа или истечения времени блокировки;

– управление сессиями;

– изоляция данных – все запросы к задачам проверяют принадлежность задачи текущему пользователю. Прямой доступ к URL чужой задачи должен блокироваться.

Нефункциональные требования:

– технологический стек – бэкенд на языке Python 3.10+ с использованием веб-фреймворка Django, база данных SQLite. Фронтенд – стандартные HTML-шаблоны Django, так же фреймворк Bootstrap для быстрой вёрстки адаптивных сайтов;

– интерфейс – веб-интерфейс должен быть интуитивно понятным. Основные действия (создание задачи, фильтрация) должны быть доступны с главной страницы.

В результате выполнения работы должно быть создано функционирующее веб-приложение, удовлетворяющее перечисленным требованиям. Приложение должно быть развёрнуто на локальном сервере.

2 Реализация

2.1 Описание архитектуры проекта

На основе требований, сформулированных в разделе 1.4, и в соответствии с реализованным кодом, архитектура веб-приложения для управления задачами с повышенной безопасностью построена по классической клиент-серверной модели. Клиентом выступает веб-браузер пользователя, который отображает HTML-страницы, генерируемые сервером, отправляет HTTP-запросы (GET, POST). Серверная часть реализована на языке Python с использованием веб-фреймворка Django и включает в себя представления, модели и маршруты. Все данные сохраняются в базе данных SQLite. На рисунке 2.1 представлена общая, упрощённая схема взаимодействия компонентов.

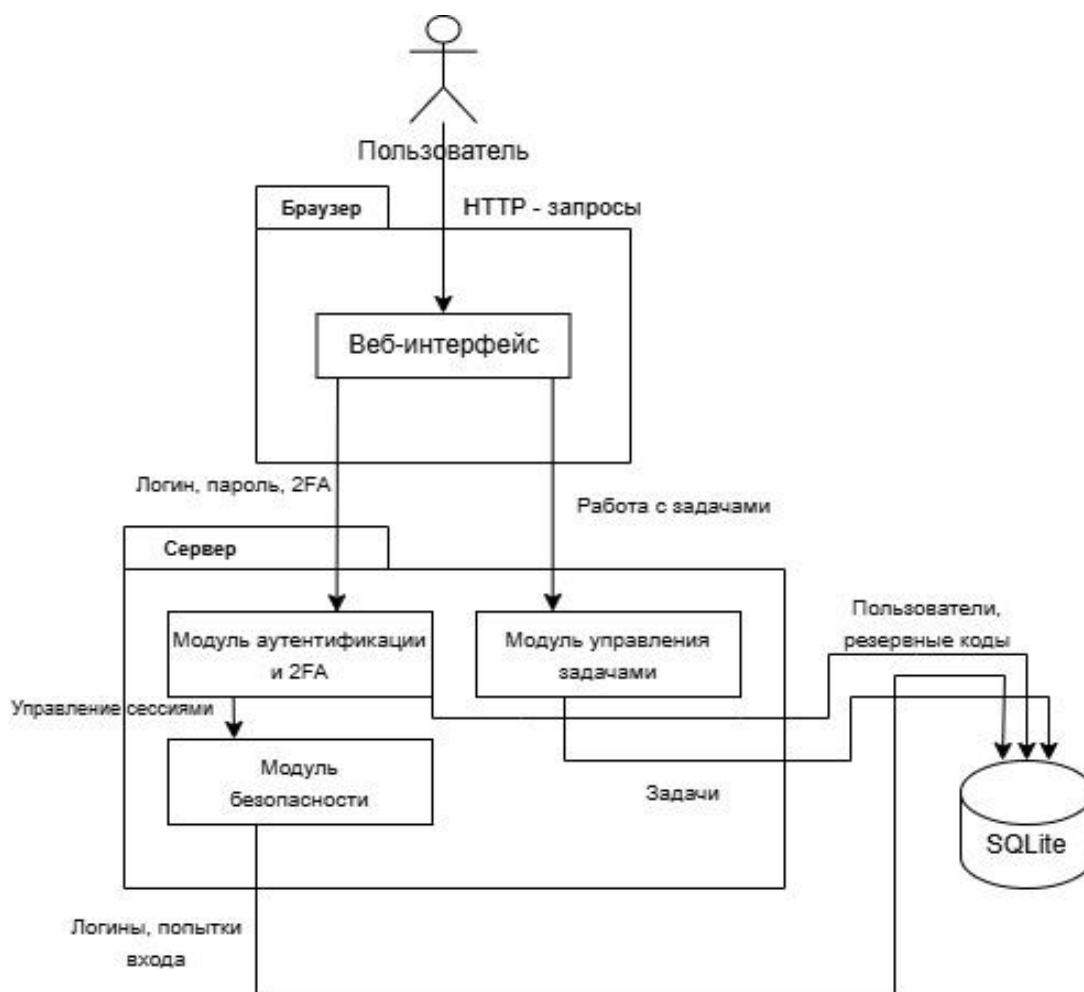


Рисунок 2.1 –Схема взаимодействия компонентов

Функциональность системы разделена на следующие модули:

– модуль аутентификации и регистрации – включает представления `user_register` и `user_login`. Регистрация требует проверки сложности пароля (минимум 8 символов, заглавные и строчные буквы, цифры, спецсимволы) и уникальности имени пользователя. Вход проверяет логин и пароль, а также учитывает ограничение числа неудачных попыток. При успешной аутентификации (но до завершения 2FA) идентификатор пользователя сохраняется в сессии как `pre_2fa_user_id`;

– модуль двухфакторной аутентификации – реализован в представлениях `two_factor_setup` и `two_factor_verify`. При первом входе генерируется случайный TOTP-секрет, создаётся QR-код для настройки приложения-аутентификатора, а также генерируются 8 резервных кодов. Верификация кода выполняется через `pyotp.TOTP.verify()`; также поддерживается вход по неиспользованному резервному коду;

– модуль управления задачами – представления `task_list`, `task_add`, `task_edit`, `task_delete`. Обычный пользователь видит только свои задачи, администратор (`is_staff`) – все задачи. Поддерживаются фильтрация по статусу (для всех) и по пользователю (только для администратора). При создании задачи администратор может назначить её нескольким пользователям;

– модуль разграничения прав доступа – основан на флаге `user.is_staff`. Здесь проверяется роль: для администратора доступны просмотр всех задач, создание задач для других пользователей, а также представление для сброса пароля и 2FA любому пользователю. Обычный пользователь не может получить доступ к чужим задачам, так как выборка ограничивает его данные;

– модуль защиты от перебора паролей – реализован через модель `LoginAttempt`, которая хранит для каждого логина количество неудачных попыток и время блокировки. После 5 неудачных попыток учётная запись блокируется на 10 минут. При успешном входе счётчик сбрасывается;

– модуль управления сессиями – использует стандартные механизмы Django. При смене пароля, сбросе 2FA или принудительном сбросе

администратором происходит удаление всех сессий пользователя из таблицы. Это гарантирует, что старые сессии становятся недействительными. Выход из системы выполняется представлением с вызовом `logout(request)`;

– модуль восстановления доступа – позволяет пользователю, забывшему пароль, сбросить его с использованием неиспользованного резервного кода. При этом проверяется сложность нового пароля, и все старые сессии пользователя удаляются.

В разработанном приложении принят принцип, согласно которому все операции аутентификации, проверки прав и доступа к данным выполняются на сервере. Клиент не имеет прямого доступа к базе данных, а все входящие данные проходят валидацию. Пароли пользователей хешируются средствами Django (используется PBKDF2 с солью). TOTP-секреты хранятся в модели `UserProfile` в открытом виде, однако доступ к ним имеют только серверные представления. Резервные коды хранятся в модели `BackupCode` в открытом виде, но их использование контролируется флагом `used` – после одноразового применения код становится недействительным. Логирование неудачных попыток входа позволяет отслеживать атаки перебора и автоматически блокировать злоумышленников. Защита от CSRF и XSS обеспечивается встроенными механизмами Django.

Типичный сценарий работы с приложением выглядит следующим образом. Пользователь регистрируется, указывая логин и пароль, который проверяется на сложность. После успешной регистрации его перенаправляют на страницу настройки двухфакторной аутентификации, где генерируются TOTP-секрет, QR-код и резервные коды. Пользователь сканирует QR-код приложением-аутентификатором и вводит полученный шестизначный код для подтверждения. При последующих входах пользователь вводит логин и пароль; если зафиксировано менее 5 неудачных попыток, сервер проверяет пароль, затем запрашивает TOTP-код (или резервный код). После успешной верификации создаётся сессия, и пользователь попадает на страницу со списком своих задач. Он может создавать, редактировать, удалять задачи и фильтровать их по статусу.

Администратор, помимо этого, видит задачи всех пользователей, может фильтровать по пользователям, создавать задачи для других и сбрасывать чужие пароль или 2FA через специальную форму. При смене пароля или сбросе 2FA все существующие сессии пользователя немедленно аннулируются. Выход из системы уничтожает текущую сессию.

Предложенная и реализованная архитектура веб-приложения для управления задачами обеспечивает обработку всех критических операций, изоляцию данных между пользователями и внедрение механизмов повышенной безопасности: Выбранные модули напрямую соответствуют требованиям, сформулированным в разделе 1.4, и подтверждаются листингами кода.

2.2 Выбор инструментов разработки

В качестве основного языка программирования выбран Python версии 3.12. Этот выбор обусловлен несколькими факторами. Во-первых, Python обладает мощными и удобными средствами для криптографических операций: библиотеки `pyotp`, `cryptography`, `hashlib` и встроенные механизмы Django для хеширования паролей. Во-вторых, синтаксис языка позволяет быстро разрабатывать прототипы и реализовывать сложную логику при малом объёме кода. В-третьих, наличие обширной системы библиотек для веб-разработки и безопасности сокращает время реализации и снижает вероятность ошибок.

Для построения серверной части выбран Django – один из наиболее зрелых и безопасных фреймворков для Python. Ниже приведены основные аргументы в пользу Django:

- встроенные механизмы аутентификации и управления сессиями. Django предоставляет готовые модели пользователей, систему сессий на основе `cookie`, а также гибкие инструменты для их расширения. В проекте стандартная модель `User` была дополнена профилем `UserProfile` для хранения TOTP-секрета и флага включения 2FA, а также моделью `LoginAttempt` для отслеживания попыток входа. Сессии сбрасываются при смене пароля или 2FA;

– защита от распространённых веб-уязвимостей «из коробки». Django автоматически экранирует вывод в шаблонах (защита от XSS), требует CSRF-токены для POST-запросов, предотвращает SQL-инъекции через ORM. Это соответствует требованиям безопасности, предъявляемым к веб-приложению;

– объектно-реляционное отображение (ORM). Позволяет безопасно и удобно работать с базой данных, не формируя SQL-запросы вручную. Все модели описаны на языке Python, а миграции автоматически поддерживают схему БД;

– встроенная админ-панель. Может быть использована для управления пользователями и просмотра логов, что полезно для администратора;

– алгоритм хеширования паролей по умолчанию – PBKDF2 с солью. Это соответствует современным рекомендациям OWASP и требованию безопасности.

Используется SQLite – встраиваемая база данных, не требующая отдельного серверного процесса. Выбор обусловлен следующими причинами:

– простота настройки и развёртывания. Для учебного проекта это критично: нет необходимости устанавливать и настраивать отдельный сервер БД;

– достаточная производительность. При предполагаемой нагрузке до 10 одновременных пользователей, SQLite справляется без задержек;

– поддержка транзакций и стандартного SQL. Все необходимые функции ограничения целостности, индексы, внешние ключи присутствуют;

– лёгкость миграции. Благодаря ORM Django, в будущем можно заменить SQLite на PostgreSQL без изменений в коде.

Ограничение: SQLite не поддерживает параллельную запись большим числом потоков, но в условиях дипломного проекта это не является проблемой. Также принято допущение, что приложение разворачивается в защищённой среде, поэтому дополнительные меры безопасности на уровне СУБД не требуются.

Для реализации конкретных механизмов безопасности были выбраны следующие библиотеки:

– pyotp – реализует алгоритм TOTP. Используется для генерации TOTP-секретов (pyotp.random_base32()), создания URI для QR-кода и верификации введённых пользователем кодов. Библиотека проста, легковесна и не требует внешних зависимостей;

– qrcode – генерирует QR-код в формате PNG, который передаётся на клиентскую сторону в виде base64-строки. Пользователь сканирует этот код приложением – аутентификатором (Яндекс.Ключ и др.). Использование qrcode стандартно для настройки 2FA;

– secrets и re – стандартные модули Python. secrets.token_hex() генерирует криптостойкие резервные коды; re используется для проверки сложности пароля;

– защита от перебора паролей – реализована вручную с помощью модели LoginAttempt, хранящей счётчик неудачных попыток и время блокировки для каждого логина.

Интерфейс пользователя построен на основе шаблонов Django с использованием HTML, CSS. Из Django были использованы стандартные формы. Например, форма смены пароля уже содержит проверки. Все данные, приходящие от клиента, проходят повторную проверку на сервере. Дополнительно был использован Bootstrap для адаптивной вёрстки. Это ускоряет разработку и обеспечивает интуитивно понятный интерфейс, соответствующий нефункциональным требованиям.

2.3 Установка и настройка инструментов разработки

Для реализации веб-приложения была подготовлена среда разработки, включающая необходимые интерпретаторы, библиотеки и инструменты. В данном подразделе подробно описаны выполненные шаги по установке и настройке программного обеспечения.

Установка интерпретатора Python. Установку Python рекомендуется начинать с перехода на официальный сайт. На главной странице сайта необходимо найти раздел загрузок (Downloads) и выбрать версию Python 3.12, доступную для операционной системы Windows. Для большинства современных

компьютеров с 64-разрядной архитектурой следует выбрать установочный файл «Windows installer (64-bit)»[3].

После завершения загрузки установочный файл необходимо запустить двойным щелчком мыши. В открывшемся окне установщика обязательно отметить флажок «Add Python to PATH» в нижней части окна – это добавляет интерпретатор в системные переменные окружения, позволяя запускать Python из любой директории командной строки. Далее рекомендуется выбрать вариант «Install Now», который выполняет установку со стандартными параметрами. Это помогает избежать проблем с некоторыми сторонними библиотеками в будущем. Процесс установки занимает не более одной-двух минут. После его завершения необходимо проверить корректность установки: открыть командную строку и выполнить команду `python --version`. Если установка выполнена правильно, на экране отобразится версия установленного интерпретатора. Соответствующие скриншоты процесса приведены на рисунках 2.2-2.4.

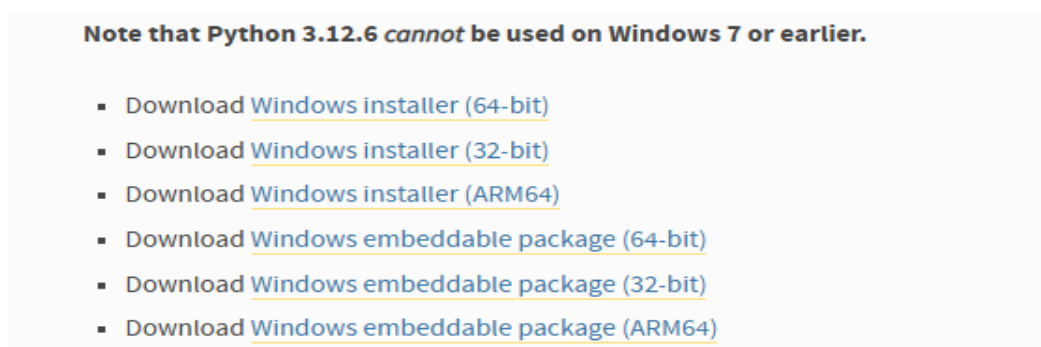


Рисунок 2.2 – Страница загрузки Python на официальном сайте

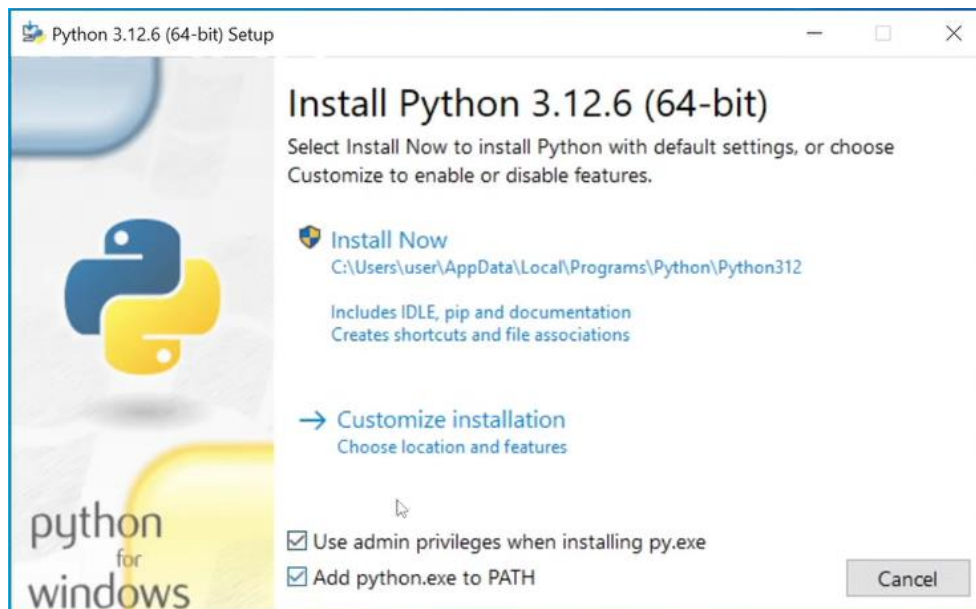


Рисунок 2.3 – Установщик Python с отмеченным флажком «Add Python to PATH»

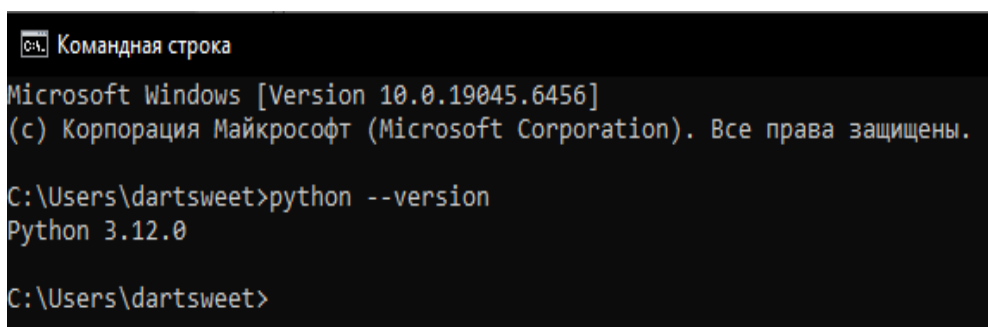


Рисунок 2.4 – Проверка версии Python в командной строке

Установка среды разработки PyCharm Community Edition. Для написания, отладки и запуска кода на Python используется интегрированная среда разработки PyCharm. Загрузка осуществляется с официального сайта компании JetBrains. На странице загрузки представлены две редакции: платная Professional и бесплатная Community. Для выполнения данной работы достаточно версии Community. Необходимо выбрать установочный файл для операционной системы Windows и запустить его [4].

В процессе установки открывается окно мастера установки. Рекомендуется выполнить следующие действия:

- на первом экране нажать кнопку «Next»;

– в окне выбора параметров установки отметить флажки: «Create Desktop Shortcut»– для добавления иконки приложения на рабочий стол, «Update PATH variable (restart needed)»– для добавления исполняемых файлов PyCharm в системную переменную PATH, «Add «Open Folder as Project»– для добавления пункта в контекстное меню проводника, «.py»– для ассоциации файлов с расширением .py с PyCharm.

Нажать «Install» и дождаться завершения установки. Соответствующие скриншоты процесса приведены на рисунках 2.5-2.7.

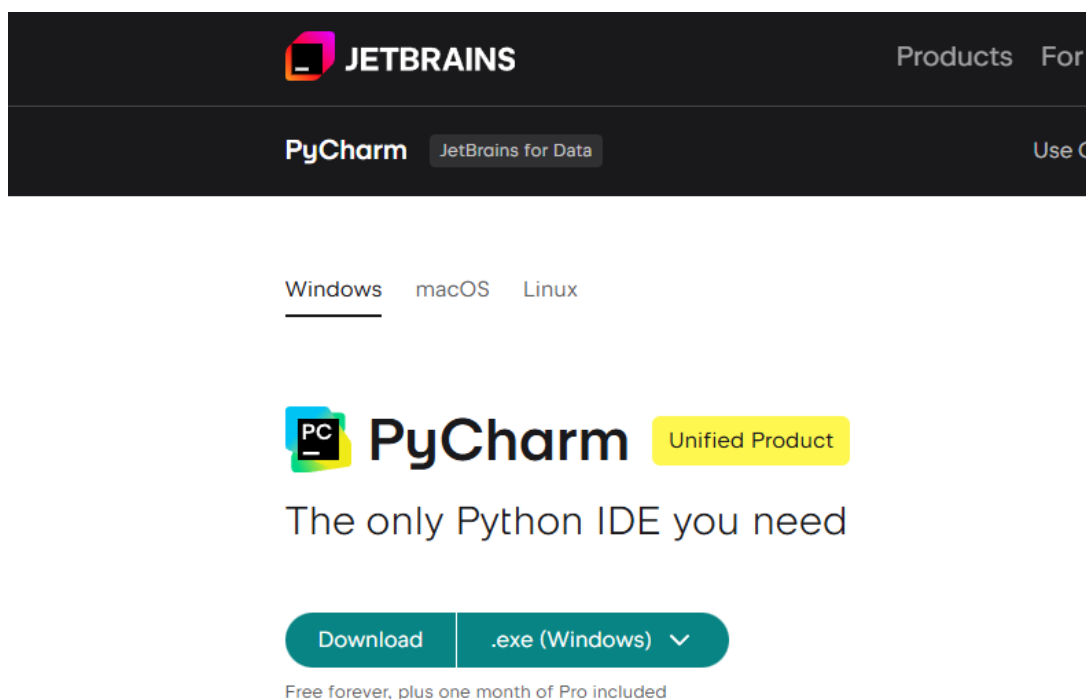


Рисунок 2.5 –Страницазагрузки PyCharm Community Edition

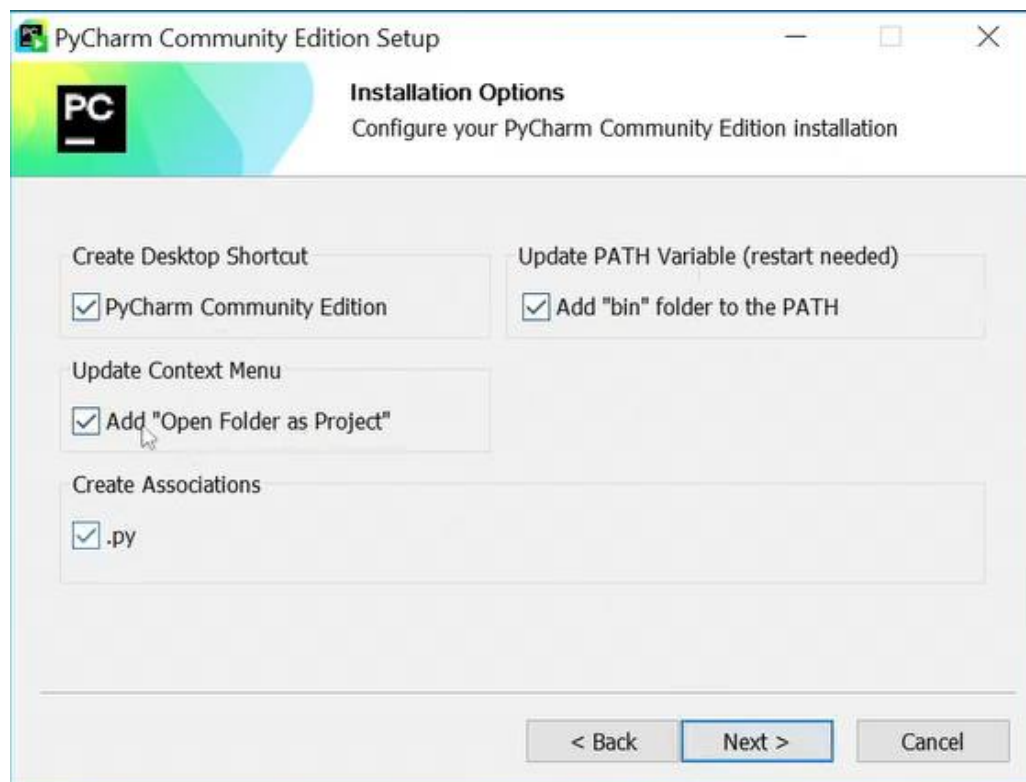


Рисунок 2.6 – Установка PyCharm: выбор параметров конфигурации

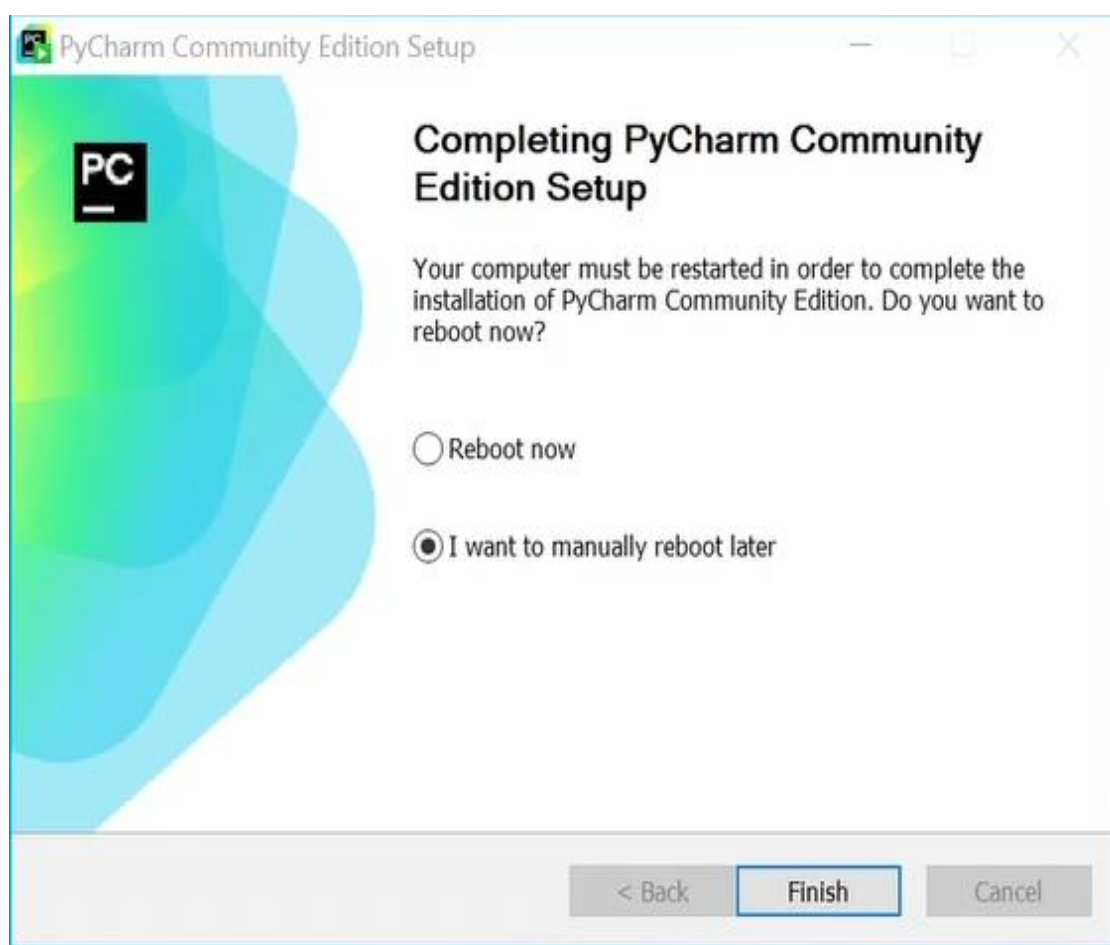


Рисунок 2.7 – Завершающий этап установки PyCharm

Первый запуск PyCharm и настройка интерпретатора. После установки PyCharm следует запустить программу через ярлык на рабочем столе или в меню «Пуск». Далее открывается стартовое окно, в котором необходимо создать новый проект. При создании проекта PyCharm автоматически предлагает создать виртуальное окружение на основе установленного интерпретатора Python. На этом этапе важно убедиться, что в качестве базового интерпретатора выбран именно Python, установленный на первом шаге. После подтверждения настроек проект будет создан, и откроется основной редактор кода. На рисунке 2.8 показан процесс создания проекта.

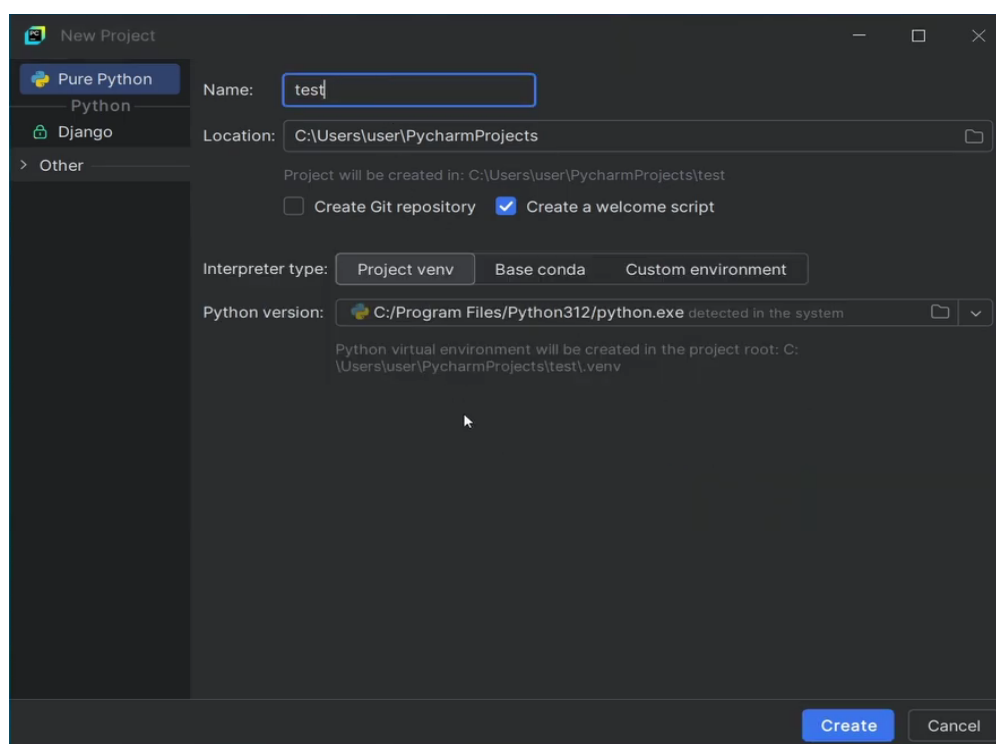


Рисунок 2.8 – Создание нового проекта с выбором интерпретатора

Проверка работоспособности среды. Для окончательной проверки корректности установки и настройки всех компонентов рекомендуется создать внутри проекта простой Python-файл с кодом и запустить его. Если в нижней части окна PyCharm в панели вывода отобразится строка «Hi, PyCharm», это

означает, что среда разработки полностью готова к дальнейшей работе. Рисунок 2.9.

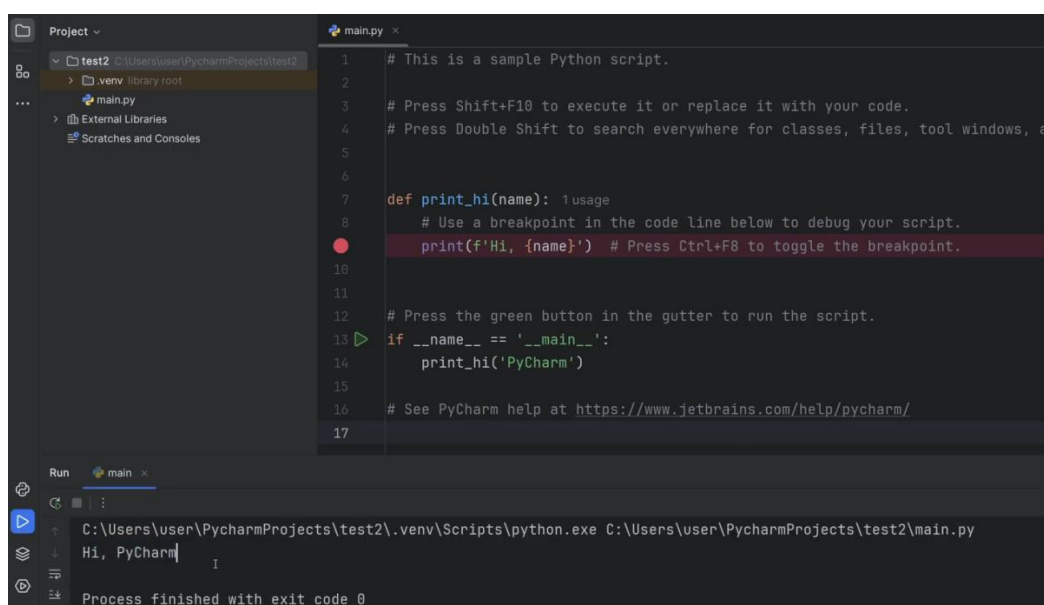


Рисунок 2.9 – Запуск тестового скрипта в PyCharm

Установка Django. В нижней части окна PyCharm открыта встроенная панель терминала. Убедившись, что в начале строки присутствует префикс (venv), была выполнена команда `pip install django`. Процесс завершился сообщением «Successfully installed...» (рисунок 2.10). Django стал доступен внутри виртуального окружения.

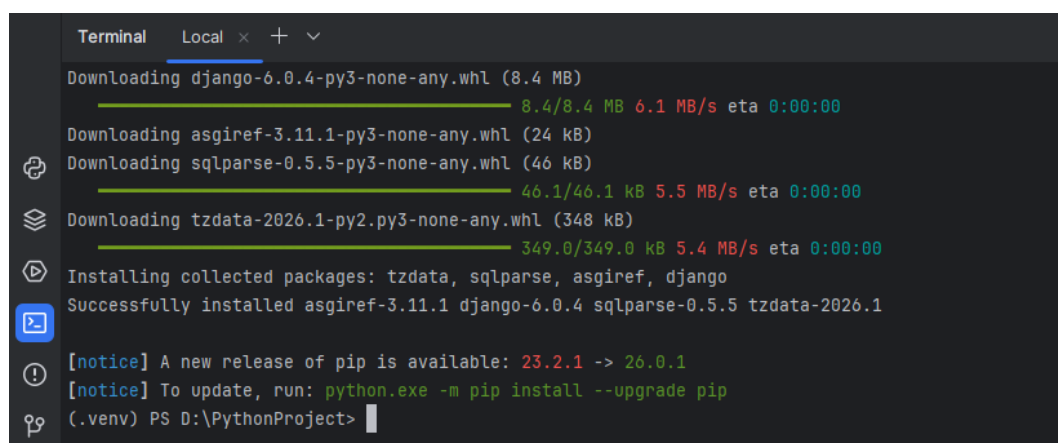


Рисунок 2.10 – Установка Django в терминале PyCharm

Установка библиотек безопасности. Для реализации двухфакторной аутентификации и вспомогательных функций в том же терминале выполнена

команда `pip install pyotp qrcode`. Библиотека `pyotp` реализует TOTP, `qrcode` генерирует QR-код для настройки 2FA. Успешное завершение установки показано на рисунке 2.11.

```
Using cached pyotp-2.9.0-py3-none-any.whl (13 kB)
Using cached qrcode-8.2-py3-none-any.whl (45 kB)
Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Installing collected packages: pyotp, colorama, qrcode
Successfully installed colorama-0.4.6 pyotp-2.9.0 qrcode-8.2

[notice] A new release of pip is available: 23.2.1 -> 26.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(.venv) PS D:\PythonProject>
```

Рисунок 2.11 – Установка библиотек безопасности

Запуск сервера разработки. Для проверки работоспособности окружения выполнена команда `python manage.py runserver`. В терминале появилось сообщение «Starting development server at <http://127.0.0.1:8000/>» (рисунок 2.12). Это подтверждает, что среда разработки настроена корректно и готова к написанию кода.

```
Terminal Local x + v
(.venv) PS C:\Users\dartsweet\secure_task_tracker> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 20, 2026 - 16:47:43
Django version 5.2.8, using settings 'secure_tracker.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

WARNING: This is a development server. Do not use it in a production setting. Use a production WSGI or ASGI server instead.
For more information on production servers see: https://docs.djangoproject.com/en/5.2/howto/deployment/
```

Рисунок 2.12 – Запуск сервера разработки и проверка в браузере

В результате выполненных действий среда разработки полностью готова к реализации функциональности приложения. Все последующие модули разрабатывались и отлаживались именно в этой среде.

2.4 Описание структуры программы

Разрабатываемое приложение построено на основе архитектурного шаблона Model-Template-View (MTV), реализованного в веб-фреймворке Django. Данный шаблон является вариацией классического MVC (Model-View-Controller), где модель отвечает за данные и логику, шаблон – за представление данных пользователю, а представление – за обработку запросов и выбор шаблона. Такой подход обеспечивает чёткое разделение ответственности, упрощает тестирование кода, а также способствует повторному использованию компонентов.

В проекте создано одно приложение `tasks`, внутри которого сгруппированы все модули. Логика сосредоточена в `views`, который обращается к моделям для чтения и записи данных. Маршрутизация направляет HTTP-запросы к соответствующим функциям-представлениям. Представления в свою очередь используют шаблоны для генерации HTML-ответов. Механизмы безопасности реализованы внутри представлений и вспомогательных функций.

На рисунке 2.13 представлена диаграмма сущностей. На рисунке 2.14 приведена диаграмма модели `urls`. Диаграмма классов отображена в Приложении А.1.

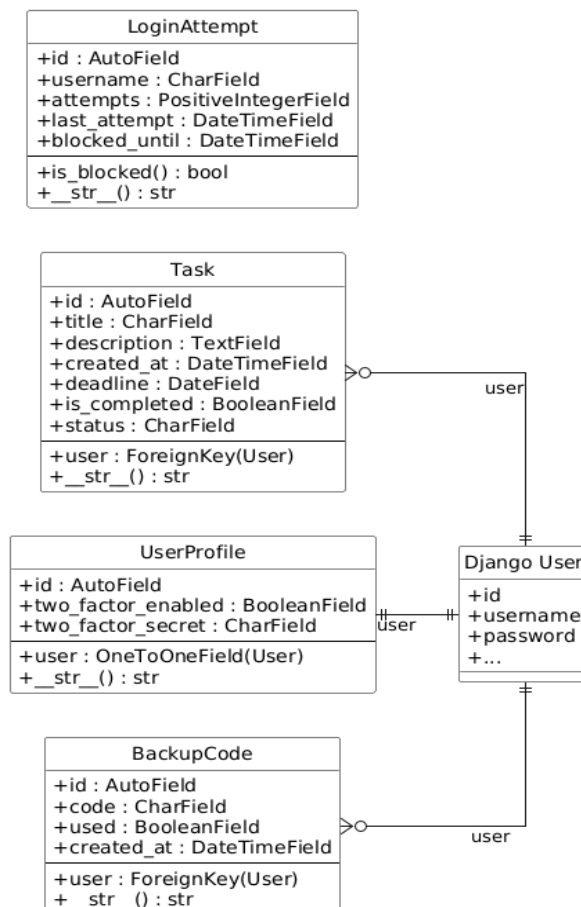


Рисунок 2.13 – Диаграмма сущностей

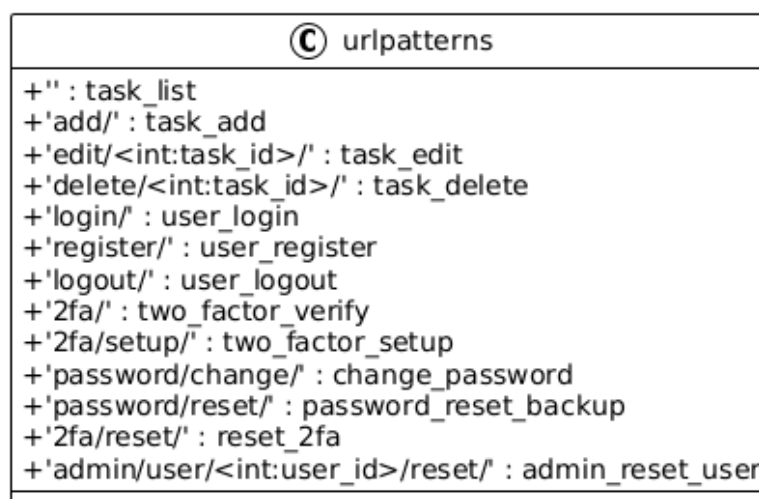


Рисунок 2.14 – Диаграмма urls

На рисунке 2.15 представлена блок-схема, иллюстрирующая логику работы модуля аутентификации, включая проверку пароля, двухфакторную верификацию и защиту от перебора.

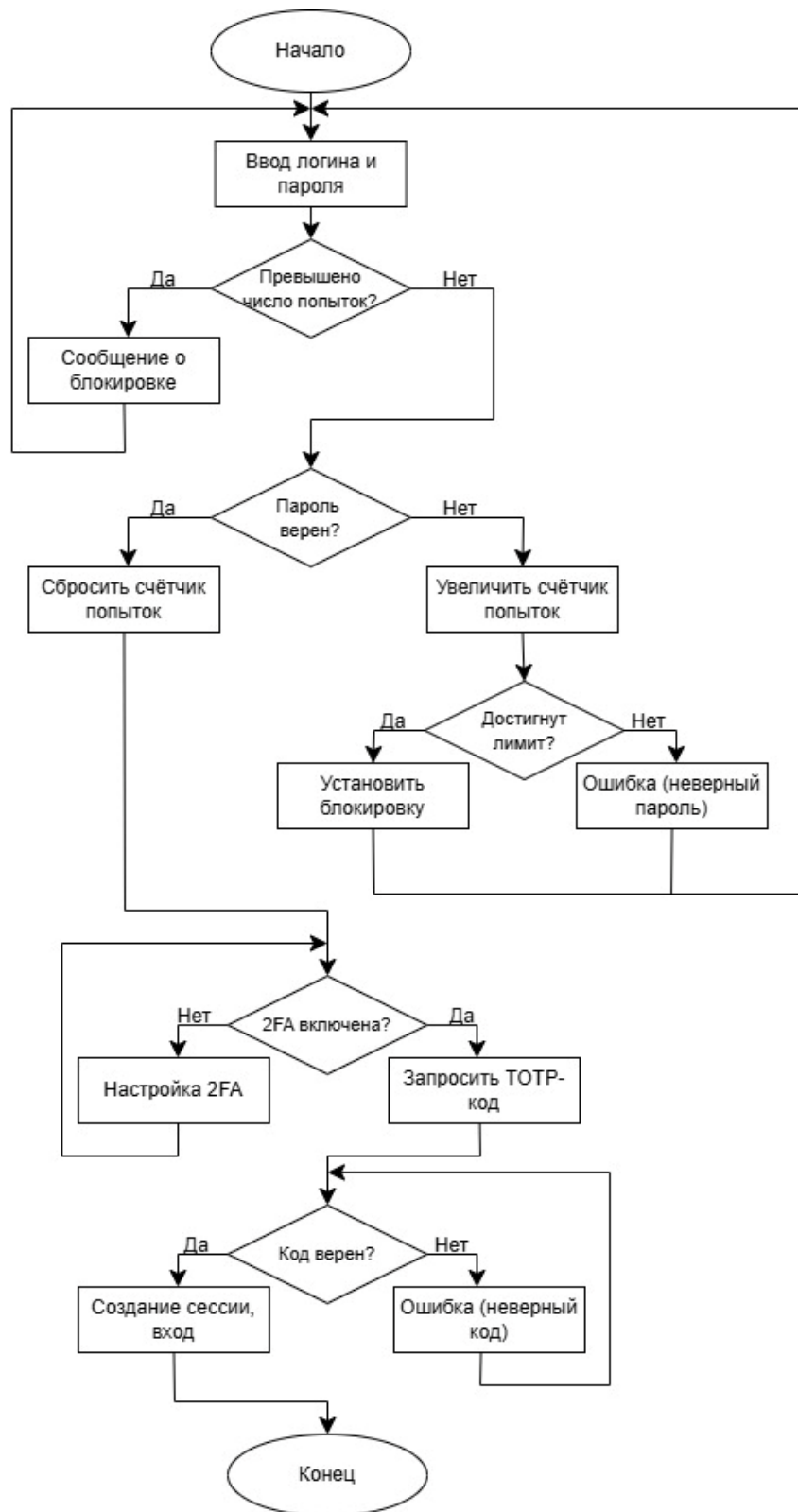


Рисунок 2.15 – Блок–схема процесса аутентификации

На рисунке 2.16 приведена диаграмма последовательности. Она отображает взаимодействие между пользователем, клиентом, сервером и базой данных.

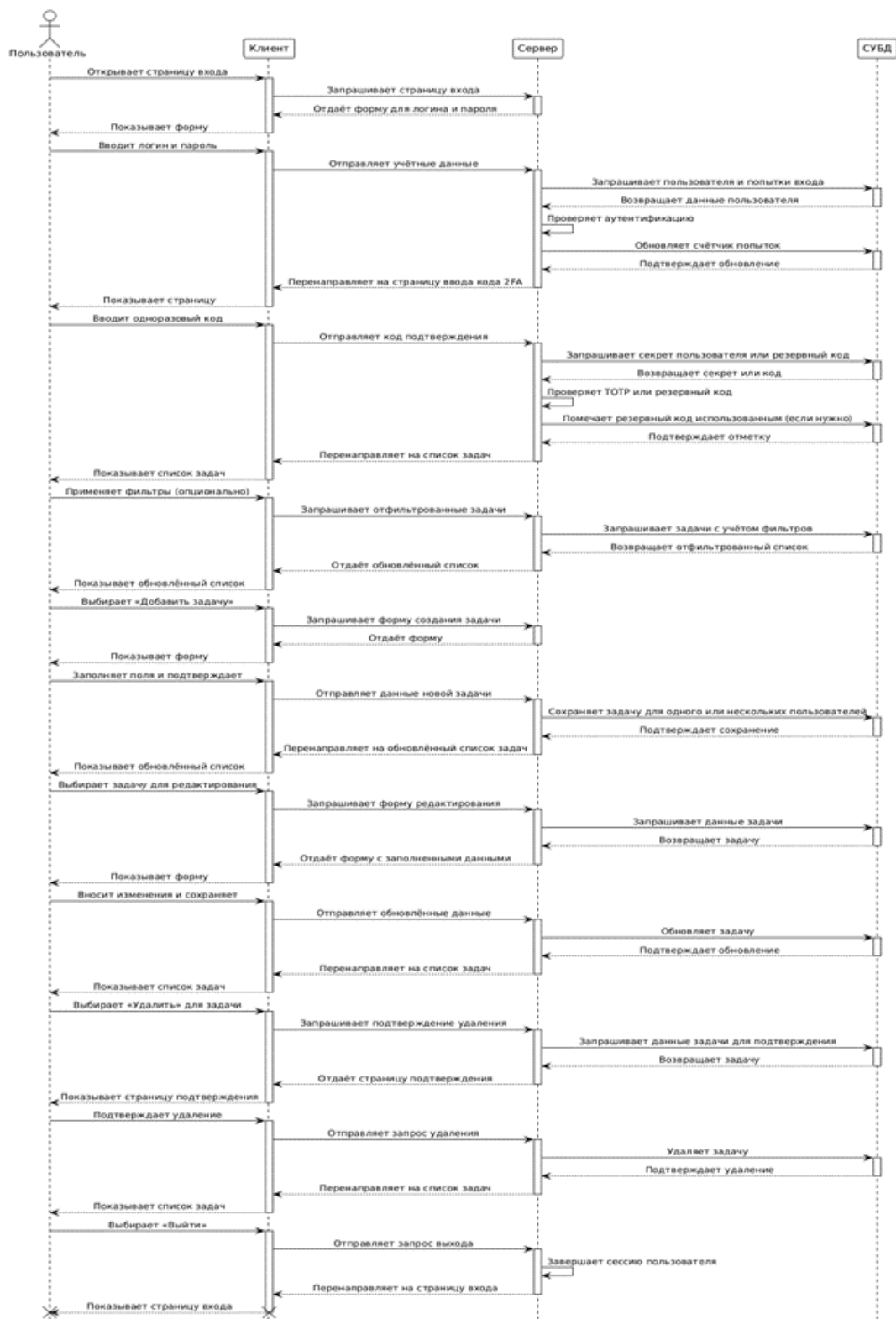


Рисунок 2.16 – Диаграмма последовательности

2.5 Разработка программы

Изм.	Лист	№докум.	Подпись	Дата

В данном разделе описывается реализация ключевых компонентов веб-приложения: модуля аутентификации с двухфакторной защитой, модуля управления задачами и модуля обеспечения безопасности. Разработка велась на языке Python с использованием фреймворка Django. Полные листинги кода приведены в приложении Б.

2.5.1 Реализация аутентификации и двухфакторной защиты

Модуль аутентификации включает регистрацию, вход, а также настройку и проверку двухфакторной аутентификации на основе временных одноразовых паролей.

При регистрации полученные от пользователя логин и пароль проходят проверку: уникальность логина, а также сложность пароля с помощью. Пароль должен содержать не менее 8 символов, заглавные и строчные буквы, цифры и специальные символы. После создания пользователя его идентификатор сохраняется в сессии, и выполняется перенаправление на страницу настройки 2FA.

Далее проверяется наличие профиля пользователя, при необходимости генерирует TOTP-секрет. Затем создаётся URI для настройки приложения-аутентификатора, на основе которого формируется QR-код. QR-код передаётся в шаблон в виде base64-строки. Одновременно генерируются 8 одноразовых резервных кодов. Пользователь сканирует QR-код и затем подтверждает настройку.

После успешной проверки логина и пароля идентификатор пользователя также помещается в сессию, и если у пользователя уже включена 2FA, выполняется редирект на страницу ввода кода. Полученный код проверяется сначала по TOTP, а затем среди неиспользованных резервных кодов. При успехе выполняется штатный вход.

2.5.2 Реализация управления задачами

Управление задачами реализовано в представлениях `task_list`, `task_add`, `task_edit`, `task_delete`. Модель `Task` содержит поля: название, описание, дату создания, дедлайн, статус (на выбор из четырёх состояний) и внешний ключ на пользователя.

В `task_list` в зависимости от флага формируется выборка задач: администратор видит все задачи, обычный пользователь – только свои. Фильтрация по статусу и по пользователю (для администратора) реализована через параметры GET-запроса. Аналогичные проверки прав встроены в `task_add`: администратор может выбрать нескольких пользователей для создания задачи, обычный пользователь создаёт задачу только для себя.

Создание задачи выполняется в цикле по выбранным пользователям. Проверка полей выполняется на стороне клиента и сервера. После успешного выполнения операций выполняется редирект на список задач.

2.5.3 Реализация механизмов безопасности

Для каждого логина ведётся учёт неудачных попыток входа с помощью модели `LoginAttempt`. Перед аутентификацией проверяется, не заблокирована ли учётная запись. При превышении последовательных неудачных попыток входа, учётная запись временно блокируется. При успешном входе счётчик попыток сбрасывается, блокировка снимается.

При смене пароля или при сбросе 2FA, все сессии данного пользователя принудительно удаляются. Это исключает возможность использования старых сессий после компрометации учётных данных.

При утере доступа к TOTP-генератору пользователь может использовать один из ранее сохранённых резервных кодов. Кроме того, администратор может принудительно сбросить пароль и отключить 2FA любому пользователю, что также сопровождается удалением всех его резервных кодов и сессий.

2.5.4 Реализация вспомогательных модулей

В системе применяется встроенный способ разделения обычных пользователей и администраторов. У каждой учётной записи есть специальный флаг, который показывает, является ли человек администратором. Именно по

этому флагу определяется, может ли пользователь выполнять служебные действия.

Функция принудительного сброса чужого пароля доступна только администраторам. Перед её выполнением система проверяет, установлен ли у текущего пользователя административный флаг. Если нет – доступ закрыт. Такая же проверка делается и в интерфейсе: шаблоны страниц показывают или скрывают административные элементы в зависимости от того, какие права у пользователя.

Надёжность паролей обеспечивается проверкой по строгим правилам. Пароль оценивается на соответствие шаблону, в котором заданы минимальная длина и обязательное присутствие символов разных типов. Это гарантирует, что пользователи не смогут задать слишком простые или короткие комбинации.

Для двухфакторной аутентификации система создаёт запасные одноразовые коды. Они генерируются при помощи криптографически надёжного источника случайных. Такой способ исключает возможность подбора или угадывания кодов.

В адресации веб-приложения реализовано чёткое разделение: адреса, отвечающие за работу с задачами, за вход и регистрацию, а также за двухфакторную аутентификацию, выделены в отдельные логические группы. Каждый адрес связан со своим обработчиком, что делает структуру проекта понятной и удобной для дальнейшей поддержки.

2.6 Описание доступных функций

В данном разделе демонстрируется работа разработанного веб-приложения с точки зрения конечного пользователя. Все скриншоты, приведённые ниже, иллюстрируют выполнение функциональных требований и механизмов безопасности, сформулированных в главе 1.

2.6.1 Домашняя страница

При переходе на сайт отображается домашняя страница, на которой есть: название сайта, сверху есть «шапка» (navbar) и так же 3 кнопки: «Создать заметку», «Войти», «Зарегистрироваться». Рисунок 2.17.

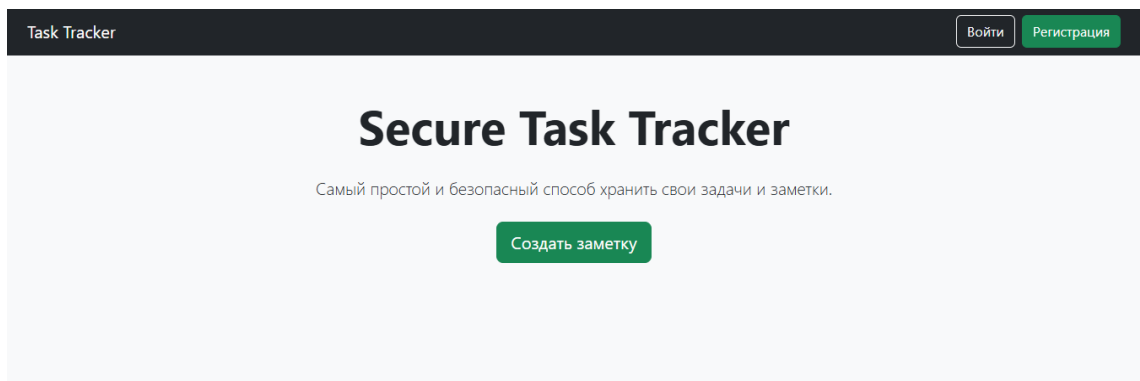


Рисунок 2.17 – Домашняя страница

2.6.2 Регистрация и аутентификация

Форма регистрации содержит поля «Логин» и «Пароль». При вводе пароля проверяется его сложность. Если пароль не соответствует требованиям, пользователь видит список ошибок. Рисунок 2.18.

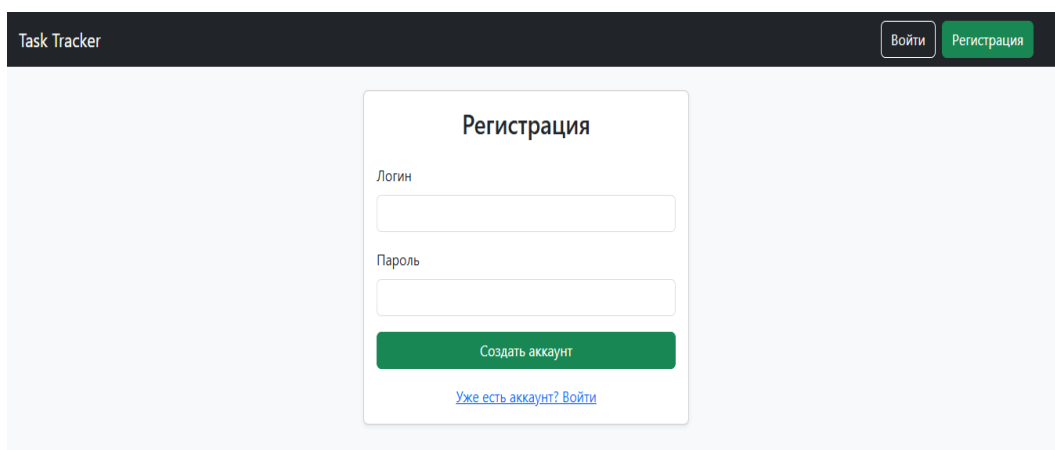


Рисунок 2.18 – Форма регистрации нового пользователя

В форме входа пользователь вводит логин и пароль. При успешной проверке (и если двухфакторная аутентификация уже включена) он перенаправляется на страницу ввода TOTP-кода. Если 2FA ещё не настроена, после входа автоматически запускается процесс настройки. Рисунок 2.19.

Рисунок 2.19 – Форма входа в систему

2.6.3 Двухфакторная аутентификация (2FA)

После регистрации отображается страница с QR-кодом, содержащим TOTP-секрет. Пользователь сканирует QR-код любым приложением-аутентификатором (Яндекс.Ключ и др.). Ниже QR-кода показаны 8 одноразовых резервных кодов, которые рекомендуется сохранить. Рисунок 2.20.

Одноразовые резервные коды				
Сохраните эти коды! Каждый код можно использовать только один раз.				
	c4f03391			
	1e387eeb			
	7ecc1958			
	2f48bc5e			
	99860cfa			
	5481c06c			
	069e88a2			
	5c606476			

Рисунок 2.20 – Настройка двухфакторной аутентификации: QR-код и резервные коды

После сканирования QR-кода пользователь вводит текущий шестизначный код из приложения-аутентификатора и нажимает «Подтвердить». При правильном коде 2FA активируется, и пользователь попадает на главную страницу. Рисунок 2.21.

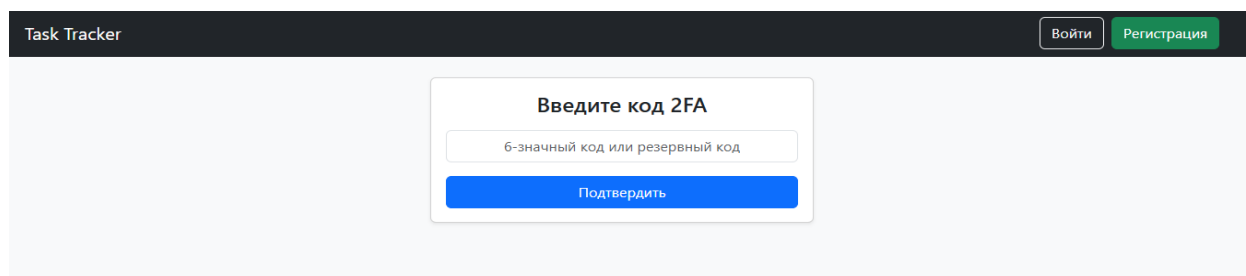
The screenshot shows a web interface for 'Task Tracker'. At the top right, there are buttons for 'Войти' (Login) and 'Регистрация' (Registration). The main content area is a light gray box containing a white card. The card has the title 'Введите код 2FA' (Enter 2FA code). Below the title is a text input field with the placeholder '6-значный код или резервный код' (6-digit code or backup code). At the bottom of the card is a blue button labeled 'Подтвердить' (Confirm).

Рисунок 2.21 – Подтверждение кода двухфакторной аутентификации

2.6.4 Навигация и общий вид интерфейса

На главной странице отображается таблица задач текущего пользователя. В верхней части находится панель навигации: ссылки на список задач, добавление задачи, смену пароля, настройку 2FA и выход. Для администратора также появляется ссылка «Сбросить доступ», только если есть активная задача от конкретного пользователя. Интерфейс выполнен с использованием Bootstrap 5, что обеспечивает адаптивность и интуитивную навигацию. Рисунок 2.22.

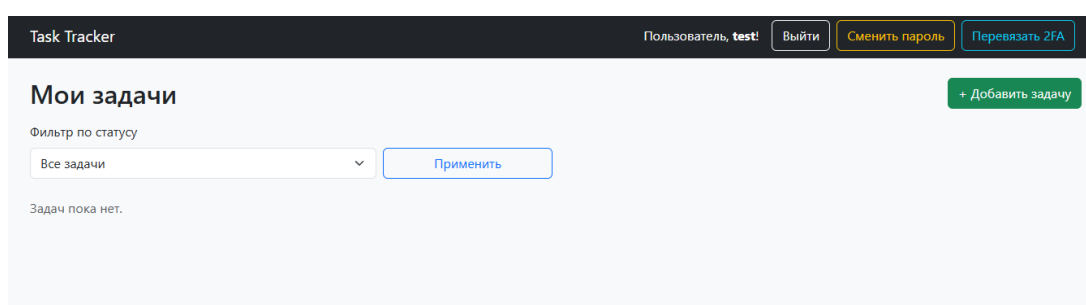
The screenshot shows the main page of the 'Task Tracker' application after a successful login. The top navigation bar is dark gray and contains the text 'Task Tracker' on the left and user information on the right: 'Пользователь, test!' followed by buttons for 'Выйти' (Logout), 'Сменить пароль' (Change password), and 'Перезагрузить 2FA' (Reload 2FA). Below the navigation bar, the main content area has a title 'Мои задачи' (My tasks) and a green button '+ Добавить задачу' (Add task). There is a 'Фильтр по статусу' (Filter by status) section with a dropdown menu currently showing 'Все задачи' (All tasks) and a 'Применить' (Apply) button. Below this, it says 'Задач пока нет.' (No tasks yet).

Рисунок 2.22 – Страница после входа в систему

2.6.5 Управление задачами (CRUD)

На странице списка задач для каждого пользователя выводятся его задачи с полями: название, описание, статус, дедлайн, а также кнопки редактирования и удаления. Выше списка расположены кнопка фильтрации по статусу. Рисунок 2.23.

Фильтр по статусу

Все задачи

▼

Применить

test_edit

test?

Дедлайн: May 8, 2026

✎ Редактировать

🗑 Удалить

В работе

Рисунок 2.23 – Активный список задач

При нажатии «Добавить задачу» открывается форма с полями: название, описание, дедлайн, статус. Для администратора дополнительно отображается список пользователей, для которых можно создать задачу. После сохранения задачи пользователь возвращается к списку. Рисунок 2.24.

Task Tracker

Пользователь, admin! Выйти Сменить пароль Перевязать 2FA

Создать задачу

Назначить пользователям

admin
proverka
i
test

Можно выбрать несколько пользователей (Ctrl / Shift)

Название задачи

Описание

Дедлайн

ДД.ММ.ГГГГ

Статус

К выполнению

▼

Создать

Отмена

Рисунок 2.24 – Форма создания новой задачи

При нажатии кнопки «Редактировать» открывается форма с предзаполненными данными задачи. Пользователь может изменить название, описание, дедлайн и статус. После сохранения обновлённая задача отображается в списке. Рисунок 2.25.

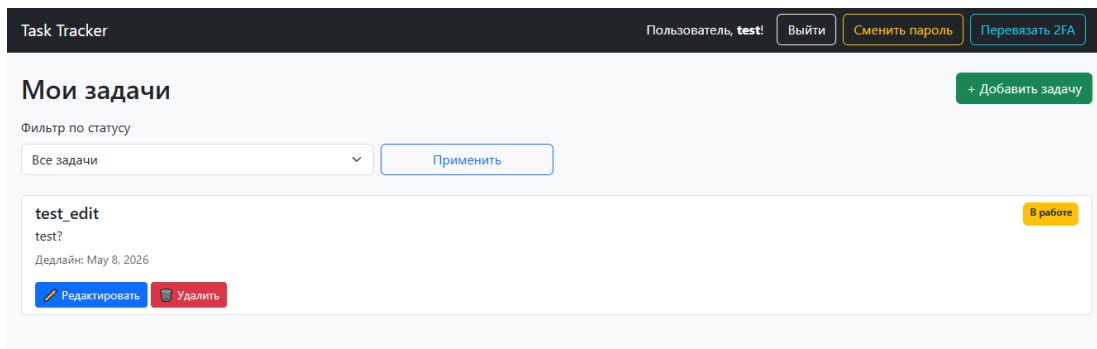


Рисунок 2.25 – Редактирование задачи

При нажатии кнопки «Удалить» отображается страница подтверждения. После подтверждения задача удаляется из базы данных, и отображается обновлённый список. Рисунок 2.26.

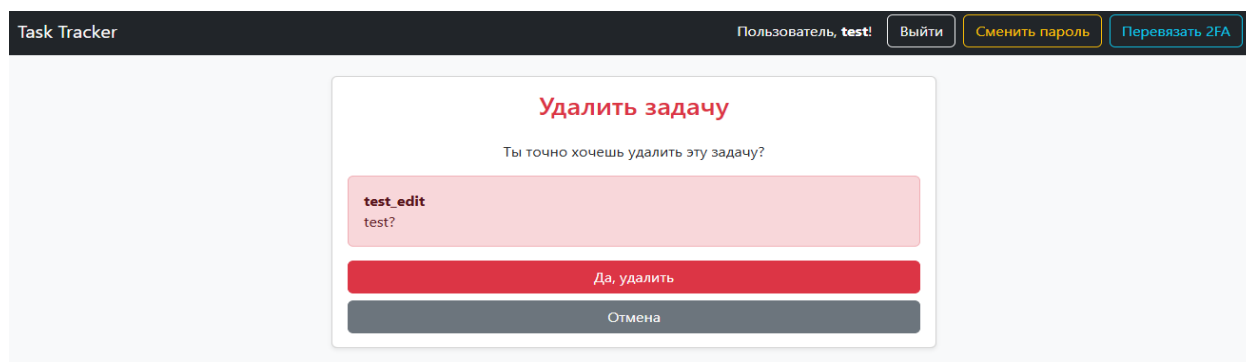


Рисунок 2.26 – Удаление задачи

2.6.6 Разграничение прав доступа

Обычный пользователь видит только те задачи, в которых он указан как исполнитель. Отсутствуют также элементы для создания задач для других и административные ссылки. Рисунок 2.27.

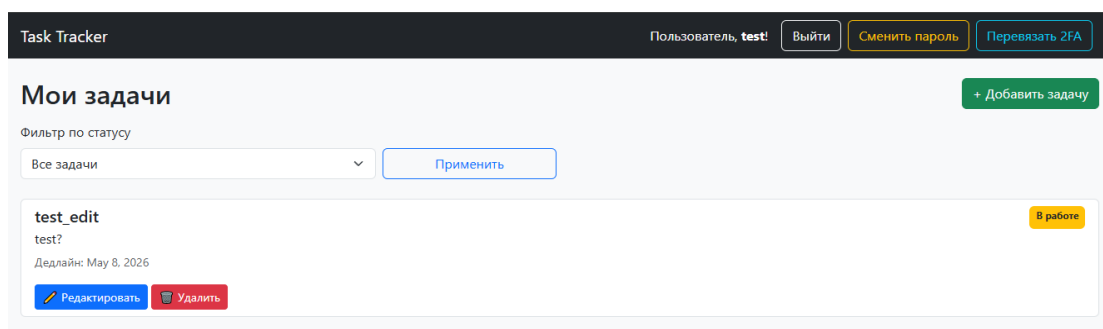


Рисунок 2.27 – Список задач для обычного пользователя

Администратор видит все задачи всех пользователей, а также логин создателя задачи. В панели задачи появляется ссылка «Сбросить доступ», а в форме создания задачи – возможность выбрать исполнителей. Администратор также может фильтровать задачи по пользователям. Рисунок 2.28.

Рисунок 2.28 – Список задач для администратора

2.6.7 Защита от перебора паролей

При вводе неверного логина или пароля счётчик неудачных попыток увеличивается. После 5 неудачных попыток подряд учётная запись блокируется на 10 минут. При попытке входа в заблокированном состоянии пользователь видит сообщение: «Слишком много попыток. Попробуйте через 10 мин.». Рисунок 2.29.

Рисунок 2.29 – Сообщение о временной блокировке после неудачных попыток
входа

После успешного входа счётчик попыток обнуляется, и блокировка снимается. В дальнейшем пользователь снова может выполнять вход без задержек.

2.6.8 Смена пароля

Аутентифицированный пользователь может изменить свой пароль, перейдя по ссылке «Сменить пароль» в панели навигации. Открывается стандартная форма Django PasswordChangeForm, содержащая поля: старый пароль, новый пароль и подтверждение нового пароля. При вводе нового пароля проверяется его сложность (аналогично регистрации). После успешной смены пароля все существующие сессии пользователя, включая текущую, завершаются, и он перенаправляется на страницу входа для повторной аутентификации. Рисунок 2.30.

Рисунок 2.30 – Форма смены пароля

2.6.9 Сброс двухфакторной аутентификации

Если пользователь утратил доступ к приложению-аутентификатору или хочет переустановить 2FA, он может нажать ссылку «Сбросить 2FA». При этом старые TOTP-секрет и резервные коды удаляются, генерируются новые, а пользователь выходит из системы и перенаправляется на страницу настройки 2FA, аналогично первому входу.

2.6.10 Выход из системы и завершение сессии

При нажатии на ссылку «Выход» в правом верхнем углу текущая сессия пользователя завершается, и он перенаправляется на страницу входа. Повторный доступ к защищённым страницам (список задач, настройки 2FA) становится невозможным без повторной аутентификации.

3 Анализ результатов

3.1 Результаты разработки приложения

В данном разделе анализируется степень достижения поставленных задач, а также проверяется соответствие разработанного веб-приложения функциональным требованиям и требованиям безопасности.

В данном приложении реализовано: регистрация с проверкой сложности пароля, вход по логину и паролю, двухфакторная аутентификация (TOTP), резервные коды, смена пароля, сброс 2FA. Управление задачами: создание, редактирование, удаление, просмотр, фильтрация по статусу. Ролевое разграничение: обычный пользователь видит только свои задачи; администратор – все задачи, может создавать задачи для других, сбрасывать пароль и 2FA любому пользователю. Защита от перебора паролей: 5 неудачных попыток, блокировка на 10 минут. Безопасное управление сессиями: завершение всех сессий при смене пароля или сбросе 2FA. Восстановление доступа: вход по резервному коду.

И не реализовано: шифрование отдельных полей и модуль аудита с веб-интерфейсом для администратора.

Все ключевые требования, кроме шифрования и отдельного модуля аудита, полностью выполнены. Отсутствующие функции не являются критическими для безопасности учётных записей и могут быть добавлены в следующих версиях.

Для проверки работоспособности реализованных механизмов проведено тестирование по четырём сценариям:

Сценарий 1. Защита от перебора паролей

Выполнено 6 последовательных попыток входа с неверным паролем. После 5-й попытки учётная запись заблокирована на 10 минут, шестая попытка отклонена с сообщением о блокировке. После успешного входа счётчик сброшен. Рисунок 3.1.

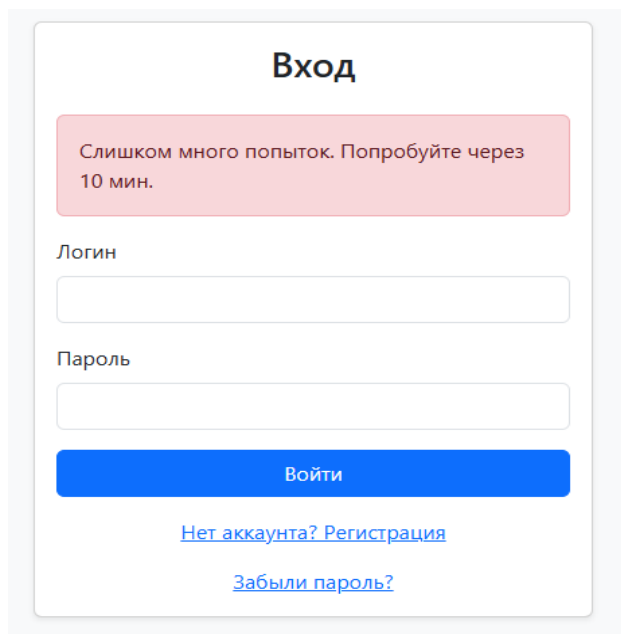


Рисунок 3.1 – Итог сценария 1

Сценарий 2. Двухфакторная аутентификация

Для пользователя с включённой 2FA после ввода верного пароля запрошен ТОТР-код. Ввод неверного кода отклонён, ввод верного – предоставлен доступ к списку задач. Работа резервных кодов также проверена: вход с неиспользованным кодом успешен, повторное использование того же кода отклонено. Рисунок 3.2.

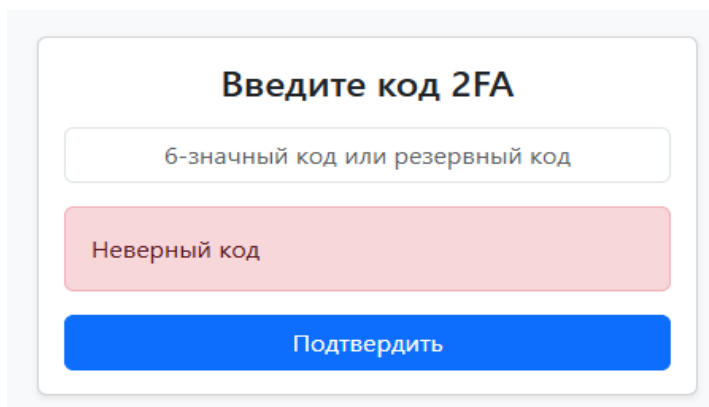


Рисунок 3.2 – Итог сценария 2

Сценарий 3. Управление сессиями

После смены пароля все старые сессии пользователя завершены – попытка использовать ранее сохранённую cookie привела к перенаправлению на страницу входа. Аналогичное поведение зафиксировано после сброса 2FA.

Сценарий 4. Проверка конфиденциальности данных

Шифрование поля «описание задачи» не входило в обязательные требования. При проверке базы данных было подтверждено, что значения поля хранятся в открытом виде. Тест показал ожидаемый результат: шифрование отсутствует. Данный функционал может быть добавлен в следующих версиях. Рисунок 3.3.

	id	title	description	created_at	user_id	deadline	is_completed	status
	Фи...	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр	Фильтр
1	36	test_edit	test?	2026-05-04 17:14:51.572108	48	2026-05-08	0	progress
2	37	proverka_polya	encrypted?	2026-05-17 15:10:14.152349	45	2026-05-17	0	hold

Рисунок 3.3 – Итог сценария 4

Разработанное веб-приложение для управления задачами успешно реализует все обязательные функциональные требования и основные механизмы повышенной безопасности: двухфакторную аутентификацию (TOTP), резервные коды, защиту от перебора паролей с временной блокировкой, безопасное управление сессиями и ролевое разграничение.

3.2 Рекомендации к дальнейшей разработке

Разработанное веб-приложение представляет собой полностью функционирующий прототип системы управления задачами с повышенным набором механизмов безопасности. Для превращения его в полноценный продукт, пригодный для использования или развёртывания в организации, существует ряд направлений доработки.

Для хеширования паролей используется стандартный механизм с случайной солью на каждого пользователя, которая хранится открыто вместе с

хешем и числом итераций – это общепринятая практика, затрудняющая подбор за счёт большого количества итераций, однако для повышения безопасности рекомендуется рассмотреть три шага: переход на более современный алгоритм, устойчивый к атакам на графических процессорах и специализированных чипах, с возможностью настройки памяти и параллельности; внедрение глобального секрета, который не хранится в базе данных и добавляется к паролю перед хешированием, что делает перебор бесполезным даже при утечке; а также генерация соли на основе глобального секрета и идентификатора пользователя без хранения её в базе – это полезно для распределённых систем, но снижает энтропию соли, поэтому рекомендуется только при жёсткой необходимости горизонтального масштабирования.

Данные хранятся в открытом виде в базе данных, поэтому для обеспечения конфиденциальности информации о задачах необходимо внедрить симметричное шифрование при сохранении и расшифрование при чтении.

Журналирование критических операций с фиксацией времени и типа действия позволит администратору отслеживать подозрительную активность и расследовать инциденты. В текущей реализации используются только стандартные логи Django, не предназначенные для удобного просмотра в интерфейсе.

Сброс пароля возможен только при наличии резервного кода или через администратора. Для повышения удобства рекомендуется добавить стандартный механизм восстановления по email с отправкой одноразовой ссылки.

SQLite отлично подходит для разработки и тестирования, но при росте числа одновременных пользователей его производительность падает, особенно при записи. PostgreSQL обеспечивает конкурентный доступ, блочную блокировку и расширенные возможности мониторинга.

Каждая задача имеет только название, описание, статус и дедлайн. Для полноценного управления проектами не хватает обсуждений и вложений.

Реализация предложенных направлений позволит превратить учебный прототип в полноценное решение, пригодное для ежедневного использования в командах.

					ПГУ 1.090301.07.001 ПЗ	Лист
						52
Изм.	Лист	№докум.	Подпись	Дата		

4 Безопасность жизнедеятельности

4.1 Правила безопасности при использовании ПК

При работе с персональным компьютером необходимо учитывать вредные и опасные факторы. К основным из них относятся:

- повышенный уровень напряжённости электрического и магнитного полей в зоне установки компьютера;
- повышенная температура поверхности отдельных элементов ПК (процессора, блока питания);
- монотонность трудового процесса при длительной работе с программным обеспечением;
- высокий уровень зрительных нагрузок вследствие длительного наблюдения за экраном монитора;
- вероятность поражения статическим электричеством, генерируемым компьютерной техникой;
- недостаточная освещённость рабочей зоны при выполнении операций ввода-вывода данных.

Полностью исключить воздействие перечисленных факторов невозможно, однако разработчик и пользователь приложения обязаны принимать меры к их минимизации. К таким мерам относятся: соблюдение режима труда и отдыха (перерывы каждые 45-60 минут), поддержание комфортного микроклимата и освещения, использование качественного заземления оборудования, выполнение гимнастики для глаз и разминки для опорно-двигательного аппарата.

4.2 Пожарная безопасность

При эксплуатации электроустановок и персональных компьютеров наиболее частыми причинами возникновения пожаров являются:

- короткие замыкания в проводках и электрических устройствах;

- воспламенение горючих материалов (бумага, пластик, ткань), находящихся вблизи от работающей техники;
- большие переходные сопротивления в местах контактных соединений (розетках, удлинителях, переходниках).

Профилактика пожаров регламентируется Федеральным законом от 21 декабря 1994 г. № 69-ФЗ (ред. от 28.09.2010) «О пожарной безопасности». Для обеспечения пожарной безопасности в помещении с компьютерной техникой необходимо выполнить следующие организационные мероприятия:

- проходы в помещении и запасные выходы не должны быть загромождены посторонними предметами, препятствующими свободной эвакуации;
- к средствам пожаротушения должен быть обеспечен свободный доступ, средства должны находиться в исправном состоянии;
- система вентиляции при необходимости оснащается устройствами автоматического отключения при пожаре и огнезадерживающими клапанами;
- автоматическая пожарная сигнализация должна содержаться в работоспособном состоянии в соответствии с типовыми правилами технического обслуживания.

Заключение

В ходе выполнения дипломной работы была достигнута поставленная цель – разработано веб-приложение для управления задачами, обеспечивающее повышенную безопасность учётных записей пользователей.

В рамках работы проведён анализ предметной области, выявлены основные угрозы безопасности веб-приложений. Выполнен обзор существующих решений (Trello, Asana, Jira, Vikunja), показавший, что популярные системы не предоставляют расширенных механизмов безопасности в бесплатных версиях или имеют закрытый исходный код. На основе анализа спроектирована архитектура приложения и структура базы данных с учётом хеширования паролей, хранения TOTP-секретов и резервных кодов.

Реализованы следующие ключевые модули и механизмы:

- модуль аутентификации, включающий стандартный вход по паролю, двухфакторную аутентификацию на основе TOTP (RFC 6238), генерацию и проверку одноразовых резервных кодов, защиту от перебора паролей;
- модуль управления задачами с базовыми операциями (CRUD, фильтрация по статусу), привязкой задач к пользователю и ролевым разграничением;
- безопасное управление сессиями.

Разработанное приложение может использоваться в учебных целях для изучения современных механизмов защиты веб-приложений или внедрения в небольших организациях.

Список использованных источников

1. Годовой отчет Positive Technologies 2023. [Электронный ресурс]. URL: <https://ar2023.ptsecurity.com> (дата обращения: 11.04.2026).
2. Verizon Business. 2023 Data Breach Investigations Report. – 2023. [Электронный ресурс]. URL: <https://www.verizon.com/business/resources/reports/2023-data-breach-investigations-report-dbir.pdf> (дата обращения: 11.04.2026).
3. Документация по языку программирования Python. [Электронный ресурс]. URL: <https://www.python.org/> (дата обращения: 20.04.2026).
4. Официальный сайт среды разработки PyCharm. [Электронный ресурс]. URL: <https://www.jetbrains.com/pycharm/> (дата обращения: 20.04.2026).
5. Руководство по веб-фреймворку Django. – METANIT.COM, 2025. [Электронный ресурс]. URL: <https://metanit.com/python/django/> (дата обращения: 21.04.2026).
6. M'Raihi, D., Machani, S., Pei, M., Rydell, J. RFC 6238: TOTP: Time-Based One-Time Password Algorithm. – IETF, 2011. [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc6238> (дата обращения: 23.04.2026).
7. Что такое OWASP Top 10? – Ngenix, 2025. [Электронный ресурс]. URL: <https://ngenix.net/learning-center/what-is-aowasp> (дата обращения: 23.04.2026).

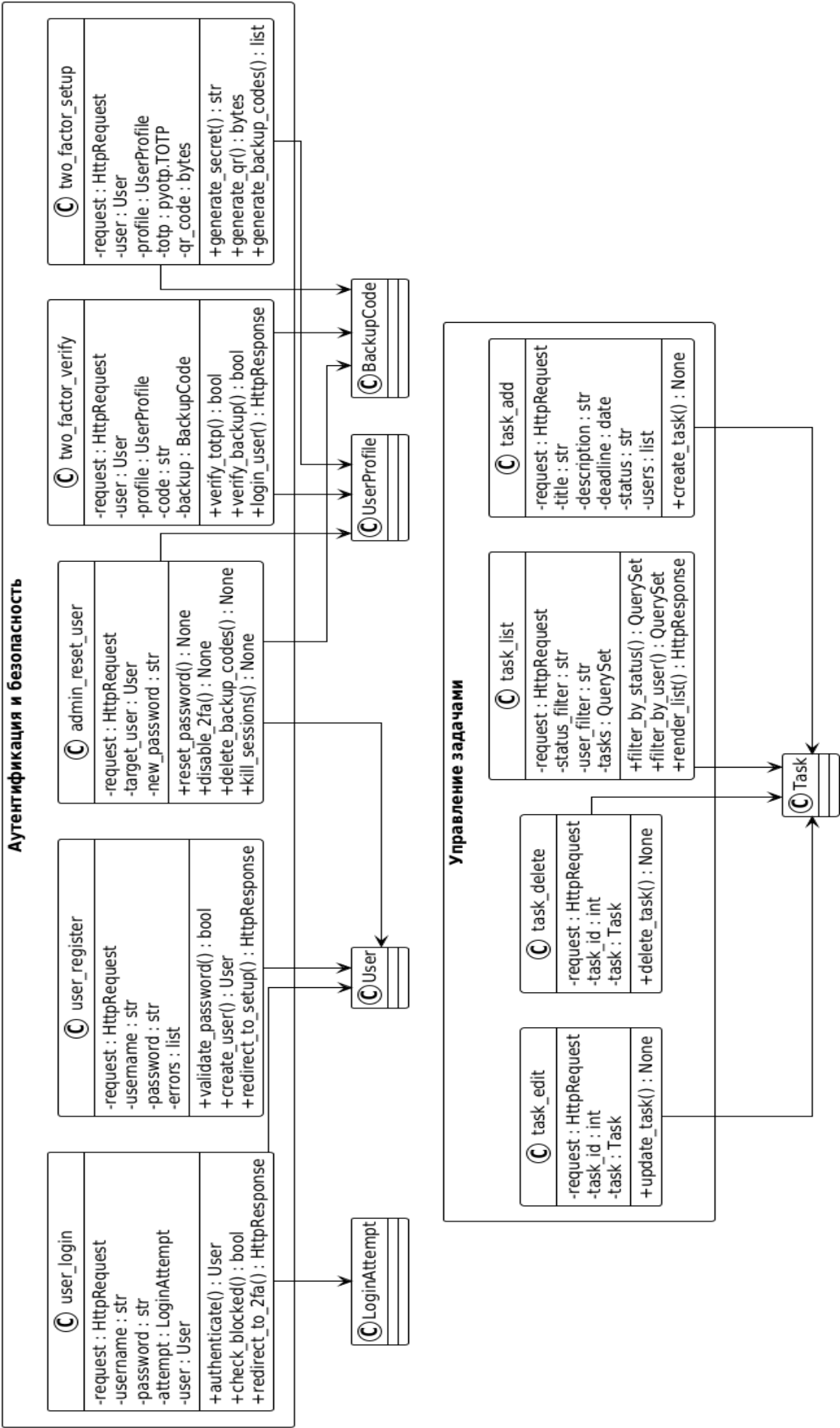
Приложение А

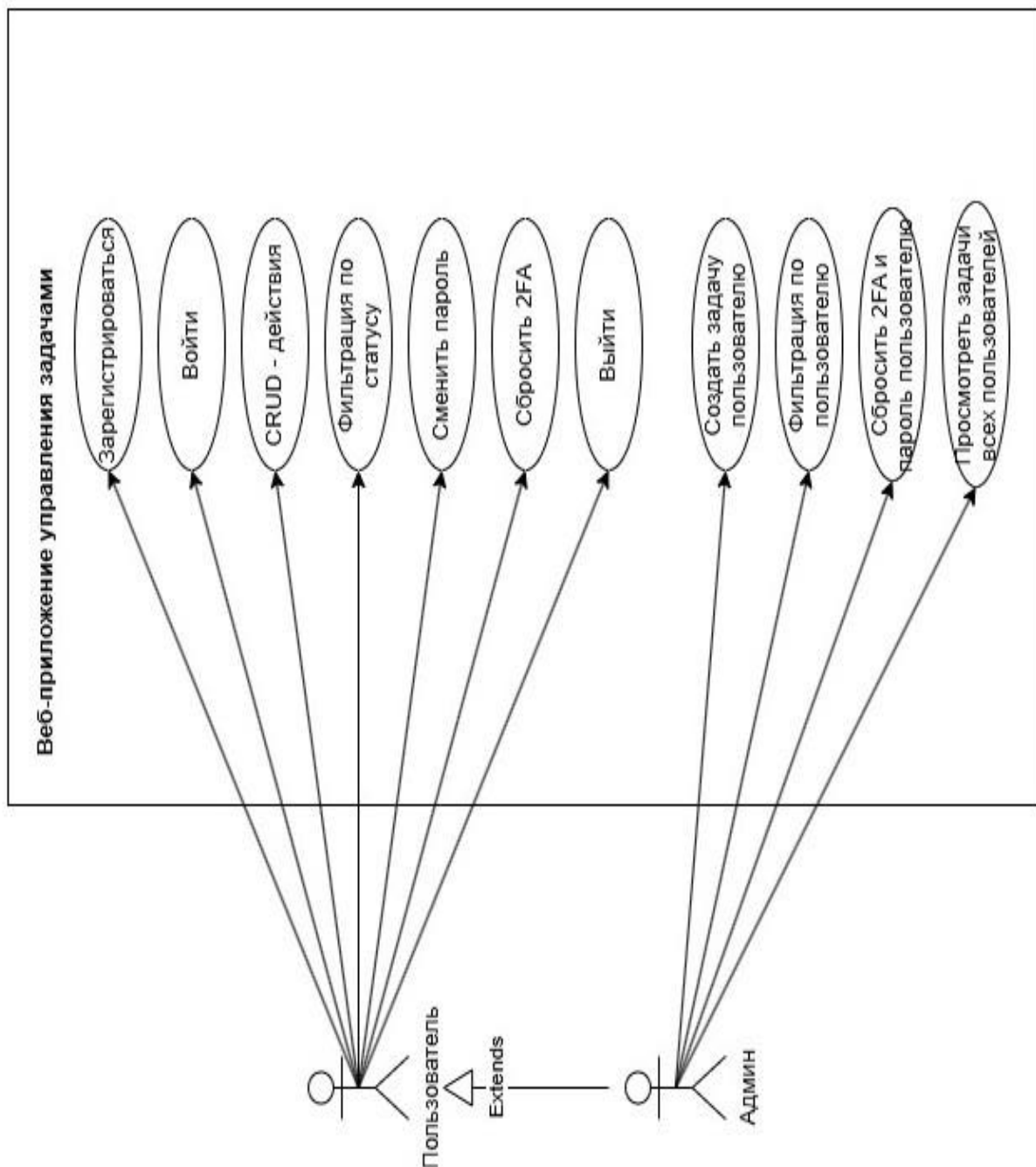
(обязательное)

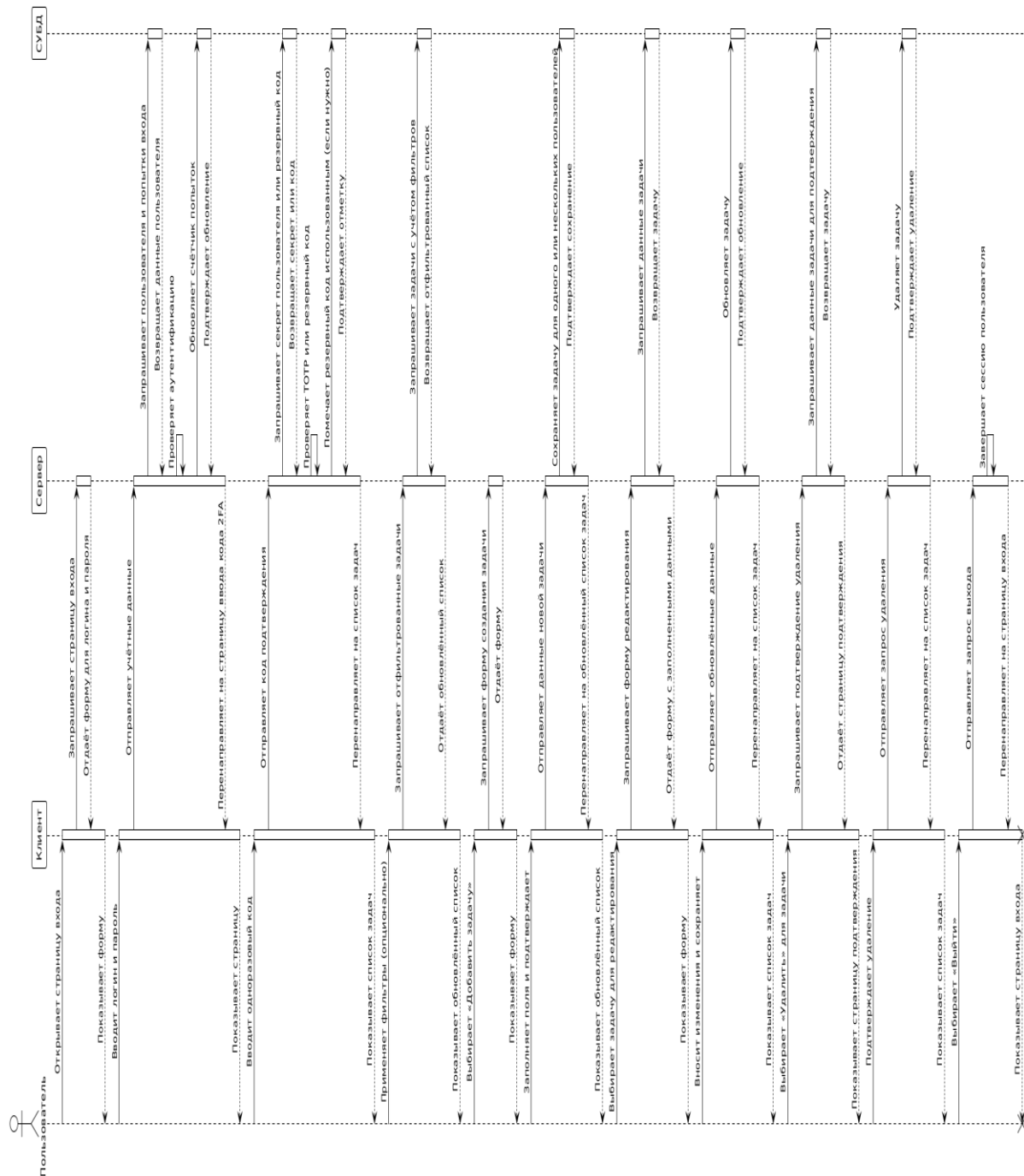
Веб-приложение управления задачами с повышенной системой безопасности

UML-диаграммы

					ПГУ 1.090301.07.001 ПЗ	Лист
						57
Изм.	Лист	№докум.	Подпись	Дата		

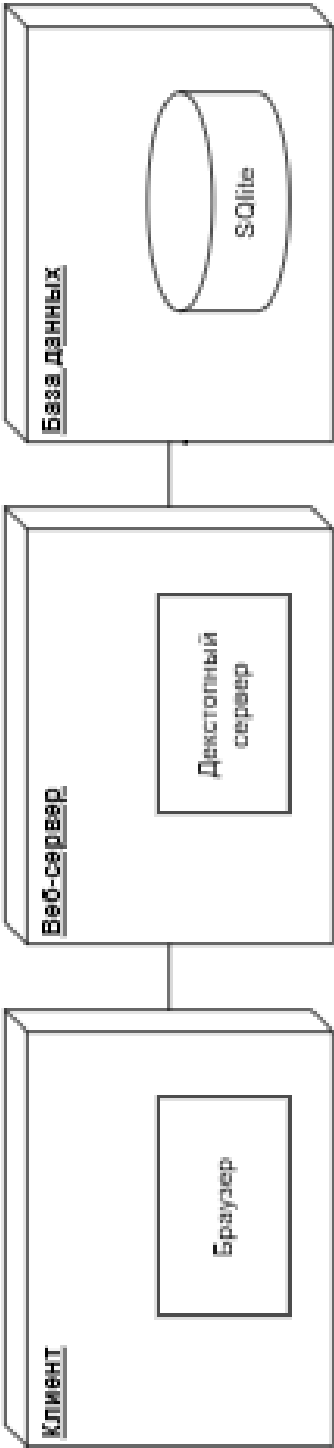






Веб-приложение упра

мой безопасности



«Файлviews.py»

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth import authenticate, login, logout,
update_session_auth_hash
from django.contrib.auth.decorators import login_required
from django.contrib.auth.models import User
from django.contrib.auth.forms import PasswordChangeForm
from django.utils import timezone
from django.contrib.auth.decorators import user_passes_test
from django.contrib.auth.hashers import make_password
from django.contrib.sessions.models import Session
from django.contrib import messages
from datetime import timedelta
from io import BytesIO
import pyotp
import secrets
import qrcode
import base64
from .models import Task, BackupCode, UserProfile, LoginAttempt
MAX_LOGIN_ATTEMPTS = 5
BLOCK_TIME_MINUTES = 10
def home(request):
    return render(request, «tasks/home.html»)
@login_required
def task_list(request):
    status_filter = request.GET.get(«status»)
    user_filter = request.GET.get(«user»)
    if request.user.is_staff:
        tasks = Task.objects.all()
    if user_filter:
        tasks = tasks.filter(user__id=user_filter)
    else:
        tasks = Task.objects.filter(user=request.user)
    if status_filter:
        tasks = tasks.filter(status=status_filter)
    tasks = tasks.order_by(«-created_at»)
    users = None
    if request.user.is_staff:
        users = User.objects.all()
    return render(request, «tasks/task_list.html», {
        «tasks»: tasks,
        «current_status»: status_filter,
        «users»: users,
        «current_user»: user_filter,
    })
@login_required
def task_add(request):
    users = User.objects.all() if request.user.is_staff else None
    if request.method == «POST»:
        title = request.POST.get(«title»)
        description = request.POST.get(«description»)
        deadline = request.POST.get(«deadline») or None
```

					ПГУ 1.090301.07.001 ПЗ	Лист
						62
Изм.	Лист	№докум.	Подпись	Дата		

```

status = request.POST.get(«status»)
if request.user.is_staff:
user_ids = request.POST.getlist(«users»)
selected_users = User.objects.filter(id__in=user_ids)
else:
selected_users = [request.user]
for u in selected_users:
Task.objects.create(
user=u,
title=title,
description=description,
deadline=deadline,
status=status,
)
return redirect(«task_list»)
return render(request, «tasks/task_add.html», {
«users»: users
})
@login_required
def task_edit(request, task_id):
task = Task.objects.get(id=task_id)
if request.method == 'POST':
task.title = request.POST.get('title')
task.description = request.POST.get('description')
task.status = request.POST.get('status')
task.deadline = request.POST.get('deadline') or None
task.save()
return redirect('task_list')
return render(request, 'tasks/task_edit.html', {'task': task})
@login_required
def task_delete(request, task_id):
task = get_object_or_404(Task, pk=task_id)
if request.method == 'POST':
task.delete()
return redirect('task_list')
return render(request, 'tasks/task_delete.html', {'task': task})
import re
def validate_password_strength(password):
errors = []
if len(password) < 8:
errors.append(«Пароль должен содержать минимум 8 символов.»)
if not re.search(r»[A-Z]», password):
errors.append(«Пароль должен содержать хотя бы одну заглавную букву (A-Z).»)
if not re.search(r»[a-z]», password):
errors.append(«Пароль должен содержать хотя бы одну строчную букву (a-z).»)
if not re.search(r»[0-9]», password):
errors.append(«Пароль должен содержать хотя бы одну цифру (0-9).»)
if not re.search(r»[!@#$%^&*(),.?\\»:{}|<>]», password):
errors.append(«Пароль должен содержать как минимум один спецсимвол (!@#$%^&*....).»)
return errors

```

```

def user_register(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        if User.objects.filter(username=username).exists():
            return render(request, 'tasks/register.html', {«error»:
            «Имяпользователязанято»})
        password_errors = validate_password_strength(password)
        if password_errors:
            return render(request, 'tasks/register.html', {
            «error»: password_errors
            })
        user = User.objects.create_user(username=username,
        password=password)
        request.session[«pre_2fa_user_id»] = user.id
        return redirect(«two_factor_setup»)
        return render(request, 'tasks/register.html')
    def user_login(request):
        if request.method == 'POST':
            username = request.POST.get('username')
            password = request.POST.get('password')
            attempt, _ = LoginAttempt.objects.get_or_create(username=username)
            if attempt.is_blocked():
                remaining = int((attempt.blocked_until -
                timezone.now()).total_seconds() // 60) + 1
                return render(request, 'tasks/login.html', {
                «error»: f«Слишком много попыток. Попробуйтечерез {remaining} мин.»
                })
            user = authenticate(request, username=username, password=password)
            if not user:
                attempt.attempts += 1
                if attempt.attempts >= MAX_LOGIN_ATTEMPTS:
                    attempt.blocked_until = timezone.now() +
                    timedelta(minutes=BLOCK_TIME_MINUTES)
                    attempt.save()
                    return render(request, 'tasks/login.html', {
                    «error»: «Неверный логин или пароль»
                    })
                attempt.attempts = 0
                attempt.blocked_until = None
                attempt.save()
            profile, _ = UserProfile.objects.get_or_create(user=user)
            request.session[«pre_2fa_user_id»] = user.id
            if not profile.two_factor_enabled:
                return redirect(«two_factor_setup»)
            else:
                return redirect(«two_factor_verify»)
            return render(request, 'tasks/login.html')
        def two_factor_verify(request):
            user_id = request.session.get(«pre_2fa_user_id»)
            if not user_id:
                return redirect(«login»)
            user = User.objects.get(id=user_id)

```

```

profile = user.userprofile
totp = pyotp.TOTP(profile.two_factor_secret)
if request.method == «POST»:
code = request.POST.get(«code»)
backup = BackupCode.objects.filter(
user=user,
code=code,
used=False
).first()
if backup:
backup.used = True
backup.save()
profile.two_factor_enabled = True
profile.save()
login(request, user)
del request.session[«pre_2fa_user_id»]
return redirect(«task_list»)
if totp.verify(code):
profile.two_factor_enabled = True
profile.save()
login(request, user)
del request.session[«pre_2fa_user_id»]
return redirect(«task_list»)
return render(request, «tasks/2fa_verify.html», {
«error»: «Неверный код»
})
return render(request, «tasks/2fa_verify.html»)
def two_factor_setup(request):
user_id = request.session.get(«pre_2fa_user_id»)
if not user_id:
return redirect(«login»)
user = User.objects.get(id=user_id)
profile = user.userprofile
if profile.two_factor_enabled:
return redirect(«task_list»)
if not profile.two_factor_secret:
profile.two_factor_secret = pyotp.random_base32()
profile.save()
existing_codes = BackupCode.objects.filter(
user=user,
used=False
)
if not existing_codes.exists():
backup_codes = generate_backup_codes(user)
else:
backup_codes = list(existing_codes.values_list(«code», flat=True))
totp = pyotp.TOTP(profile.two_factor_secret)
uri = totp.provisioning_uri(
name=user.username,
issuer_name=«Task Tracker»
)
qr = qrcode.make(uri)
buffer = BytesIO()

```

					ПГУ 1.090301.07.001 ПЗ	Лист
						65
Изм.	Лист	№докум.	Подпись	Дата		

```

qr.save(buffer, format=«PNG»)
qr_base64 = base64.b64encode(buffer.getvalue()).decode()
return render(request, «tasks/two_factor_setup.html», {
«qr_code»: qr_base64,
«secret»: profile.two_factor_secret,
«backup_codes»: backup_codes
})
@login_required
def reset_2fa(request):
user = request.user
profile = user.userprofile
profile.two_factor_secret = pyotp.random_base32()
profile.two_factor_enabled = False
profile.save()
BackupCode.objects.filter(user=user).delete()
backup_codes = generate_backup_codes(user)
from django.contrib.sessions.models import Session
for session in Session.objects.all():
data = session.get_decoded()
if data.get(«_auth_user_id») == str(user.id):
session.delete()
logout(request)
request.session[«pre_2fa_user_id»] = user.id
return redirect(«two_factor_setup»)
def generate_backup_codes(user, count=8):
codes = []
for _ in range(count):
code = secrets.token_hex(4)
BackupCode.objects.create(user=user, code=code)
codes.append(code)
return codes
@login_required
def change_password(request):
if request.method == «POST»:
form = PasswordChangeForm(request.user, request.POST)
if form.is_valid():
user = form.save()
for session in Session.objects.all():
data = session.get_decoded()
if data.get(«_auth_user_id») == str(user.id):
session.delete()
return redirect(«login»)
else:
form = PasswordChangeForm(request.user)
return render(request, «tasks/change_password.html», {«form»: form})
from django.contrib.sessions.models import Session
from django.contrib.auth.hashers import make_password
def password_reset_backup(request):
if request.method == «POST»:
username = request.POST.get(«username»)
code = request.POST.get(«code»)
new_password = request.POST.get(«new_password»)
try:

```

```

user = User.objects.get(username=username)
except User.DoesNotExist:
return render(request, «tasks/password_reset_backup.html», {
«error»: «Неверные данные»
})
backup = BackupCode.objects.filter(
user=user,
code=code,
used=False
).first()
if not backup:
return render(request, «tasks/password_reset_backup.html», {
«error»: «Неверные данные»
})
password_errors = validate_password_strength(new_password)
if password_errors:
return render(request, «tasks/password_reset_backup.html», {
«error»: password_errors
})
user.password = make_password(new_password)
user.save()
backup.used = True
backup.save()
for session in Session.objects.all():
data = session.get_decoded()
if data.get(«_auth_user_id») == str(user.id):
session.delete()
return redirect(«login»)
return render(request, «tasks/password_reset_backup.html»)
def is_admin(user):
return user.is_staff
@login_required
@user_passes_test(is_admin)
def admin_reset_user(request, user_id):
target_user = get_object_or_404(User, id=user_id)
if request.method == «POST»:
new_password = request.POST.get(«new_password»)
if not new_password:
messages.error(request, «Введите новый пароль»)
return redirect(request.path)
target_user.set_password(new_password)
target_user.save()
profile = target_user.userprofile
profile.two_factor_enabled = False
profile.otp_secret = «»
profile.save()
target_user.backupcode_set.all().delete()
sessions = Session.objects.all()
for session in sessions:
data = session.get_decoded()
if data.get('_auth_user_id') == str(target_user.id):
session.delete()
messages.success(request, «Пароли 2FA успешно сброшены»)

```

```

return redirect(«task_list»)
return render(request, «tasks/admin_reset_user.html», {
«target_user»: target_user
})
def user_logout(request):
logout(request)
return redirect('login')

```

«Файл urls.py»

```

from django.urls import path
from . import views
urlpatterns = [
path('', views.task_list, name='task_list'),
path('add/', views.task_add, name = 'task_add'),
path('edit/<int:task_id>', views.task_edit, name='task_edit'),
path('delete/<int:task_id>', views.task_delete,
name='task_delete'),
path('login/', views.user_login, name='login'),
path('register/', views.user_register, name='register'),
path('logout/', views.user_logout, name='logout'),
path(«2fa/», views.two_factor_verify, name=«two_factor_verify»),
path(«2fa/setup/», views.two_factor_setup, name=«two_factor_setup»),
path(«password/change/», views.change_password,
name=«change_password»),
path(«password/reset/», views.password_reset_backup,
name=«password_reset_backup»),
path(«2fa/reset/», views.reset_2fa, name=«reset_2fa»),
path('admin/user/<int:user_id>/reset/', views.admin_reset_user,
name='admin_reset_user'),]

```

«Файл models.py»

```

from django.db import models
from django.contrib.auth.models import User
from django.utils import timezone
from datetime import timedelta
import secrets
class Task(models.Model):
STATUS_CHOICES = [
(«backlog», «Квыполнению»),
(«progress», «В работе»),
(«hold», «На паузе»),
(«done», «Выполнено»),
]
title = models.CharField(max_length=200,
verbose_name=«Названиезадачи»)
description = models.TextField(blank=True, verbose_name=«Описание»)
created_at = models.DateTimeField(auto_now_add=True)
deadline = models.DateField(null=True, blank=True,
verbose_name=«Дедлайн»)
is_completed = models.BooleanField(default=False,
verbose_name=«Выполнено»)

```

```

user = models.ForeignKey(User, on_delete=models.CASCADE,
verbose_name=«Пользователь»)
status = models.CharField(max_length=20, choices=STATUS_CHOICES,
default=«backlog», verbose_name=«Статус»)
def __str__(self):
return self.title
class Meta:
verbose_name = «Задача»
verbose_name_plural = «Задачи»
class UserProfile(models.Model):
user = models.OneToOneField(User, on_delete=models.CASCADE)
two_factor_enabled = models.BooleanField(default=False)
two_factor_secret = models.CharField(max_length=32, blank=True,
null=True)
def __str__(self):
return f»Profile: {self.user.username}»
class BackupCode(models.Model):
user = models.ForeignKey(User, on_delete=models.CASCADE)
code = models.CharField(max_length=12)
used = models.BooleanField(default=False)
created_at = models.DateTimeField(auto_now_add=True)
def __str__(self):
return f»{self.user.username} - {self.code}»
class LoginAttempt(models.Model):
username = models.CharField(max_length=150)
attempts = models.PositiveIntegerField(default=0)
last_attempt = models.DateTimeField(auto_now=True)
blocked_until = models.DateTimeField(null=True, blank=True)
def is_blocked(self):
return self.blocked_until and self.blocked_until > timezone.now()
def __str__(self):
return f»{self.username} ({self.attempts})»

```

«Файл two_factor_setup.html»

```

{% extends «tasks/base.html» %}
{% block content %}
<div class=«row justify-content-center»>
<div class=«col-md-6»>
<h3 class=«mb-3»>Настройка двухфакторной аутентификации</h3>
<p><b>Имя аккаунта:</b> {{ username }}</p>
<div class=«text-center mb-4»>
<img src=«data:image/png;base64,{{ qr_code }}» alt=«QR Code»
class=«img-fluid»>
</div>
<p class=«mt-3»>
Отсканируйте QR-код выше в приложении аутентификации
(ЯндексКлюч, Google Authenticator ит.д.)
</p>
<p><b>Или введите ключ вручную:</b></p>
<div class=«alert alert-secondary text-break text-center»>
{{ secret }}
</div>

```

```

<hr>
<h5 class=«mt-4»>Одноразовые резервные коды</h5>
<div class=«alert alert-warning»>
<b>Сохраните эти коды!</b><br>
Каждый код можно использовать только один раз.
</div>
<ul class=«list-group mb-3»>
{% for code in backup_codes %}
<li class=«list-group-item text-monospace text-center»>
{{ code }}
</li>
{% endfor %}
</ul>
<div class=«d-grid gap-2 d-md-flex justify-content-md-end mt-4»>
<a href=«{% url 'two_factor_verify' %}» class=«btn btn-primary»>
Я добавил ключ
</a>
</div>
</div>
</div>
{% endblock %}
Файл task_list.html
{% extends «tasks/base.html» %}
{% block title %}Мои задачи- Task Tracker{% endblock %}
{% block content %}
<div class=«d-flex justify-content-between align-items-center mb-3»>
<h2 class=«m-0»>Мои задачи</h2>
<a href=«{% url 'task_add' %}» class=«btn btn-success»>+
Добавить задачу</a>
</div>
<form method=«get» class=«mb-4»>
<div class=«row g-2 align-items-end»>
{% if user.is_staff %}
<div class=«col-md-4»>
<label class=«form-label»>Фильтр по пользователю</label>
<select name=«user» class=«form-select»>
<option value=«>>Все пользователи</option>
{% for u in users %}
<option value=«{{ u.id }}»
{% if current_user == u.id|stringformat:»s» %}selected{% endif %}>
{{ u.username }}
</option>
{% endfor %}
</select>
</div>
{% endif %}
<div class=«col-md-4»>
<label class=«form-label»>Фильтр по статусу</label>
<select name=«status» class=«form-select»>
<option value=«>>Все задачи</option>
<option value=«backlog» {% if current_status ==
«backlog» %}selected{% endif %}>К выполнению</option>

```

```

<option value=«progress» {% if current_status ==
«progress»}%>selected{% endif %}>В работе</option>
<option value=«hold» {% if current_status == «hold»}%>selected{%
endif %}>На паузе</option>
<option value=«done» {% if current_status == «done»}%>selected{%
endif %}>Выполнено</option>
</select>
</div>
<div class=«col-md-2»>
<button class=«btn btn-outline-primary w-100» type=«submit»>
Применить
</button>
</div>
</div>
</form>
{% if tasks %}
<div class=«list-group»>
{% for task in tasks %}
<div class=«list-group-item»>
<div class=«d-flex justify-content-between»>
<h5 class=«mb-1»>{{ task.title }}</h5>
{% if task.status == «backlog» %}
<span class=«badge bg-secondary»>К выполнению</span>
{% elif task.status == «progress» %}
<span class=«badge bg-warning text-dark»>В работе</span>
{% elif task.status == «hold» %}
<span class=«badge bg-primary»>На паузе</span>
{% elif task.status == «done» %}
<span class=«badge bg-success»>Выполнено</span>
{% endif %}
</div>
<p class=«mb-1»>{{ task.description }}</p>
{% if user.is_staff %}
<small class=«text-muted d-block»>
Пользователь: <strong>{{ task.user.username }}</strong>
</small>
{% endif %}
<small class=«text-muted»>
Дедлайн:
{% if task.deadline %}
{{ task.deadline }}
{% else %}
не задан
{% endif %}
</small>
<div class=«mt-3»>
<a href=«{% url 'task_edit' task.id %}» class=«btn btn-primary btn-
sm»>✎ Редактировать</a>
<a href=«{% url 'task_delete' task.id %}» class=«btn btn-danger btn-
sm»>🗑 Удалить</a>
{% if user.is_staff %}

```

```

<a href=«{% url 'admin_reset_user' task.user.id %}» class=«btn btn-
warning btn-sm»>
Сбросить доступ
</a>
{% endif %}
</div>
</div>
{% endfor %}
</div>
{% else %}
<pclass=«text-muted»>Задачпоканет.</p>
{% endif %}
{% endblock %}

```

«Файл task_add.html»

```

{% extends «tasks/base.html» %}
{% block title %}Создатьзадачу- Task Tracker{% endblock %}
{% block content %}
<div class=«row justify-content-center»>
<div class=«col-12 col-md-8 col-lg-6»>
<div class=«card shadow-sm»>
<div class=«card-body»>
<h3 class=«mb-4 text-center»>Создатьзадачу</h3>
<form method=«POST»>
{% csrf_token %}
{% if user.is_staff %}
<div class=«mb-3»>
<label class=«form-label»>Назначить пользователям</label>
<select class=«form-select» name=«users» multiple required>
{% for u in users %}
<option value=«{{ u.id }}»>{{ u.username }}</option>
{% endfor %}
</select>
<div class=«form-text»>
Можно выбрать несколько пользователей (Ctrl / Shift)
</div>
</div>
{% endif %}
<div class=«mb-3»>
<label class=«form-label»>Названиезадачи</label>
<input type=«text» class=«form-control» name=«title» required>
</div>
<div class=«mb-3»>
<label class=«form-label»>Описание</label>
<textarea class=«form-control» name=«description»
rows=«3»></textarea>
</div>
<div class=«mb-3»>
<label class=«form-label»>Дедлайн</label>
<input type=«date» class=«form-control» name=«deadline»>
</div>
<div class=«mb-3»>

```

```

<label class=«form-label»>Статус</label>
<select class=«form-select» name=«status»>
<option value=«backlog»>К выполнению</option>
<option value=«progress»>В работе</option>
<option value=«hold»>На паузе</option>
<option value=«done»>Выполнено</option>
</select>
</div>
<button class=«btn btn-primary w-100 mb-2»
type=«submit»>Создать</button>
<a href=«{% url 'task_list' %}» class=«btn btn-secondary w-
100»>Отмена</a>
</form>
</div>
</div>
</div>
{% endblock %}

```

«Файл base.html»

```

<!DOCTYPE html>
<html lang=«ru»>
<head>
<meta charset=«UTF-8»>
<title>{% block title %}Task Tracker{% endblock %}</title>
<link
href=«https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap
p.min.css» rel=«stylesheet»>
</head>
<body class=«bg-light»>
<nav class=«navbar navbar-expand-lg navbar-dark bg-dark mb-4»>
<div class=«container»>
<a class=«navbar-brand» href=«{% url 'home' %}»>Task Tracker</a>
<button class=«navbar-toggler» type=«button» data-bs-
toggle=«collapse» data-bs-target=«#navbarNav»>
<span class=«navbar-toggler-icon»></span>
</button>
<div class=«collapse navbar-collapse justify-content-end»
id=«navbarNav»>
{% if user.is_authenticated %}
<ul class=«navbar-nav align-items-center»>
<li class=«nav-item me-3 text-white»>
Пользователь, <strong>{{ user.username }}</strong>!
</li>
<li class=«nav-item me-2»>
<a class=«btn btn-outline-light» href=«{% url 'logout' %}»>Выйти</a>
</li>
<li class=«nav-item me-2»>
<a class=«btn btn-outline-warning» href=«{% url 'change_password'
%}»>
Сменить пароль
</a>

```

```

</li>
<li class=«nav-item me-2»>
<a class=«btn btn-outline-info» href=«{% url 'reset_2fa' %}»>
Перевязать 2FA
</a>
</li>
</ul>
{% else %}
<ul class=«navbar-nav align-items-center»>
<li class=«nav-item me-2»>
<a class=«btn btn-outline-light» href=«{% url 'login' %}»>Войти</a>
</li>
<li class=«nav-item»>
<a class=«btn btn-success» href=«{% url 'register'
%}»>Регистрация</a>
</li>
</ul>
{% endif %}
</div>
</div>
</nav>
<div class=«container»>
{% block content %}{% endblock %}
</div>
<script
src=«https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.
bundle.min.js»></script>
</body>
</html>

```

Приложение В

(обязательное)

Веб-приложение управления задачами с повышенной системой безопасности

Презентация

					ПГУ 1.090301.07.001 ПЗ	Лист
						75
Изм.	Лист	№докум.	Подпись	Дата		

Веб-приложение для управления задачами с повышенной системой безопасности

Автор работы: студент гр. 22BBB1 Лёвин Артём Дмитриевич
Научный руководитель: к.т.н., доцент каф. ВТ Токарев Андрей Николаевич

Цель работы – разработка веб-приложения для управления задачами, обеспечивающего повышенную безопасность учётных записей пользователей за счёт двухфакторной аутентификации, резервных кодов, защиты от перебора паролей и безопасного управления сессиями.

Задачи:

- провести анализ предметной области;
- выполнить обзор существующих решений и их механизмов безопасности;
- обосновать актуальность разработки;
- спроектировать архитектуру;
- реализовать модули защиты и модуль управления задачами;
- оценить результаты.

Анализ предметной области

Что такое системы управления задачами?

Системы управления задачами – это программные инструменты, которые помогают отдельным пользователям и целым командам планировать работу, ставить задачи, назначать исполнителей, следить за сроками и контролировать результат. Они используются повсеместно: от личных списков дел до корпоративного управления проектами.

Почему безопасность стала критически важной?

Современные системы хранят не только рабочую информацию, но и персональные данные пользователей: логины, пароли, адреса электронной почты, а иногда и конфиденциальное описание проектов. При этом доступ к таким системам чаще всего осуществляется через интернет, что делает их уязвимыми для атак.

Основные угрозы безопасности веб-приложений:

1. атаки перебора паролей – злоумышленник автоматически перебирает тысячи комбинаций, пытаясь угадать пароль;
2. фишинг – пользователь сам вводит свой пароль на поддельной странице, думая, что это настоящий сервис;
3. утечки баз данных – при взломе сервера злоумышленник получает хеши паролей;
4. перехват сессии – после того как пользователь вошёл в систему, злоумышленник может украсть идентификатор сессии и получить доступ к учётной записи.

Обзор существующих решений

Название аналога	Преимущество	Недостаток
Trello	Наглядность (канбан-доски), бесплатная базовая версия	Закрытый код, нет двухфакторной аутентификации и защиты от перебора
Asana	Расширенный функционал(списки, доски, зависимости задач), поддержка командной работы	Поддерживает 2FA только через сторонние SSO-провайдеры, сложный интерфейс для новичков
Jira	Встроенный аудит, гибкая настройка рабочих процессов, есть защиты от перебора	Избыточность для малых проектов, система сложна в настройке, 2FA только через Atlassian Access
Vikunja	Открытый код, самостоятельный хостинг, присутствует ролевая модель, есть 2FA	Отсутствует защита от перебора пароля, нет шифрования полей задач

Обоснование актуальности

- рост числа кибератак на учётные записи;
- большинство бесплатных систем не имеют 2FA и защиты от перебора;
- открытые аналоги не реализуют резервные коды и инвалидацию сессий;
- нет готового бесплатного решения с полным набором современных механизмов безопасности;
- разработка собственного приложения позволяет закрыть эти недостатки и может послужить учебным примером.

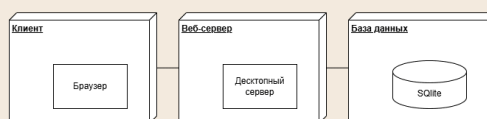
Проектирование архитектуры

Архитектура: клиент-серверная.

Стек технологий: Python 3.12, веб-фреймворк Django, SQLite.

Среда разработки: PyCharm Community edition.

Клиент – браузер.
Использует HTML-страницы для пользования системой



Сервер – обрабатывает HTTP-запросы, управляет доступом, взаимодействует с БД

База данных – хранит информацию: учётные записи, задачи

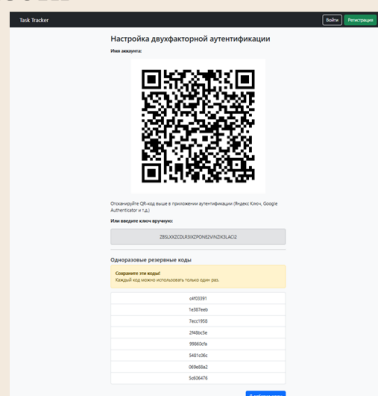
Реализация модуля безопасности

Двухфакторная аутентификация по стандарту TOTP:

- При регистрации пользователя генерируется секретный ключ и QR-код, пользователь его сканирует приложением-аутентификатором, далее при входе запрашивается 6-значный код, который меняется каждые 30 секунд.

Резервные коды:

- При настройке 2FA создаются 8 одноразовых кодов, каждый из которых можно использовать один раз для входа. Коды хранятся в базе данных с отметкой об использовании.



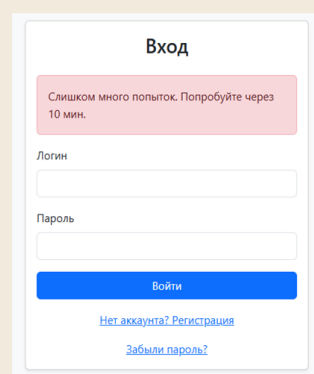
Реализация модуля безопасности

Защита от перебора паролей:

- Ведётся учёт неудачных попыток входа для каждого логина, после 5 неверных попыток учётная запись блокируется на 10 минут, при успешном же входе счётчик сбрасывается.

Управление сессиями:

- При смене пароля или 2FA, закрытии браузера, фактическом выходе или же по истечении «времени жизни» сессия пользователя завершается, это предотвращает использование старых сессий.



Реализация модуля управления задачами и ролевого разграничения

Функции управления задачами: создание, редактирование, удаление, просмотр. У самой задачи есть: название, описание, статус и дедлайн. У пользователя есть только фильтрация по статусу задачи

Ролевая модель: обычный пользователь видит и управляет только своими задачами. Администратор же видит задачи всех пользователей, может создать задачи другим и появляется новая фильтрация, так же есть кнопка «Сбросить доступ»

Тестирование

Результат защиты от перебора

Результат неверного кода 2FA

Заключение

Разработано веб-приложение, решающее поставленные задачи. Обеспечена безопасность учётных данных, а само приложение может использоваться в учебных целях.

Рекомендации к дальнейшей разработке:

- шифрование отдельных полей;
- отдельный модуль аудита;
- переход на PostgreSQL и контейнеризация Docker;
- перевод сервера на протокол HTTPS.