**пункт 1: Определение требований в виде технического задания (ТЗ):**

Техническое задание: Веб-приложение для управления проектами и задачами

1. Определение требований

1.1. Назначение системы

Разработка веб-приложения для управления проектами и задачами, ориентированного на командную работу. Система должна предоставлять пользователям возможность создавать проекты, ставить задачи, назначать исполнителей, отслеживать статус выполнения и вести коммуникацию через комментарии.

1.2. Цели приложения

• Централизованное управление проектами.

• Планирование и распределение задач.

• Мониторинг выполнения задач и соблюдения сроков.

• Командное взаимодействие через встроенные комментарии.

• Гибкое управление пользователями и ролями.

1.3. Функциональные требования

Проекты

Функция Описание

Создание проекта Пользователь может создать новый проект с указанием названия и описания

Редактирование проекта Изменение информации о проекте

Удаление проекта Удаление проекта и всех связанных задач

Просмотр списка проектов Отображение всех доступных пользователю проектов

Назначение владельца проекта Один пользователь — владелец проекта (автор или назначенный админом)

Задачи

Функция Описание

Создание задачи Создание задачи в рамках выбранного проекта

Назначение исполнителя Выбор пользователя, ответственного за выполнение

Установка дедлайна Указание даты завершения задачи

Выбор статуса Открыта / В работе / Завершена / Отложена

Установка приоритета Низкий / Средний / Высокий

Редактирование и удаление задач Изменение или удаление существующей задачи

Фильтрация и сортировка задач По статусу, исполнителю, дедлайну, приоритету

Пользователи

Функция Описание

Регистрация и вход Создание учётной записи и вход в систему

Роли и права доступа Разделение на обычных пользователей и администраторов

Просмотр задач Доступ к своим задачам или задачам проекта (в зависимости от роли)

Комментарии

Функция Описание

Добавление комментария Оставление комментария к задаче

Отображение комментариев Комментарии упорядочены по дате/времени

Информация об авторе Каждый комментарий содержит имя автора и дату публикации

Дополнительные функции

Функция Описание

Панель управления (Dashboard) Краткая статистика по проектам и задачам

Поиск и фильтрация Быстрый поиск задач по ключевым параметрам

Адаптивный дизайн Удобная работа на ПК и мобильных устройствах

Уведомления (опционально) Уведомления о назначении задачи, новых комментариях и дедлайнах

1.4. Нефункциональные требования

• Интерфейс должен быть интуитивно понятным.

• Производительность: отклик интерфейса — менее 1 секунды.

• Надёжность: сохранность данных при сбоях, использование транзакций.

• Безопасность: авторизация, защита от XSS/CSRF, безопасное хранение паролей.

**пункт 2: Проектирование модели данных с текстовым описанием и логической структурой:**

2. Проектирование модели данных

2.1. Основные сущности

Приложение будет оперировать следующими сущностями:

1. User (Пользователь)

Представляет зарегистрированного пользователя системы.

2. Project (Проект)

Объединяет задачи и принадлежит одному пользователю (владельцу).

3. Task (Задача)

Конкретная единица работы в рамках проекта. Может быть назначена пользователю.

4. Comment (Комментарий)

Сообщение пользователя, оставленное к задаче.

2.2. Атрибуты сущностей

User

Поле Тип Описание

id AutoField (PK) Уникальный идентификатор пользователя

username CharField Имя пользователя

email EmailField Электронная почта

password CharField (hashed) Хешированный пароль

is\_admin BooleanField Признак администратора

date\_joined DateTimeField Дата регистрации

Project

Поле Тип Описание

id AutoField (PK) Уникальный идентификатор проекта

name CharField Название проекта

description TextField Описание проекта

owner ForeignKey(User) Владелец проекта

created\_at DateTimeField Дата создания проекта

Task

Поле Тип Описание

id AutoField (PK) Уникальный идентификатор задачи

project ForeignKey(Project) Проект, к которому относится задача

title CharField Название задачи

description TextField Подробное описание задачи

assignee ForeignKey(User) Назначенный исполнитель (может быть null)

status CharField Статус задачи (Открыта, В работе и т.д.)

priority CharField Приоритет задачи (Низкий/Средний/Высокий)

due\_date DateField Дата дедлайна

created\_at DateTimeField Дата создания

updated\_at DateTimeField Дата последнего изменения

Comment

Поле Тип Описание

id AutoField (PK) Уникальный идентификатор комментария

task ForeignKey(Task) Задача, к которой относится комментарий

author ForeignKey(User) Автор комментария

content TextField Текст комментария

created\_at DateTimeField Дата и время публикации

2.3. Связи между сущностями (логическая структура)

• User 1 — N Project (один пользователь может владеть многими проектами)

• Project 1 — N Task

• User 1 — N Task (назначение исполнителя)

• Task 1 — N Comment

• User 1 — N Comment (автор комментария)

2.4. Возможные расширения

• Таблица ProjectParticipant для добавления нескольких участников в проект.

• История изменений задач (таблица логов).

• Прикрепляемые файлы (таблица Attachment).

**пункта 3: Реализация (Backend — Django) как части технического плана проекта:**

3. Реализация (Backend — Django)

3.1. Инициализация проекта

• Установка виртуального окружения:

python -m venv venv

source venv/bin/activate

pip install django

• Создание проекта:

django-admin startproject project\_manager

cd project\_manager

• Создание приложений:

python manage.py startapp users

python manage.py startapp projects

python manage.py startapp tasks

python manage.py startapp comments

3.2. Настройка базового проекта

• Добавление приложений в INSTALLED\_APPS

• Подключение базы данных

• Настройка статических и медиа-файлов

3.3. Модели (models.py)

users.models.User

from django.contrib.auth.models import AbstractUser

class User(AbstractUser):

is\_admin = models.BooleanField(default=False)

projects.models.Project

class Project(models.Model):

name = models.CharField(max\_length=255)

description = models.TextField(blank=True)

owner = models.ForeignKey('users.User', on\_delete=models.CASCADE)

created\_at = models.DateTimeField(auto\_now\_add=True)

tasks.models.Task

class Task(models.Model):

project = models.ForeignKey(Project, on\_delete=models.CASCADE)

title = models.CharField(max\_length=255)

description = models.TextField()

assignee = models.ForeignKey('users.User', null=True, blank=True, on\_delete=models.SET\_NULL)

status = models.CharField(max\_length=50, choices=[...])

priority = models.CharField(max\_length=20, choices=[...])

due\_date = models.DateField()

created\_at = models.DateTimeField(auto\_now\_add=True)

updated\_at = models.DateTimeField(auto\_now=True)

comments.models.Comment

class Comment(models.Model):

task = models.ForeignKey(Task, on\_delete=models.CASCADE)

author = models.ForeignKey('users.User', on\_delete=models.CASCADE)

content = models.TextField()

created\_at = models.DateTimeField(auto\_now\_add=True)

3.4. Миграции и база данных

• Команды:

python manage.py makemigrations

python manage.py migrat

3.5. Админ-панель

• Регистрация моделей:

admin.site.register(Project)

admin.site.register(Task)

admin.site.register(Comment)

• Подключение кастомной модели пользователя:

AUTH\_USER\_MODEL = 'users.User'

⸻

3.6. Представления (views.py)

• Для начала — использовать class-based views (ListView, CreateView, DetailView, UpdateView)

• Реализовать:

• Список проектов

• Детали проекта и его задачи

• CRUD для задач

• Комментарии

⸻

3.7. URL-маршруты

• Использовать include() для каждого приложения:

path('projects/', include('projects.urls')),

path('tasks/', include('tasks.urls')),

• Внутри каждого urls.py — маршруты на list, detail, create, update, delete

⸻

3.8. Аутентификация

• Использовать встроенные представления Django

• Защита маршрутов

• Шаблоны для входа и регистрации

3.9. Работа с формами

• Использовать ModelForm для создания/обновления задач и проектов

• Добавление валидации

3.10. API (опционально)

• Подключить Django REST Framework

• Создание сериализаторов для моделей

• Реализация ViewSet и маршрутов через routers

**пункт 4: Реализация (Frontend) в рамках проекта веб-приложения для управления проектами и задачами:**

4. Реализация (Frontend)

4.1. Структура шаблонов Django

Базовый шаблон (base.html)

• Общая структура страницы (HTML5)

• Подключение Bootstrap или Tailwind CSS

• Подключение пользовательских CSS/JS

• Навигационное меню (navbar) с авторизацией

• Блоки

Шаблоны для страниц

• projects/project\_list.html — список проектов

• projects/project\_detail.html — просмотр проекта и его задач

• tasks/task\_form.html — создание/редактирование задачи

• tasks/task\_detail.html — просмотр задачи, комментарии

• comments/comment\_form.html — добавление комментария

• registration/login.html, registration/signup.html — формы входа/регистрации

• dashboard.html — пользовательская панель с активными проектами и задачами

4.2. Стилизация интерфейса

CSS

• Использование Bootstrap 5 или Tailwind CSS для адаптивной верстки

• Подключение кастомного CSS-файла для доработок

UX/UI Элементы

• Карточки для задач и проектов

• Использование цветовых меток (label/badge) для приоритетов, статусов

• Иконки

4.3. Интерфейс пользователя (UI)

Раздел Описание

Главная панель Список всех проектов пользователя, кнопка “Создать проект”

Проект Название, описание, список задач, кнопка “Добавить задачу”

Задача Название, исполнитель, дедлайн, приоритет, статус, список комментариев

Форма задачи Поля ввода (проект, исполнитель, описание, приоритет, дедлайн)

Комментарии Список комментариев, форма добавления нового

Навигация Меню: “Мои проекты”, “Задачи”, “Создать”, “Выйти”, “Войти/Регистрация”

4.4. Динамика и взаимодействие

JavaScript (JS)

• Подключение Bootstrap JS

• Использование AJAX (fetch или jQuery) для:

• Асинхронного добавления комментариев

• Обновления статуса задачи без перезагрузки

Дополнительно (опционально)

• Использование Django Messages Framework для уведомлений (успех/ошибка)

• Отображение всплывающих окон (modal) для быстрого редактирования задачи

• Подсветка просроченных задач

4.5. Адаптивность

• Все страницы должны корректно отображаться на телефонах и планшетах

• Использование Bootstrap Grid или Tailwind Flex/Grid

• Тестирование интерфейса через DevTools

4.6. Интерактивные компоненты (опционально)

• Фильтры задач (по статусу, исполнителю и т.п.)

• Канбан-доска (если планируется)

• Подсветка активного меню

**пункт 5: Тестирование веб-приложения на Django для управления проектами и задачами:**  
  
  
**5. Тестирование**  
  
**5.1. Цели тестирования**  
 • Проверить корректность бизнес-логики (создание/редактирование задач и проектов)  
 • Обеспечить надёжную работу всех представлений и маршрутов  
 • Защитить систему от несанкционированного доступа  
 • Убедиться, что формы работают корректно, включая валидацию  
  
  
**5.2. Виды тестирования**  
  
**Тип** **Описание**  
**Unit-тесты** Проверка отдельных функций и моделей  
**Интеграционные тесты** Проверка взаимодействия компонентов (модели + представления)  
**Функциональные тесты** Проверка логики с точки зрения пользователя (например, создание задачи)  
**UI/End-to-End (опционально)** Автоматизация тестов с Selenium или Playwright  
**Тестирование безопасности** Проверка аутентификации, прав доступа  
  
  
  
**5.3. Используемые технологии**  
 • Встроенный модуль unittest (входит в Django)  
 • pytest (опционально)  
 • coverage — для оценки покрытия кода тестами  
 • Selenium — для автоматизации UI-тестов (опционально)  
  
**5.4. Примеры тестов**  
  
**Тест моделей**  
  
from django.test import TestCase  
from projects.models import Project  
from users.models import User  
  
class ProjectModelTest(TestCase):  
    def test\_create\_project(self):  
        user = User.objects.create\_user(username='test', password='1234')  
        project = Project.objects.create(name="Test", description="Test desc", owner=user)  
        self.assertEqual(str(project), "Test")  
  
**Тест представлений**  
  
from django.urls import reverse  
from django.test import Client  
  
class ProjectViewTest(TestCase):  
    def setUp(self):  
        self.client = Client()  
        self.user = User.objects.create\_user(username='test', password='1234')  
  
    def test\_project\_list\_requires\_login(self):  
        response = self.client.get(reverse('projects:list'))  
        self.assertRedirects(response, '/login/?next=/projects/')  
  
  
  
**5.5. Покрытие кода**  
 • Запуск покрытия:  
  
coverage run manage.py test  
coverage report -m  
  
  
  
**5.6. Тестирование форм**  
  
Проверка:  
 • Валидации полей (пустые, неправильные данные)  
 • Формат даты дедлайна  
 • Уведомления об ошибках  
  
  
**5.7. Тестирование безопасности**  
 • Попытка редактировать задачу другим пользователем  
 • Попытка просматривать чужие проекты  
 • Тесты доступа к URL’ам без авторизации  
  
  
**5.8. Подготовка к CI**  
 • Интеграция с GitHub Actions или GitLab CI:  
 • Автоматический запуск тестов при пуше  
 • Отчёты о покрытии

**пункт 6: Развёртывание (Deployment) веб-приложения на Django для управления проектами и задачами:**  
  
**6. Развёртывание (Deployment)**  
  
**6.1. Подготовка к продакшн-среде**  
  
Перед развёртыванием необходимо:  
 • Убедиться, что отключён DEBUG = False  
 • Настроить переменные окружения (через .env, python-decouple или os.environ)  
 • Использовать безопасный SECRET\_KEY  
 • Настроить ALLOWED\_HOSTS для продакшн-домена  
  
  
**6.2. Выбор платформы для хостинга**  
  
Возможные варианты:  
 • **Render**, **Railway**, **Heroku** — простое бесплатное/платное облачное хостинг-решение  
 • **VPS (DigitalOcean, Linode, Hetzner)** — более гибкий, ручная настройка сервера  
 • **Docker-контейнеры** — для переноса и масштабирования  
 • **GitHub Pages** — только для frontend (если бы был SPA)  
  
  
**6.3. Развёртывание на VPS (на примере Ubuntu)**  
  
**1. Установка зависимостей:**  
  
sudo apt update  
sudo apt install python3-pip python3-venv nginx git  
  
**2. Клонирование проекта и создание виртуального окружения:**  
  
git clone https://github.com/username/project\_manager.git  
cd project\_manager  
python3 -m venv venv  
source venv/bin/activate  
pip install -r requirements.txt  
  
**3. Миграции и статические файлы:**  
  
python manage.py migrate  
python manage.py collectstatic  
  
**4. Настройка Gunicorn (WSGI):**  
  
pip install gunicorn  
gunicorn project\_manager.wsgi:application --bind 127.0.0.1:8000  
  
**5. Настройка Nginx:**  
  
sudo nano /etc/nginx/sites-available/project\_manager  
  
Пример конфига:  
  
server {  
    listen 80;  
    server\_name your\_domain.com;  
  
    location /static/ {  
        alias /home/user/project\_manager/static/;  
    }  
  
    location / {  
        proxy\_pass http://127.0.0.1:8000;  
        include proxy\_params;  
    }  
}  
  
Затем:  
  
sudo ln -s /etc/nginx/sites-available/project\_manager /etc/nginx/sites-enabled  
sudo systemctl restart nginx  
  
  
**6.4. Настройка домена и HTTPS**  
 • Зарегистрировать домен (через Namecheap, Cloudflare и т.д.)  
 • Прописать DNS-записи  
 • Настроить HTTPS с помощью **Let’s Encrypt**:  
  
sudo apt install certbot python3-certbot-nginx  
sudo certbot --nginx -d your\_domain.com  
  
  
  
**6.5. Альтернативный способ — Docker**  
  
Создать Dockerfile, docker-compose.yml:  
  
FROM python:3.11  
WORKDIR /app  
COPY . .  
RUN pip install -r requirements.txt  
CMD ["gunicorn", "project\_manager.wsgi:application", "--bind", "0.0.0.0:8000"]  
  
  
  
  
**6.6. CI/CD**   
  
Интеграция с GitHub Actions для автоматического развёртывания:  
 • Автоматическая сборка и тесты  
 • Деплой на сервер (через SSH, Docker или Render API)13:59