**В шаге от первой вершины**

На протяжении многих месяцев вы изучали основы бэкенд-разработки: продирались через теорию, решали задачи, бились над финальными заданиями спринтов. Фундамент заложен, пришло время возвести на нём здание — написать дипломный проект.

# Что такое дипломный проект

Это большая работа, выполняя которую вы сможете применить знания, полученные за время обучения на курсе. Решение некоторых задач будет не так очевидно, поэтому понадобится и ваш навык искать информацию самостоятельно.

## Чем диплом отличается от обычных финальных проектов

Во-первых, у вас не будет строгих требований «как должен работать код», поэтому для каждого шага при решении большой задачи вы сможете выбрать лучшие на ваш взгляд практики.

Во-вторых, у вас не будет автоматических тестов. Единственное, что проверит система, — соответствие вашего кода PEP 8. Дипломный проект требует большой степени самостоятельности, но и на этом этапе мы будем рядом: если понадобится — поддержим и поможем.

## Что мы подготовили для работы над дипломным проектом

Описание проекта и его возможностей.

Готовый фронтенд в виде SPA-приложения на фреймворке React.

Спецификацию API.

Коллекцию Postman с запросами к API.

Данные для загрузки в БД.

Заготовку инфраструктуры.

### **Возможности проекта**

Вам предстоит поработать с проектом «Фудграм» — сайтом, на котором пользователи будут публиковать свои рецепты, добавлять чужие рецепты в избранное и подписываться на публикации других авторов. Зарегистрированным пользователям также будет доступен сервис «Список покупок». Он позволит создавать список продуктов, которые нужно купить для приготовления выбранных блюд.

# **Как должно работать веб-приложение**

Проект состоит из следующих страниц:

главная, страница входа, страница регистрации, страница рецепта, страница пользователя, страница подписок, избранное, список покупок, создание и редактирование рецепта, страница смены пароля, статические страницы «О проекте» и «Технологии».

# **Главная**

Содержимое главной — список первых шести рецептов, отсортированных по дате публикации «от новых к старым». На этой странице подразумевается постраничная пагинация. Остальные рецепты должны быть доступны на следующих страницах.

# **Страница регистрации**

В проекте доступна система регистрации и аутентификации пользователей.

Обязательные поля для пользователя при регистрации:

имя, фамилия, имя пользователя (никнейм), адрес электронной почты, пароль.

# **Страница входа**

После регистрации пользователь переадресовывается на страницу входа.

# **Статические страницы**

Проект предусматривает как минимум две статические страницы: «О проекте» и «Технологии». По умолчанию эти страницы отключены, и по их адресу отображается страница с ошибкой 404.

# **Страница рецепта**

Здесь — полное описание рецепта. Залогиненные пользователи могут добавить рецепт в избранное и список покупок, а также подписаться на автора рецепта.   
Для каждого рецепта можно получить прямую короткую ссылку, нажав на соответствующую иконку справа от названия рецепта.  
Эта ссылка не меняется после редактирования рецепта.  
В большинстве браузеров, например на основе Chromium, ссылка автоматически скопируется в буфер обмена, а в Safari — выведется на экран в правом верхнем углу, откуда её можно скопировать в буфер обмена. Такое поведение реализовано в спаприложении.

# **Страница пользователя**

На странице — имя пользователя, все рецепты, опубликованные пользователем, и кнопка, чтобы подписаться или отписаться от него.

# **Страница подписок**

Только владелец аккаунта может просмотреть свою страницу подписок. Ссылка на неё находится в выпадающем меню в правом верхнем углу.

Подписаться на публикации могут только залогиненные пользователи.

Сценарий поведения пользователя:

1. Пользователь переходит на страницу другого пользователя или на страницу рецепта и подписывается на публикации автора кликом по кнопке **Подписаться на автора**.
2. Пользователь переходит на страницу **Мои подписки** и просматривает список рецептов, опубликованных теми авторами, на которых он подписался. Записи сортируются по дате публикации — от новых к старым.
3. При необходимости пользователь может отказаться от подписки на автора. Тогда ему нужно перейти на страницу автора, его рецепта или на страницу своих подписок и нажать кнопку **Отписаться**.

# **Избранное**

Добавлять рецепты в избранное может только залогиненный пользователь.  
Если незалогиненный пользователь попробует добавить рецепт в избранное, подписки или покупки, то приложение попросит его зарегистрироваться или войти в свой аккаунт.  
Добавить рецепт в избранное можно с главной страницы и со страницы самого рецепта, нажав на соответствующую иконку.  
Сам список избранного может просмотреть только его владелец.

Сценарий поведения пользователя:

1. Пользователь отмечает один или несколько рецептов кликом по кнопке **Добавить в избранное**.
2. Пользователь переходит на страницу **Избранное** и просматривает свой список избранных рецептов.
3. При необходимости пользователь может удалить рецепт из избранного.

# **Список покупок**

Работать со списком покупок могут только залогиненные пользователи. Доступ к собственному списку покупок есть только у владельца аккаунта.

Сценарий поведения пользователя:

1. Пользователь отмечает один или несколько рецептов кликом по кнопке **Добавить в покупки**. Сделать это можно с главной страницы или со страницы рецепта.
2. Пользователь переходит на страницу **Список покупок**, там доступны все добавленные в список рецепты. Пользователь нажимает кнопку **Скачать список** и получает файл с перечнем и количеством необходимых ингредиентов для всех рецептов, сохранённых на странице **Список покупок**.
3. При необходимости пользователь может удалить рецепт из списка покупок.

Пользователь может скачать свой список покупок в формате .txt, .pdf или любом другом, который разработчик выбрал при реализации программы.

Ингредиенты в скачанном списке не дублируются. Если в двух рецептах есть сахар (в одном рецепте 5 г, в другом — 10 г), в списке должен быть один пункт: *Сахар — 15 г.* То есть ингредиенты в списке покупок суммируются.

Сам список выглядит примерно так:

Фарш (баранина и говядина) (г) — 600

Сыр плавленый (г) — 200

Лук репчатый (г) — 50 Картофель (г) — 1000

Молоко (мл) — 250

Яйцо куриное (шт.) — 5

Соевый соус (ст. л.) — 8

Сахар (г) — 230

Растительное масло рафинированное (ст. л.) — 2

Соль (по вкусу) — 4

Перец чёрный (щепотка) — 3

Дизайн — на усмотрение разработчика.

# **Создание и редактирование рецепта**

Эта страница доступна только для залогиненных пользователей. Все поля на ней обязательны для заполнения.

Сценарий поведения пользователя:

1. Пользователь переходит на страницу добавления рецепта, нажав на кнопку **Создать рецепт** в шапке сайта.
2. Пользователь заполняет все обязательные поля.
3. Пользователь нажимает кнопку **Создать рецепт**.

Также пользователь может отредактировать любой рецепт, который он создал.

# **Страница изменения пароля**

Доступ к этой странице есть только у залогиненных пользователей, через кнопку **Сменить пароль** в выпадающем меню в правом верхнем углу страницы.

Страница изменения пароля выглядит следующим образом:

# **Смена аватара**

После регистрации новый пользователь получает изображение профиля по умолчанию.

При клике на аватар у пользователя в шапке сайта появляется возможность поменять его или удалить.

# **Разграничение прав**

В проекте описаны разные уровни доступа пользователей:

гость (анонимный пользователь), аутентифицированный (залогиненный) пользователь, администратор.

**Стартовый набор и задание**

У будущего веб-приложения уже есть готовый фронтенд — это одностраничное SPA-приложение, написанное на фреймворке React. Файлы, необходимые для его сборки, хранятся в репозитории foodgram, в папке frontend. Этот репозиторий вы сможете склонировать себе на компьютер после того, как прочтёте уроки этой темы.

Кроме папки frontend, в репозитории также есть папки backend, infra, data и docs:

В папке infra — заготовка инфраструктуры проекта: конфигурационный файл nginx и docker-compose.yml.

В папке backend лежит только скрытый файл .gitkeep. Он нужен для того, чтобы папка отображалась в Git. Эта папка предназначена для бэкенда сервиса, который вы разработаете с нуля.

В папке data подготовлен список ингредиентов с единицами измерения — это часть данных для БД, с которой вам предстоит работать. Список сохранён в форматах JSON и CSV.

В папке docs — файлы для отображения спецификации API.

# **Задание**

Ваша задача как бэкенд-разработчика — написать бэкенд в виде REST API для веб-приложения «Фудграм» и проверить, что проект работает в соответствии с требованиями.

Чтобы «подружить» ваш бэкенд с подготовленным SPA, REST API необходимо реализовать строго в соответствии со спецификацией.

Проект должен быть полностью рабочим и соответствовать описанию из предыдущего урока.

У вас нет жёстких рамок по структуре и содержанию кода, однако есть технические условия общего плана, которые должны быть соблюдены.

# **Базовые модели проекта**

У проекта «Фудграм» должно быть две базовые модели: «Рецепт» и «Ингредиент». Скорее всего, кроме этих моделей, вам потребуется создать и другие. Какими они будут — вы сможете решить на этапе проектирования архитектуры веб-приложения. **Рецепт**

Атрибуты модели:

Автор публикации (пользователь).

Название.

Картинка.

Текстовое описание.

Ингредиенты — продукты для приготовления блюда по рецепту. Множественное поле с выбором из предустановленного списка и с указанием количества и единицы измерения.

Время приготовления в минутах.

Все поля обязательны для заполнения.

**Ингредиент**

Атрибуты модели:

Название.

Единица измерения.

Все поля обязательны для заполнения.

Более подробные требования к полям моделей можно найти в спецификации к API.

# **Как должна быть настроена админка**

В интерфейс админ-зоны нужно вывести необходимые поля моделей и настроить поиск:

вывести все модели с возможностью редактирования и удаления записей; для модели пользователей добавить поиск по адресу электронной почты и имени пользователя; для модели рецептов:

в списке рецептов вывести название и имя автора рецепта; добавить поиск по автору, названию рецепта; на странице рецепта вывести общее число добавлений этого рецепта в избранное.

для модели ингредиентов:

в список вывести название ингредиента и единицы измерения; добавить поиск по названию.

# **Каким требованиям должна соответствовать инфраструктура проекта**

Проект должен использовать базу данных PostgreSQL. Вы можете разрабатывать проект на SQLite, а PostgreSQL подключить позже, при подготовке к деплою.

В Django-проекте должен быть файл requirements.txt со всеми зависимостями.   
Проект нужно запустить в четырёх контейнерах — nginx, PostgreSQL, backend и frontend через docker-compose. Контейнер frontend должен использоваться лишь для подготовки файлов.

Данные должны сохраняться в volumes.

Образы проекта должны быть запушены на Docker Hub.

Должен быть настроен CI/CD через GitHub Actions.

# **Что ещё должно быть в проекте**

Веб-приложение должно быть наполнено тестовыми данными: нужно создать несколько пользователей с разными уровнями доступа и добавить хотя бы по одному рецепту от имени каждого пользователя.

# Запустите заготовку проекта

Когда вам станет доступен репозиторий foodgram, склонируйте его себе на компьютер.

Затем, находясь в папке infra, выполните в терминале команду docker compose up.

При выполнении этой команды контейнер frontend, описанный в docker-compose.yml, подготовит файлы, необходимые для фронтенд-приложения, а затем прекратит свою работу.

По адресу [http://localhost](http://localhost/) изучите фронтенд веб-приложения, а по адресу <http://localhost/api/docs/> — спецификацию API.

# Спроектируйте модели и связи между ними

Из прошлого урока вам точно известно, что в проекте должно быть минимум две модели: «Рецепт» и «Ингредиент». Но какие ещё модели нужно создать, чтобы проект полноценно работал? Где будут нужны множественные связи, а где может потребоваться каскадное удаление?

На этапе проектирования вы должны дать ответы на эти вопросы, решить, где понадобится индексирование, и разобраться с другими задачами, которые касаются архитектуры.

Набросайте схемы моделей и связей. Скомкайте, сожгите и набросайте снова.

# Спроектируйте Django-приложения в составе проекта

Прочитайте задание, выделите сходные задачи, объедините их в Django-приложения (для начала в голове или на листе бумаги). Поразмышляйте: можно ли приложение скопировать в другой проект, заработает ли оно там в том виде, в котором вы его запланировали. Получилось? Значит, идея верна.

# Создайте основу проекта

1. В склонированном репозитории foodgram разверните виртуальное окружение, создайте новый Django-проект.
2. Разверните и подключите базу данных — на этапе разработки подойдёт SQLite.
3. Создайте суперпользователя и нескольких обычных пользователей, они пригодятся для проверки работы проекта.
4. Создайте файл зависимостей requirements.txt, актуализируйте его каждый раз, когда устанавливаете новые пакеты.

# Создайте Django-приложения и модели

Слона нужно есть по частям, поэтому не хватайтесь за всё подряд, а действуйте последовательно: создайте первое Djangoприложение, его модели и связи. После этого займитесь следующим Django-приложением. Чёткая очерёдность позволит вам своевременно скорректировать план, если в нём что-то пойдёт не так, ведь важно не только создать Django-приложения, но и понимать, в какой последовательности их стоит реализовывать.

После описания моделей загрузите данные из списка ингредиентов.

# Реализуйте логику работы веб-приложения

Именно в этот момент всё пойдёт не по плану. Это нормально. Но если все предыдущие этапы выполнены, работать будет несколько легче, чем если бы этой подготовки не было.

Самое время запустить в консоли интерактивный режим Python, выполнить import this и немного помедитировать: круг замкнулся.

И после этого — продолжать писать код.

# Выпустите проект в продакшен

Когда ваше веб-приложение будет корректно работать локально, выпускайте его в продакшен.

Этот пункт плана можно разделить на следующие этапы:

1. Проверьте актуальность файла requirements.txt.
2. Если вы разрабатывали проект на SQLite — замените базу на PostgreSQL.
3. Напишите докерфайл для образа с Django-проектом и Gunicorn. В конце сборки контейнера должны быть команды для выполнения миграций и сборки статики.
4. Напишите docker-compose.yml, в нём опишите запуск контейнеров из официальных образов PostgreSQL и nginx и своего контейнера с проектом. Опишите конфигурацию контейнеров.
5. Запушьте контейнер с проектом на Docker Hub.
6. Настройте автоматическую проверку кода при помощи GitHub Actions.
7. Добавьте в файл README.md инструкцию для ревьюера, как запускать проект.

План есть, а чтобы вам было немного проще реализовывать его пункты, загляните в следующий урок — там вас ждёт несколько лайфхаков и подсказок, которые помогут вам при выполнении большого задания.

**Подсказки и лайфхаки**

# **Аутентификация и регистрация пользователей**

При переносе стандартного процесса авторизации Django в формат REST API вам понадобится библиотека djoser. При подключении придётся переопределить некоторые настройки и написать кастомный вьюсет.

Авторизация должна быть реализована с помощью токенов: пользователь регистрируется по электронной почте и паролю, отдельным запросом получает токен, затем этот токен передаётся в заголовке каждого запроса.

# **Сериализаторы**

При публикации рецепта фронтенд кодирует картинку в строку base64; на бэкенде её необходимо декодировать и сохранить как файл. Для этого будет удобно создать кастомный тип поля для картинки, переопределив метод сериализатора to\_internal\_value.

Для сохранения ингредиентов рецепта потребуется переопределить методы createи update в ModelSerializer.

Используйте подходящие типы related-полей; для некоторых данных вам потребуется использовать SerializerMethodField.

# **Вьюсеты**

Если вы решите использовать вьюсеты, то вам потребуется добавлять дополнительные action. Не забывайте о том, что для разных action сериализаторы и уровни доступа (permissions) могут отличаться.

Некоторые методы, в том числе и action, могут быть похожи друг на друга. Избегайте дублирующегося кода.

**Поиск по ингредиентам**

Ищите ингредиенты по полю name регистронезависимо, по вхождению в начало названия.

# **Инфраструктура**

Настройте nginx так, чтобы запросы к адресам /api/ переадресовывались на бэкенд. Подключите админ-панель Django, настройте раздачу медиа и статики админки.

Учтите, что если ни один из предыдущих типов location не сработал, Nginx выберет блок location с наиболее длинным совпадающим префиксом запроса. Если есть несколько блоков с префиксами одинаковой длины, будет использован первый найденный в конфигурации.

# **Пагинация**

Для реализации пагинации воспользуйтесь стандартным пагинатором DRF. Вам нужно будет лишь переопределить название поля, отвечающего за количество результатов в выдаче.

# **Список покупок файлом**

Вам понадобится отдать пользователю файл со списком ингредиентов. Ровно такую же задачу вы не решали по ходу курса, но вы работали с view-функциями, а они умеют возвращать через HttpResponse не только текст, но и файлы.

# **Спецификация API**

Спецификация — ваш лучший друг. В ней описаны все поля запросов и ответов, а также дополнительные GET-параметры.

API должен быть реализован строго по спецификации.

В директории postman\_collection сохранена коллекция запросов для отладки и проверки работы проекта.

Когда проект будет готов обрабатывать запросы к API, импортируйте коллекцию в Postman и используйте её для отладки работы приложения.