****

**Факультет информатики и систем управления**

**Образовательная программа бакалавриата „информатика“**

**(концентрация – программная инженерия)**

**Разработка интеллектуальной системы персонализированных рекомендаций рецептов на основе графов знаний и нейросетевых моделей**

**Максим Морозов**

**108159**

**Руководитель: проф. Нарешелашвили Гулбаат**

**Тбилиси – 2025**

**Студент**

**Морозов Максим**

**Подпись студента:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Руководитель:**

**Проф. Гулбаат Нарешелашвили**

**Подпись руководителя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Содержание

[Введение 4](#__RefHeading___Toc2152_3841837669)

[Глава 1. Теоретические основы: Трансформеры и архитектура BAMnet 7](#__RefHeading___Toc2154_3841837669)

[Глава 1.1 История и мотивация создания 7](#__RefHeading___Toc2156_3841837669)

[Глава 1.2 Основные принципы работы трансформеров 9](#__RefHeading___Toc2158_3841837669)

[Глава 1.3 KBQA. Ответы на вопросы на графах знаний 12](#__RefHeading___Toc2160_3841837669)

[Глава 1.4 Архитектура BAMnet 13](#__RefHeading___Toc2162_3841837669)

[Глава 2. Подготовка данных и обучение языковой модели 16](#__RefHeading___Toc2164_3841837669)

[Глава 2.1 Создание обучающего датасета 16](#__RefHeading___Toc2166_3841837669)

[Глава 2.2 Архитектура модели BAMnet для задачи рекомендации 17](#__RefHeading___Toc2168_3841837669)

[Глава 2.3 Функция потерь и оптимизация 18](#__RefHeading___Toc2170_3841837669)

[Глава 2.4 Гиперпараметры и процесс обучения 19](#__RefHeading___Toc2172_3841837669)

[Глава 3. Реализация проекта 21](#__RefHeading___Toc2174_3841837669)

[Глава 3.1 Общая структура системы 21](#__RefHeading___Toc2176_3841837669)

[Глава 3.2 Реализация серверной части 25](#__RefHeading___Toc2178_3841837669)

[Глава 3.3 Реализация клиентской части (Frontend) 28](#__RefHeading___Toc2180_3841837669)

[Глава 4. Анализ и результаты 31](#__RefHeading___Toc2182_3841837669)

[Глава 4.1 Качественный анализ работы web-приложения на сценариях 32](#__RefHeading___Toc2184_3841837669)

[Глава 4.2 Анализ производительности и ограничений 35](#__RefHeading___Toc2186_3841837669)

[Глава 4.3 Возможности для улучшения и дальнейшего развития 37](#__RefHeading___Toc2188_3841837669)

[Заключение 41](#__RefHeading___Toc2190_3841837669)

[Использованная литература 43](#__RefHeading___Toc2192_3841837669)

[Приложения 44](#__RefHeading___Toc2194_3841837669)

# Введение

В современном мире, где информация о здоровом питании доступна как никогда ранее, парадоксальным образом возрастает сложность практического применения этих знаний в повседневной жизни. Люди сталкиваются с огромным потоком данных о диетах, нутриентах и влиянии продуктов на здоровье, но испытывают трудности с интеграцией этой информации в свой ежедневный рацион. Существующие онлайн-сервисы и приложения для поиска рецептов, несмотря на их популярность, зачастую предлагают универсальные решения, которые не учитывают уникальные потребности каждого конкретного пользователя. Эта проблема становится особенно острой для людей с хроническими заболеваниями, такими как диабет, пищевые аллергии или метаболические нарушения, для которых правильный подбор диеты является не просто вопросом хорошего самочувствия, а критически важным элементом управления здоровьем.

Основной целью данной работы является разработка и реализация прототипа полнофункциональной веб-системы для персонализированной рекомендации рецептов, способной обрабатывать запросы на естественном языке и учитывать сложный набор индивидуальных ограничений пользователя.

Актуальность данной работы обусловлена несколькими ключевыми факторами:

* Растущий спрос на персонализацию: Современный потребитель ожидает от цифровых сервисов индивидуального подхода. В контексте питания это означает получение рекомендаций, которые учитывают не только вкусовые предпочтения, но и личные цели (похудение, набор мышечной массы), медицинские ограничения (аллергии, непереносимость) и диетические предписания (веганство, низкоуглеводная диета).
* Недостатки существующих решений: Большинство популярных платформ для поиска рецептов работают как поисковые системы по ключевым словам или используют простые фильтры. Они не способны обрабатывать сложные запросы на естественном языке, такие как «найди рецепт ужина с высоким содержанием белка, но без молочных продуктов и орехов». Пользователю приходится вручную просматривать десятки неподходящих вариантов, что отнимает время и снижает мотивацию.
* Появление больших графов знаний о еде: в последние годы были созданы и стали доступны масштабные структурированные базы данных, такие как FoodKG, которые объединяют информацию о миллионах рецептов, ингредиентах, их химическом составе и нутриентной ценности. Эти ресурсы предоставляют уникальную возможность для создания интеллектуальных систем нового поколения, способных производить глубокий семантический анализ и находить неочевидные связи между продуктами и рецептами.

Таким образом, существует явный разрыв между потребностью общества в персонализированном подходе к питанию и технологическими возможностями существующих сервисов. Наш проект направлен на преодоление этого разрыва путем создания интеллектуальной системы, которая ставит во главу угла индивидуальные потребности пользователя.

**Задачи проекта**

Для достижения поставленной цели были определены следующие задачи:

1. Проанализировать существующие подходы к рекомендательным системам и технологиям ответов на вопросы (Question Answering) для выявления их сильных и слабых сторон в контексте кулинарной предметной области.
2. Спроектировать архитектуру комплексной системы, включающей клиентскую часть (frontend), серверную часть (backend) и модуль машинного обучения, обеспечив их эффективное взаимодействие.
3. Реализовать серверное приложение на фреймворке FastAPI, отвечающее за управление пользовательскими данными (регистрация, профили), аутентификацию и обработку API-запросов.
4. Разработать и интегрировать NLP-модуль для семантического анализа запросов пользователя на естественном языке и извлечения из них ключевых сущностей: тем, предпочтений и ограничений.
5. Адаптировать и интегрировать нейросетевую модель на основе архитектуры BAMnet, способную находить релевантные рецепты в графе знаний FoodKG с учетом сформированных персональных ограничений.
6. Создать клиентское веб-приложение на фреймворке React, предоставляющее пользователю удобный интерфейс для регистрации, управления своим профилем (списком запрещенных ингредиентов) и взаимодействия с рекомендательной системой.
7. Провести качественный анализ работы готовой системы на практических примерах, оценить ее функциональность и определить направления для дальнейшего развития.

# Глава 1. **Теоретические основы: Трансформеры и архитектура BAMnet**

Для понимания принципов, лежащих в основе интеллектуального ядра нашего проекта, необходимо рассмотреть эволюцию нейросетевых архитектур для обработки текста и их применение в задачах семантического поиска. В данном разделе мы проследим путь от появления механизма внимания, который произвел революцию в NLP, до его реализации в современных моделях для ответов на вопросы на графах знаний (KBQA), таких как BAMnet.

**Глава 1.1 История и мотивация создания**

Долгое время доминирующей парадигмой для работы с последовательными данными, такими как текст, были рекуррентные нейронные сети (RNN) и их более совершенная версия — Long Short-Term Memory (LSTM). Эти архитектуры обрабатывали последовательность элемент за элементом, сохраняя информацию о предыдущих шагах во внутреннем «состоянии». Такой подход был интуитивно понятен, но обладал фундаментальными ограничениями:

* Из-за многократного применения матричных операций в рекуррентной цепочке градиенты ошибки при обучении либо затухали, либо «взрывались», что мешало сетям улавливать связи между словами, находящимися далеко друг от друга в тексте.
* Рекуррентная природа RNN и LSTM не позволяла эффективно распараллеливать вычисления, так как для обработки n-го слова нужно было дождаться результата обработки (n-1)-го. Это существенно замедляло обучение на больших объемах данных.
* Модели типа «sequence-to-sequence», построенные на RNN/LSTM, сжимали всю входную последовательность в один вектор фиксированной длины (вектор контекста), из которого затем декодер генерировал выходную последовательность. Это создавало сложности для сохранения семантической сложности длинных предложений.

Именно для решения последней проблемы в 2014 году был предложен механизм внимания (Attention Mechanism). Его ключевая идея заключалась в том, чтобы позволить модели на каждом шаге генерации ответа динамически «обращать внимание» на разные части входной последовательности, взвешивая их по степени релевантности. Это не только улучшило качество машинного перевода, но и заложило основу для следующего архитектурного прорыва.

В 2017 году в статье «*Attention Is All You Need*» была представлена архитектура Трансформер, которая полностью отказалась от рекуррентных слоев. Авторы показали, что одного лишь механизма внимания, примененного правильным образом, достаточно для достижения нового уровня качества в NLP-задачах. Это открытие стало поворотной точкой и послужило мотивацией для создания целого поколения моделей, включая ту, что легла в основу нашего проекта.

## **Глава 1.2 Основные принципы работы трансформеров**

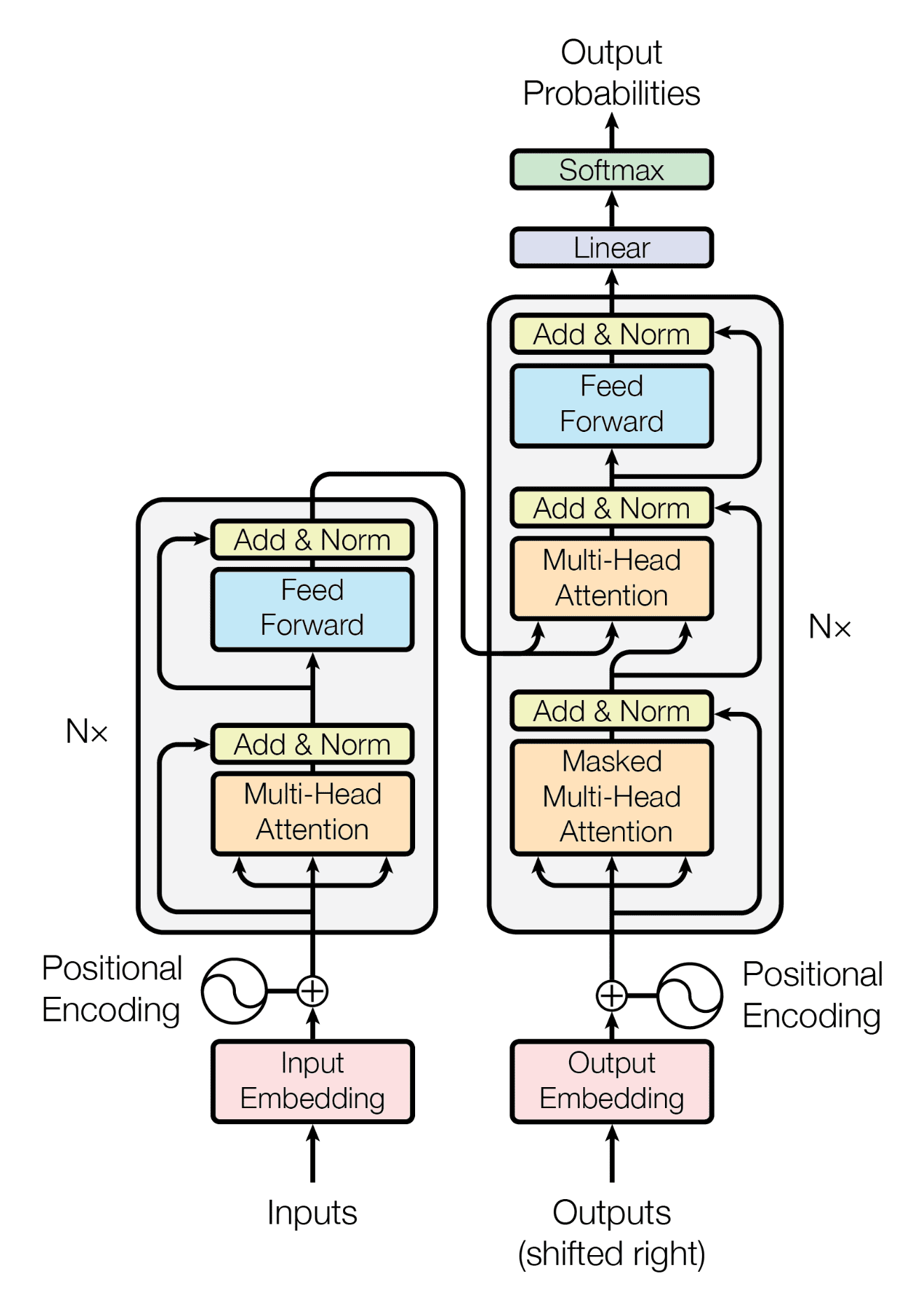
Архитектура Трансформер стала революционной благодаря своему полному отказу от рекуррентных слоев в пользу более мощного и параллелизуемого механизма — **внимания (attention)**. Это позволило моделям обрабатывать текст не последовательно, а целиком, улавливая сложные зависимости между словами независимо от их расстояния друг от друга. Работа архитектуры основана на нескольких взаимосвязанных компонентах.

Figure 1- архитектура трансформкрв

В своей основе механизм внимания имитирует когнитивный процесс человека. Когда мы слышим или читаем предложение, мы не придаем каждому слову одинаковое значение. Наше внимание интуитивно фокусируется на ключевых словах, которые определяют смысл высказывания. Например, в предложении «Посоветуй вегетарианский суп без картошки пожалуйста» слова «вегетарианский», «суп» и «без картошки» являются более значимыми для понимания сути запроса, чем служебные слова «посоветуй» или «пожалуйста».

Механизм внимания в нейронных сетях реализует именно эту идею. Он позволяет модели динамически взвешивать важность каждого элемента входной последовательности (например, каждого слова в предложении) при обработке и генерации результата. Механизм внимания сохраняет доступ ко всем входным элементам и на каждом шаге вычислений решает, на какие из них «посмотреть» пристальнее.

Наиболее распространенная форма механизма внимания, используемая в Трансформерах, — это *Scaled Dot-Product Attention*. Ее работа описывается через три ключевых вектора, которые модель обучается генерировать для каждого входного слова:

1. Query (Запрос) - вектор, представляющий *текущее* слово, для которого мы хотим найти контекст. Он как бы «задает вопрос»: "Какие слова в этом предложении важны для меня?".
2. Key (Ключ) - вектор, который описывает, *что* может предложить каждое слово в предложении. Он выступает в роли «метки» или «ярлыка».
3. Value (Значение) - вектор, содержащий *саму информацию* о слове, его семантическую суть.

Процесс вычисления внимания можно разбить на несколько шагов:

1. **Вычисление оценок схожести** (Attention Scores). Для каждого слова (представленного вектором Query) вычисляется его схожесть со всеми остальными словами в предложении (представленными векторами Key). Чаще всего для этого используется скалярное произведение (dot product).
2. **Масштабирование** (Scaling). Полученные оценки делятся на квадратный корень из размерности векторов (). Это делается для стабилизации градиентов во время обучения.
3. **Нормализация** (Softmax). К масштабированным оценкам применяется функция softmax. Она преобразует оценки в вероятностное распределение: набор положительных чисел, сумма которых равна 1. Эти числа и есть веса внимания (attention weights). Они показывают, какой процент «внимания» текущее слово должно уделить каждому другому слову в предложении.
4. **Формирование итогового вектора**. Итоговый вектор для текущего слова вычисляется как взвешенная сумма Value-векторов всех слов в предложении, где весами выступают полученные на предыдущем шаге веса внимания.

Таким образом, на выходе мы получаем новое, обогащенное контекстом представление для каждого слова, которое несет в себе информацию о наиболее релевантных для него других словах.

**Multi-Head Self-Attention —** это сердце Трансформера. В отличие от классического механизма внимания, который связывал энкодер и декодер, self-attention позволяет словам внутри *одного и того же* предложения взаимодействовать друг с другом. Для каждого слова в предложении модель вычисляет три вектора: Query (Запрос), Key (Ключ) и Value (Значение). Схожесть между Query-вектором одного слова и Key-векторами всех остальных слов в предложении определяет, насколько сильно это слово должно «обратить внимание» на другие. Результатом является новый вектор для каждого слова, представляющий собой взвешенную сумму Value-векторов всех слов, где веса — это оценки внимания.

«Многоголовочность» этого механизма означает, что процесс вычисления внимания выполняется параллельно в нескольких «головах». Каждая голова обучается фокусироваться на разных типах зависимостей — одни на синтаксических (например, связь подлежащего и сказуемого), другие на семантических (например, синонимичные отношения). Результаты всех голов затем объединяются, что позволяет получить богатое, многоаспектное представление каждого слова в контексте всего предложения.

Другие ключевые компоненты:

* *Positional Encoding* (Позиционное кодирование): так как self-attention обрабатывает все слова одновременно, информация об их порядке теряется. Для решения этой проблемы к входным векторам слов добавляются специальные позиционные векторы, которые кодируют абсолютную или относительную позицию слова в предложении.
* *Feed-Forward Networks*: после слоя внимания представления слов проходят через небольшую полносвязную нейронную сеть для дополнительного нелинейного преобразования.
* *Residual Connections* и *Layer Normalization*: каждый блок Трансформера использует остаточные соединения (как в ResNet) для облегчения прохождения градиента через глубокие слои и нормализацию по слоям для стабилизации обучения.

**Глава 1.3 KBQA. Ответы на вопросы на графах знаний**

Задача **KBQA (Knowledge Base Question Answering)** — это поиск точных ответов на вопросы в больших структурированных базах знаний (графах). Это идеальная область для применения мощных языковых моделей, так как она требует глубокого понимания как семантики вопроса, так и структуры графа.

Подходы к KBQA можно условно разделить на два типа:

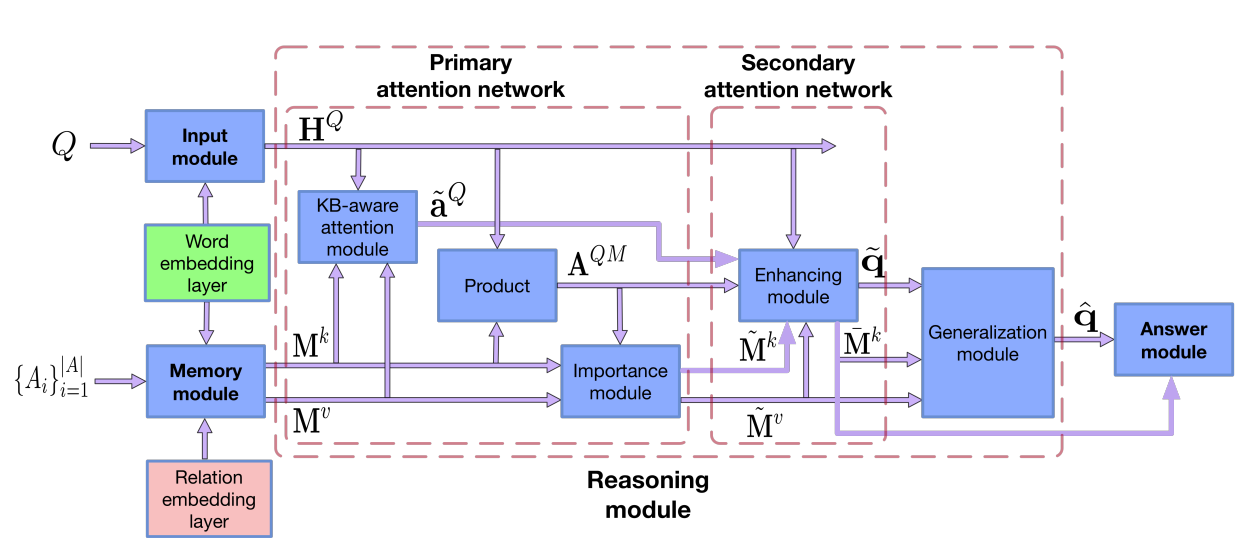
1. **На основе семантического парсинга (SP-based).** Методы этого класса пытаются перевести вопрос на естественном языке в формальный запрос (например, на языке SPARQL). Они отличаются высокой точностью, но их создание крайне трудоемко, и они плохо обобщаются на новые, не виденные ранее формулировки.
2. **На основе информационного поиска (IR-based).** Эти методы используют векторные представления (embeddings) для кодирования вопросов и сущностей/отношений из графа в едином многомерном пространстве. Задача сводится к поиску ближайших в этом пространстве соседей: модель ищет те сущности из графа, чьи векторы наиболее близки к вектору вопроса. IR-подход гораздо более гибок, устойчив к разнообразным формулировкам и легко масштабируется на большие графы знаний, что делает его идеальным для нашего проекта.

**Глава 1.4 Архитектура BAMnet**

Модель Bidirectional Attentive Memory Networks (BAMnet) представляет собой усовершенствованный IR-based подход к KBQA. Ее центральная идея заключается в необходимости явного моделирования двунаправленного взаимодействия между вопросом и графом знаний.

BAMnet утверждает, что для точного ответа недостаточно просто закодировать вопрос и сущности из графа независимо друг от друга. Вместо этого, контекст из графа должен помогать модели лучше понять вопрос, а вопрос, в свою очередь, должен подсвечивать наиболее релевантные части графа.

Эта идея реализуется через иерархическую, двухуровневую систему внимания.

Figure 2 - Архитектура BAMnet

Ключевые модули BAMnet:

1. *Input Module* (Входной модуль): стандартный блок, который кодирует входной вопрос, получая контекстуализированные векторы для каждого слова.
2. *Memory Module* (Модуль памяти): отвечает за структурированное представление кандидатов на ответ. Вместо того чтобы представлять кандидата одним вектором, BAMnet раскладывает информацию о нем на три ключевых аспекта, которые хранятся в памяти типа Key-Value:
   * Answer Type: тип сущности (например, *рецепт*, *ингредиент*).
   * Answer Path: путь в графе от темы вопроса до кандидата.
   * Answer Context: другие связанные сущности, которые могут помочь при ответе на вопросы с дополнительными ограничениями.
3. *Reasoning Module* (Модуль рассуждений): ядро модели, где и происходит двунаправленное внимание.
   * Первичная сеть внимания:
     + KB-aware attention: оценивает важность каждого слова в вопросе в свете доступных аспектов ответа из KB. Например, слово «без» в вопросе будет иметь сильную связь с аспектом «контекст», где перечислены ингредиенты.
     + Importance module: выполняет обратную задачу — оценивает важность каждого аспекта ответа (типа, пути, контекста) для данного конкретного вопроса.
   * Вторичная сеть внимания (Enhancing module): использует результаты первичного внимания для взаимного обогащения представлений. Вектор вопроса уточняется с учетом самой релевантной информации из графа, и наоборот.
4. *Answer Module* (Модуль ответа): в этом модуле на последнем шаге вычисляется итоговая оценка схожести между финальным, обогащенным вектором вопроса и векторами кандидатов-ответов, после чего происходит их ранжирование.

# Глава 2. Подготовка данных и обучение языковой модели

**Глава 2.1 Создание обучающего датасета**

Для обучения нейросетевой модели был сгенерирован специализированный датасет, имитирующий реальные сценарии взаимодействия пользователя с системой. Процесс генерации, взятый из статьи PFoodReq, был направлен на создание разнообразных и реалистичных обучающих примеров.

Генерация вопросов на основе шаблонов:

Процесс генерации является обратным по отношению к работе системы: он начинается со структурированных данных из графа и заканчивается вопросом на естественном языке.

* Анализ и создание шаблонов: был создан набор параметризованных шаблонов, которые покрывают различные типы запросов:
  + Простые запросы с позитивными ограничениями: *"Suggest {tag} dishes that contain {in\_list}?"*
  + Запросы с негативными ограничениями: *"What {tag} dishes can I make without {in\_list}?"*
  + Комплексные запросы с разными видами ограничений: *"Suggest {tag} dishes that contain {in\_list} and without {in\_list}?"*
  + Запросы, включающие нутриентные требования: *"Recommend {limit} {nutrient} {tag} recipes..."*
* Генерация обучающего примера:  
  a. **Семплирование из графа.** Из графа FoodKG случайным образом выбирается стартовая сущность-топик (например, тег mexican). Вокруг нее строится локальный подграф (h-hop neighborhood), из которого извлекаются связанные сущности (ингредиенты, другие теги).  
  b. **Заполнение шаблона.** Выбирается случайный шаблон, и его параметры ({tag}, {in\_list} и т. д.) заполняются сущностями, извлеченными на предыдущем шаге. Так формируется поле qText.  
  c. **Формирование person.** Для каждого сгенерированного вопроса создается имитация профиля пользователя. В person случайным образом добавляются списки продуктов, которые пользователь не употребляет, что позволяет обучить модель реагировать на персонализированные ограничения.  
  e. **Определение правильного ответа.** На основе всех сгенерированных ограничений (из qText, persona) в графе знаний находится рецепт, который им всем удовлетворяет. Этот рецепт становится ground-truth ответом для данного обучающего примера.

В итоговой итоговой структуре данных каждый элемент в результирующем датасете представляет собой сложный JSON-объект, содержащий всю необходимую информацию для обучения и тестирования:

* qId: уникальный идентификатор.
* qOriginText: простой, базовый вопрос.
* qText: расширенный, сложный вопрос, сгенерированный по шаблону.
* entities: список сущностей-топиков.
* persona: объект с персональными предпочтениями.
* guideline: объект с нутриентными ограничениями.
* answers: список правильных ответов (рецептов).

Этот датасет, разделенный на обучающую, валидационную и тестовую выборки, служит основой для последующего обучения и оценки качества нейросетевой модели.

**Глава 2.2 Архитектура модели BAMnet для задачи рекомендации**

В основе нашего решения лежит адаптированная архитектура BAMnet. Для нашей задачи ее компоненты интерпретируются следующим образом:

* **Входной вопрос:** расширенный qText, включающий все ограничения.
* **Память:** состоит из рецептов-кандидатов, извлеченных из графа знаний. Каждый рецепт представлен через свои аспекты: тип кухни, набор ингредиентов, пищевая ценность.
* **Модуль рассуждений:** с помощью двунаправленных механизмов внимания модель одновременно определяет, какие слова в вопросе наиболее важны (например, "без", "низкоуглеводный"), и какие аспекты рецепта (например, список ингредиентов) наиболее релевантны для проверки этих ограничений.

**Глава 2.3 Функция потерь и оптимизация**

Для обучения модели ранжированию ответов была выбрана **триплетная функция потерь (Triplet Loss)**. Этот метод хорошо зарекомендовал себя в задачах, где необходимо не классифицировать объекты, а научиться измерять степень их сходства.

Принцип работы Triplet Loss: на каждой итерации обучения модель получает на вход триплет данных:

1. **Anchor (q):** векторное представление текущего вопроса.
2. **Positive (a+):** векторное представление правильного ответа (рецепта, удовлетворяющего всем условиям).
3. **Negative (a-):** векторное представление случайного, заведомо неправильного ответа.

Задача модели — настроить свои веса таким образом, чтобы в конечном векторном пространстве "правильный" ответ a+ был ближе к вопросу q, чем "неправильный" ответ a-. Формально это выражается через функцию потерь, которая штрафует модель, если это условие не выполняется:

Здесь - функция, измеряющая схожесть вопроса и ответа (в нашем случае, скалярное произведение их векторов), а margin — это гиперпараметр, определяющий минимально допустимую разницу в оценках схожести. Модель обучается до тех пор, пока оценка для позитивного примера не станет выше оценки для негативного как минимум на величину margin.

Ключевым аспектом успешного обучения с Triplet Loss является правильный подбор негативных примеров. Если всегда выбирать очень далекие, очевидно неправильные ответы, модель быстро научится их отличать и перестанет развиваться. Поэтому используется стратегия, при которой в качестве негативных примеров выбираются "трудные" кандидаты — рецепты, которые похожи на правильный ответ, но не удовлетворяют одному или нескольким ключевым ограничениям (например, содержат запрещенный ингредиент).

**Глава 2.4 Гиперпараметры и процесс обучения**

Процесс обучения контролировался набором гиперпараметров, основанных на рекомендациях из статьи BAMnet и экспериментально подобранных на валидационной выборке:

* Оптимизатор: Adam, как надежный и широко используемый алгоритм градиентной оптимизации.
* Размер эмбеддингов: слова и сущности представлялись в виде векторов размерности 300.
* Скорость обучения (Learning Rate): установлено начальное значение с механизмом адаптивного снижения (learning rate decay), если ошибка на валидационной выборке переставала уменьшаться в течение нескольких эпох.
* Регуляризация: для борьбы с переобучением активно применялся Dropout — техника, при которой на каждой итерации обучения случайным образом "выключается" часть нейронов, что заставляет сеть учить более робастные и обобщенные признаки.
* Размер батча (Batch Size): определяет количество обучающих примеров, обрабатываемых за одну итерацию.

Обучение продолжалось до тех пор, пока не прекращалось улучшение качества модели на валидационном наборе данных (ранняя остановка, early stopping), что позволило получить модель, способную хорошо обобщаться на новые, ранее не виденные запросы.

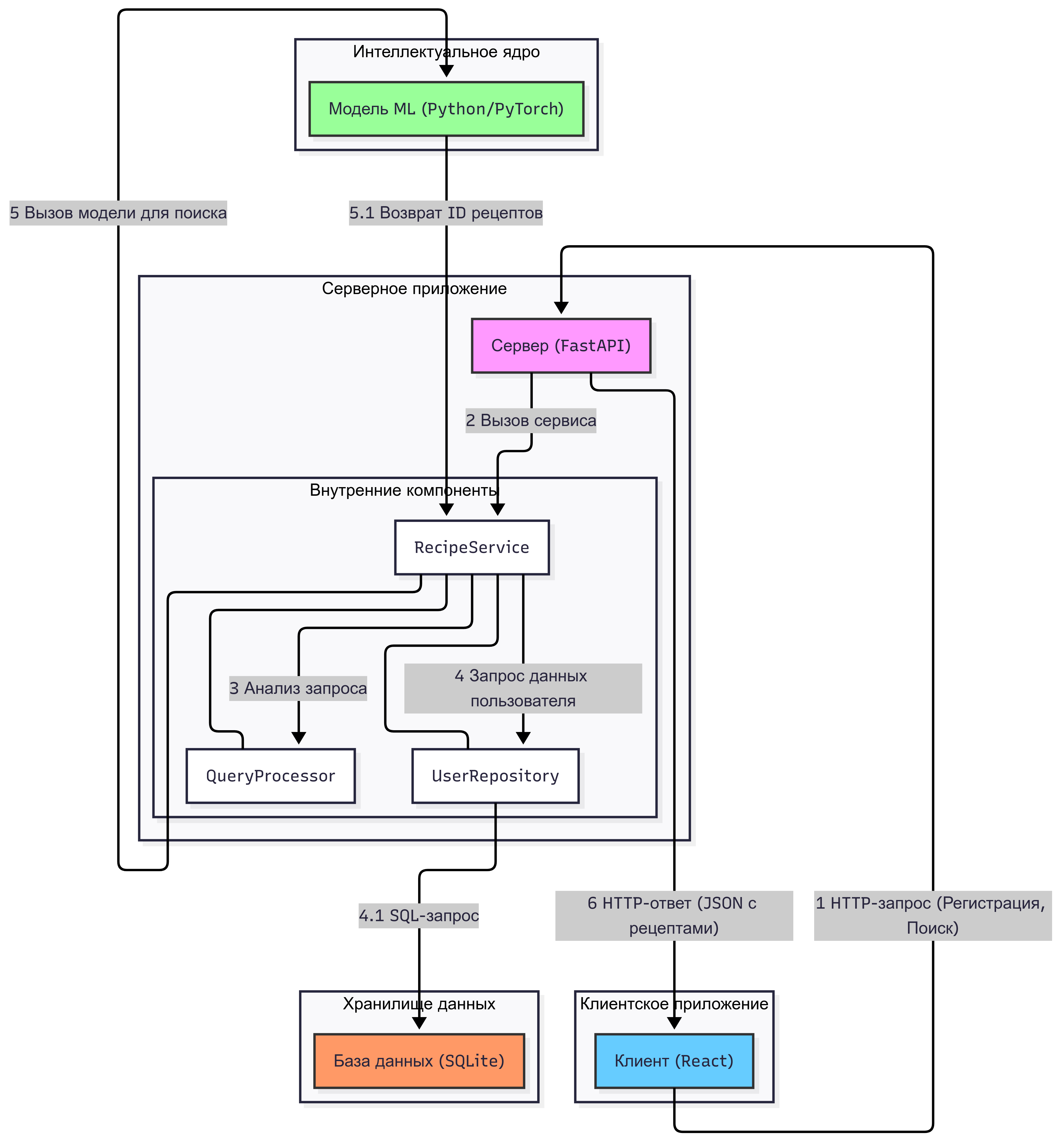
# Глава 3. Реализация проекта

Реализация проекта представляет собой создание комплексной программной системы, состоящей из трех тесно интегрированных, но архитектурно независимых компонентов: серверного приложения (Backend), клиентского веб-приложения (Frontend) и модуля машинного обучения, который является ядром интеллектуальной системы. В данном разделе подробно описана архитектура, стек технологий и ключевые аспекты реализации каждого из этих компонентов.

**Глава 3.1 Общая структура системы**

Архитектура проекта построена по классической клиент-серверной модели, где каждый компонент выполняет строго определенную роль, что обеспечивает модульность и масштабируемость системы.

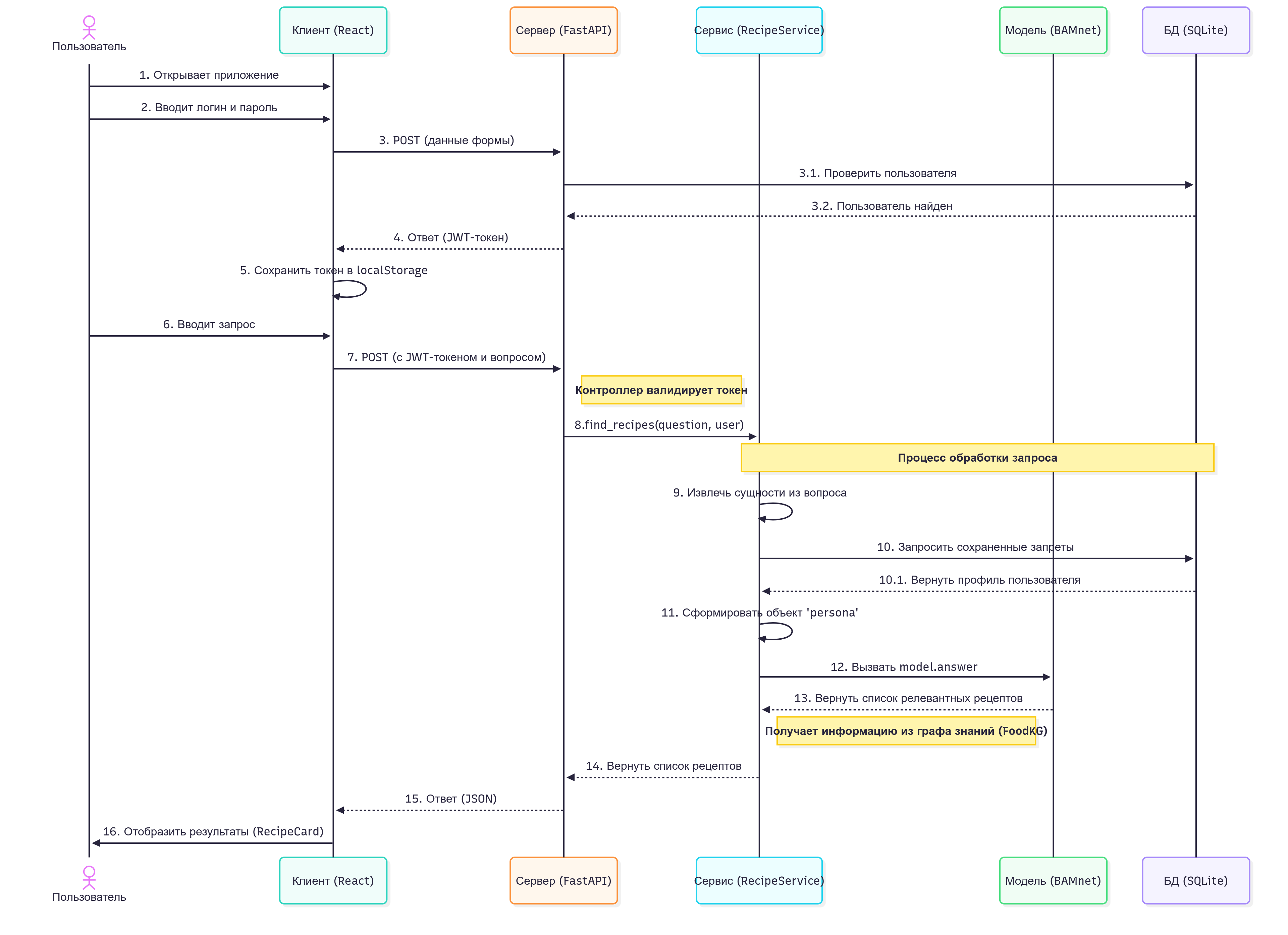
* **Frontend** (Клиент): одностраничное приложение (SPA), созданное на фреймворке React. Является точкой входа для пользователя. Отвечает за рендеринг пользовательского интерфейса, обработку действий пользователя (клики, ввод текста), управление состоянием сессии и асинхронное взаимодействие с бэкендом по REST API.
* **Backend** (Сервер): RESTful API, реализованный на Python с использованием асинхронного веб-фреймворка FastAPI. Бэкенд выполняет роль центрального координатора:
  + Управляет данными пользователей (регистрация, профили) в базе данных.
  + Реализует логику аутентификации и авторизации на основе JWT-токенов.
  + Принимает запросы от клиента, обрабатывает их и делегирует выполнение сложной логики сервисным модулям.
  + Взаимодействует с нейросетевой моделью.
* Модуль машинного обучения: состоит из нескольких Python-классов и представляет собой интеллектуальное ядро системы.
  + Модель BAMnet: Реализация архитектуры на PyTorch, загружающая предварительно обученные веса.
  + QueryProcessor: Класс-обработчик на базе spaCy для NLP-задач.
  + Эти компоненты интегрированы в бэкенд и загружаются в память при его старте.



*Figure 3 - Структура проекта*

Поток данных в системе:

1. Пользователь открывает веб-приложение в браузере.
2. Для доступа к персонализированным функциям он проходит регистрацию или аутентификацию через формы на фронтенде. Фронтенд отправляет POST-запрос на эндпоинт /api/v1/users/token бэкенда.
3. Бэкенд проверяет данные пользователя в базе данных и, в случае успеха, генерирует JWT-токен, который возвращается на фронтенд.
4. Фронтенд сохраняет токен в localStorage и в дальнейшем прикрепляет его к заголовкам всех защищенных запросов.
5. Пользователь переходит на страницу поиска рецептов и вводит запрос, например, «итальянская паста без грибов».
6. Фронтенд отправляет POST-запрос на эндпоинт /api/v1/recipes/ask, передавая текст вопроса.
7. Бэкенд-контроллер получает запрос, извлекает из JWT-токена информацию об аутентифицированном пользователе и его ID.
8. Запрос и данные пользователя передаются в RecipeService.
9. RecipeService использует QueryProcessor для извлечения сущностей из текста вопроса («паста» как тема, «грибы» как запрет).
10. RecipeService обращается к базе данных через UserRepository для получения сохраненных запретов пользователя (например, «арахис»).
11. Извлеченные и сохраненные ограничения объединяются, формируется объект persona.
12. RecipeService вызывает метод answer() у модели BAMnet, передавая ему вопрос и сформированный persona.
13. Модель BAMnet обрабатывает данные и возвращает список ID релевантных рецептов.
14. RecipeService получает ID, извлекает из графа знаний полную информацию о рецептах (название, URL, ингредиенты, нутриенты) и возвращает отформатированный список в контроллер.
15. Бэкенд отправляет JSON-ответ с детализированными рецептами на фронтенд.
16. Фронтенд получает данные и отображает их в виде интерактивных карточек RecipeCard.

*Figure 4 – Sequence диаграмма работы системы*

**Глава 3.2 Реализация серверной части**

Серверная часть спроектирована по принципам многослойной архитектуры (Layered Architecture), что обеспечивает четкое разделение ответственности между компонентами.

Стек технологий:

* Язык: Python 3.10+
* Фреймворк: FastAPI
* База данных: SQLite
* ORM: SQLAlchemy
* Аутентификация: python-jose для JWT, passlib для хеширования паролей.
* NLP: spaCy
* ML: PyTorch

Структура проекта:

/backend

├── src/

│ ├── repository/

│ │ ├── database.py # Настройка подключения к БД

│ │ ├── models.py # Модели таблиц SQLAlchemy

│ │ └── repository.py # Классы для работы с БД (UserRepository)

│ ├── service/

│ │ ├── core/

│ │ │ ├── bamnet.py # Реализация модели BAMnet

│ │ │ └── kbqa.py # Логика KBQA

│ │ ├── query\_processor.py # NLP-обработка запросов

│ │ ├── recipe\_service.py # Бизнес-логика рекомендаций

│ │ └── recipe\_data\_extractor.py # Логика получения рецептов из FoodKG

│ ├── main.py # API-эндпоинты (контроллеры)

│ ├── schemas.py # Pydantic-схемы для валидации

│ └── security.py # Логика безопасности и JWT

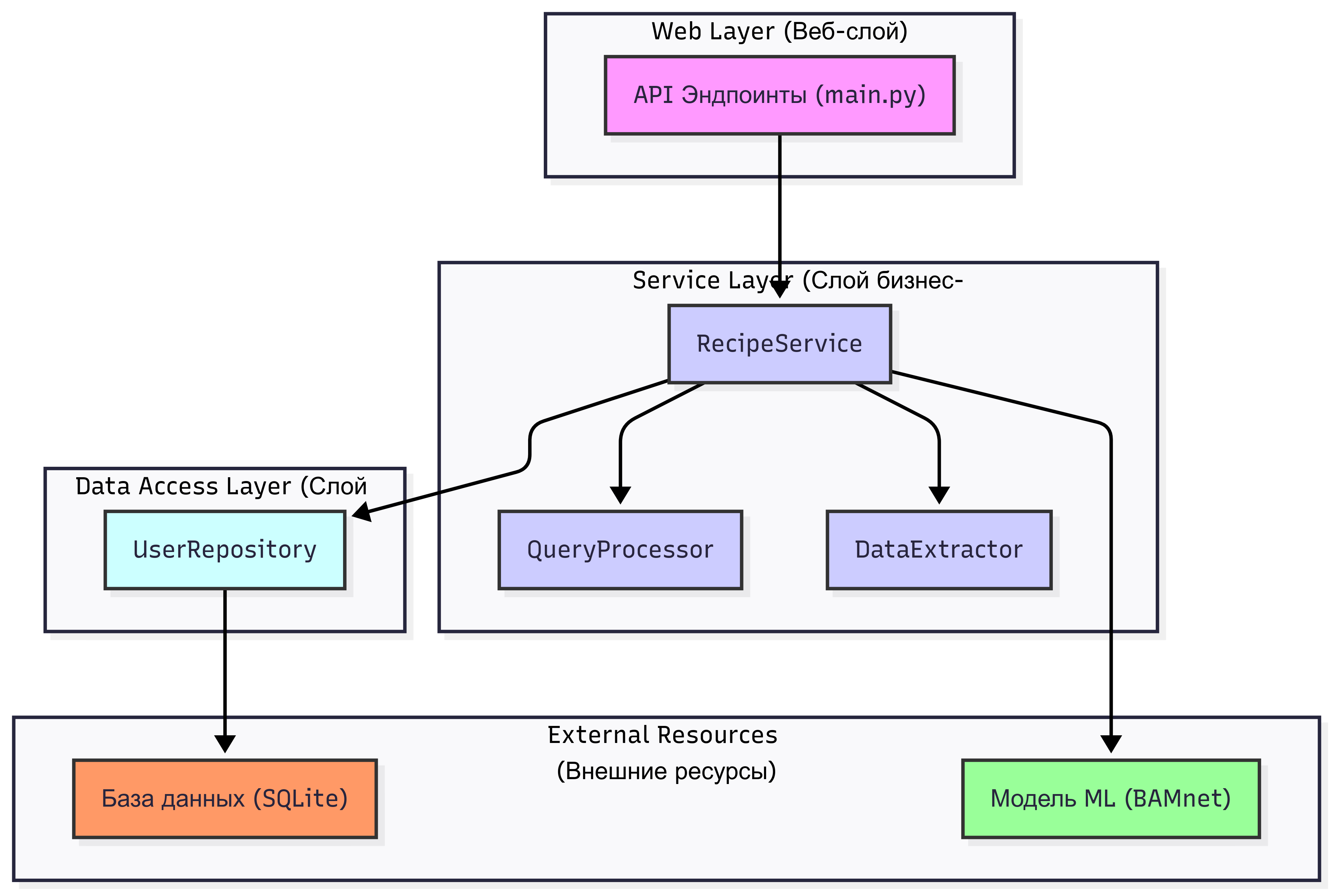
└── recipe\_finder.db # Файл базы данных SQLite

Реализация ключевых модулей:

1. Слой доступа к данным (repository/)
   * database.py: здесь настраивается engine SQLAlchemy для подключения к файлу recipe\_finder.db. Также здесь определена зависимость get\_db(), которая предоставляет сессию базы данных для каждого API-запроса.
   * models.py: определенет ORM-модель User, которая наследуется от Base из SQLAlchemy. Поля класса (id, email, hashed\_password, prohibited\_ingredients) напрямую соответствуют колонкам в таблице users. Тип JSON используется для гибкого хранения списка запрещенных ингредиентов.
   * repository.py: содержит класс UserRepository, который инкапсулирует всю логику работы с базой данных. Методы, такие как create(user\_create), get\_by\_email(email) и update\_preferences(user, preferences), выполняют конкретные SQL-операции (INSERT, SELECT, UPDATE) через ORM, скрывая детали реализации от остального приложения.
2. Слой бизнес-логики (service/)
   * QueryProcessor: при инициализации (один раз при старте сервера) загружает модель spaCy. Его главный метод process\_query принимает текст вопроса и профиль пользователя, выполняет синтаксический анализ для извлечения топиков и ограничений, а затем объединяет их с сохраненными в профиле запретами.
   * RecipeService: является центральным компонентом, координирующим процесс поиска. При инициализации он загружает обученную модель BAMnet и создает экземпляр QueryProcessor. Его основной метод find\_recipes выполняет следующую цепочку действий:
     1. Вызывает query\_processor для получения финальных списков топиков и запретов.
     2. Формирует сложный объект persona в формате, который ожидает модель BAMnet.
     3. Вызывает метод self.model.answer(...), передавая все необходимые данные.
     4. Обрабатывает ответ от модели (список ID рецептов) и, обращаясь к графу знаний, извлекает полную информацию по каждому рецепту.
     5. Возвращает отформатированный, детализированный список рецептов.

* RecipeDataExtractor: Его главные метод get\_dishes\_by\_urls(...) позволяет получать данные о списке рецептов, зная их ids.

1. Слой API (main.py)
   * Эндпоинты: с помощью декораторов FastAPI (@app.post, @app.get) определены все публичные маршруты API. Для каждого эндпоинта указаны ожидаемая модель запроса (request\_model) и модель ответа (response\_model) из schemas.py, что обеспечивает строгую валидацию.
   * Аутентификация: эндпоинты, требующие авторизации, используют зависимость Depends(security.get\_current\_user). Эта функция автоматически проверяет наличие и валидность Bearer токена в заголовке Authorization, декодирует его и возвращает объект User из базы данных.



*Figure 5- Структура серверной части программы*

**Глава 3.3 Реализация клиентской части (Frontend)**

Клиентская часть отвечает за интуитивно понятное и отзывчивое взаимодействие с пользователем.

Стек технологий:

* Фреймворк: React 18+
* Маршрутизация: react-router-dom
* HTTP-клиент: axios
* Стилизация: CSS Modules

Структура проекта:

/frontend

├── src/

│ ├── api/

│ │ └── api.js # Централизованные функции для запросов к API

│ ├── assets/ # Изображения, шрифты (включая фон)

│ ├── components/

│ │ ├── Navbar.jsx

│ │ ├── RecipeCard.jsx

│ │ └── ProtectedRoute.jsx # Компонент для защиты роутов

│ ├── context/

│ │ └── AuthContext.jsx # Глобальное состояние аутентификации

│ ├── pages/

│ │ ├── HomePage.jsx

│ │ ├── AuthPage.jsx

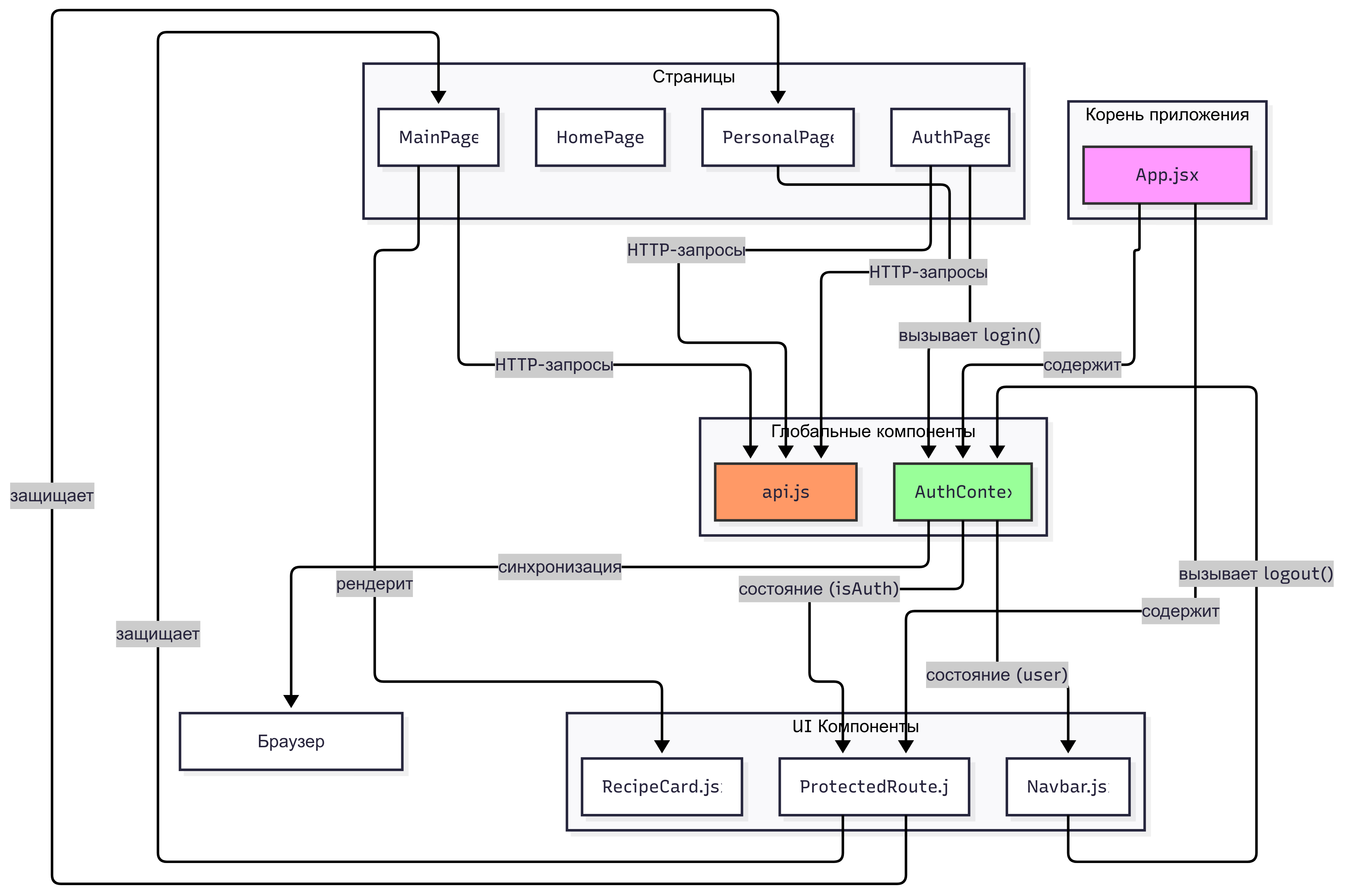
│ │ └── MainPage.jsx

│ ├── App.jsx # Главный компонент с маршрутизацией

│ └── index.css # Глобальные стили

Реализация ключевых модулей:

* Управление состоянием (AuthContext.jsx): Создан глобальный контекст, который предоставляет всему приложению информацию о текущем пользователе и его статусе аутентификации. Он синхронизирует токен с localStorage, что позволяет сохранять сессию пользователя между перезагрузками страницы.
* Маршрутизация (react-router-dom):
  + В файле App.jsx настроен главный маршрутизатор (BrowserRouter), который связывает URL-адреса со страницами-компонентами.
  + Компонент ProtectedRoute оборачивает защищенные роуты. Если пользователь не аутентифицирован, он автоматически перенаправляется на страницу входа (/auth).
* Основные компоненты:
  + HomePage: приветственная страница. Для стилизации текстового блока был применен подход "карточки" с белым полупрозрачным фоном, тенью и скругленными углами, что позволяет ему эффектно выделяться на фоне повторяющегося фонового изображения.
  + AuthPage: содержит управляемые формы для регистрации и входа. При отправке форм вызываются соответствующие функции из api.js для взаимодействия с бэкендом.
  + PersonalPage: позволяет пользователю управлять списком запрещенных ингредиентов. Список отображается с помощью переиспользуемого компонента IngredientList, а при сохранении данные отправляются на эндпоинт /users/me/preferences.
  + MainPage: содержит форму для поиска рецептов. При отправке формы формируется объект QuestionRequest (содержащий question и tags) и отправляется на эндпоинт /recipes/ask. Полученные результаты отображаются с помощью компонента RecipeCard.
  + RecipeCard: компонент, отвечающий за визуализацию одного рецепта. Он обернут в тег <a> так, что вся карточка является кликабельной ссылкой на оригинальный рецепт (recipe.dish\_url). Внутри отображается название, список ингредиентов и ключевые нутриенты.

*Figure 6- Структура клиентской части программы*

# Глава 4. Анализ и результаты

После завершения разработки и интеграции всех компонентов системы был проведен всесторонний анализ её функциональности, производительности и качества предоставляемых рекомендаций. Цель данного анализа — не только продемонстрировать работоспособность созданного прототипа, но и критически оценить эффективность выбранного архитектурного подхода, выявить его сильные стороны и определить потенциальные ограничения. Анализ проводился на основе серии репрезентативных сценариев использования, имитирующих реальное взаимодействие пользователя с системой.

**Глава 4.1 Качественный анализ работы web-приложения на сценариях**

Для оценки функциональности системы были смоделированы три ключевых сценария, каждый из которых проверяет отдельный аспект её работы: обработку простого запроса, распознавание ограничений на естественном языке и применение долгосрочной персонализации из профиля пользователя.

Сценарий 1: Базовый тематический запрос

* Ситуация: новый пользователь, успешно прошедший аутентификацию, хочет найти рецепты для ужина. Его профиль предпочтений пока пуст.
* Действия пользователя: в поле для ввода на главной странице вводится запрос: «Suggest turkish recipes with chicken and rice» (Посоветуй турецкие рецепты для ужина с курицей и рисом).
* Поведение системы:
  1. Frontend: формирует API-запрос к эндпоинту /api/v1/recipes/ask, включая в заголовок Authorization полученный при входе JWT-токен. Тело запроса: {"question": "Suggest turkish recipes with chicken and rice", "tags": [“turkish ”]}.
  2. Backend (RecipeService):
     + Получает запрос и объект current\_user (без персональных ограничений).
     + Передает текст вопроса в QueryProcessor.
     + QueryProcessor с помощью spaCy разбирает предложение и идентифицирует ключевые темы. Он распознает turkish recipes with chicken and rice как основную именную группу. После очистки от общих слов («recipes») извлекаются ключевые слова: ‘turkish’, 'chicken', 'rice'.
     + Поскольку в профиле пользователя и в тексте запроса нет запретов, persona формируется с пустым списком ingredient\_dislikes.
     + Сформированные топики передаются в модель BAMnet.
  3. Модель выполняет поиск в графе знаний, находя рецепты, которые одновременно связаны с «турецкое», «курица» и «рис». Модель ранжирует кандидатов по степени соответствия.
  4. На фронтенд возвращается JSON-объект, содержащий список отформатированных рецептов. Пользователь видит карточки таких блюд, как «Dolma», «Turkish lamb stew», «Bahar cerbasi».
* Вывод: Система успешно справляется с извлечением множественных тем из простого запроса на естественном языке и предоставляет релевантную подборку рецептов, демонстрируя базовую функциональность семантического поиска.

Сценарий 2: Запрос с явным негативным ограничением

* Ситуация: пользователь хочет приготовить блюдо из говядины, но не любит грибы.
* Действия пользователя: вводится запрос: «What are birthday dishes with beef dishes without mushrooms?» (Какие есть праздничные блюда из говядины без грибов?).
* Поведение системы:
  1. Backend (RecipeService):
     + QueryProcessor анализирует текст и извлекает:
       - Топик: [['birthday', 'tag']].
       - Запрет из вопроса: ['mushrooms']. spaCy идентифицирует слово without как маркер негативного ограничения и связывает его с объектом mushrooms.
     + RecipeService формирует persona с ingredient\_dislikes: ["mushrooms"].
     + Модель BAMnet получает задачу найти рецепты по теме «birthday» содержащие говядину и не содержащие грибы.
  2. Результат: пользователь получает список блюд, таких как «Aussie party pies» или «Lao beef jerky», но в выдаче отсутствуют «Beef Stroganoff» и «Steak with Mushroom Sauce».
* Вывод: Система демонстрирует способность корректно интерпретировать негативные конструкции в естественном языке и использовать их как строгие фильтры при поиске, что является значительным преимуществом перед простым поиском по ключевым словам.

Сценарий 3: Комплексный сценарий с персонализацией

* Ситуация: пользователь имеет аллергию на морепродукты и непереносимость лактозы. Он предварительно занес в свой профиль на странице /profile в список запрещенных ингредиентов «shrimp» и «milk».
* Действия пользователя: пользователь ищет вдохновение для праздничного ужина и вводит общий запрос: «Recommend some fancy dinner recipes» (Посоветуй несколько изысканных рецептов для ужина).
* Поведение системы:
  1. Backend (RecipeService):
     + Контроллер идентифицирует пользователя по JWT-токену.
     + RecipeService обращается к UserRepository и получает из профиля пользователя список prohibited\_ingredients: ["shrimp", "milk"].
     + Формируется итоговый объект persona, где ingredient\_dislikes содержит ["shrimp", "milk"].
     + Модель BAMnet получает задачу найти рецепты с тегами «изысканный» и «ужин», при этом отфильтровав все кандидаты, содержащие креветки или молоко.
  2. Результат: Система не предложит пользователю такие блюда, как «Lobster Thermidor» (морепродукты) или «Chicken in Creamy Sauce» (молоко). Вместо этого она может порекомендовать «Roasted Duck with Cherry Sauce» или «Vegan Mushroom Wellington» как релевантные и безопасные для данного пользователя варианты.
* Вывод: данный пример демонстрирует ключевую особенность и главное преимущество разработанной системы — способность к проактивной персонализации. Системе не нужно, чтобы пользователь каждый раз перечислял свои ограничения. Она автоматически применяет сохраненные в профиле правила к любому, даже самому общему запросу, обеспечивая высокий уровень релевантности и безопасности рекомендаций.

**Глава 4.2 Анализ производительности и ограничений**

Критическая оценка реализованного прототипа позволяет выявить как его сильные стороны, так и области для потенциальных улучшений.

**Сильные стороны проекта:**

1. **Гибкость и интуитивность интерфейса:** Использование естественного языка для формулирования запросов делает систему значительно более дружелюбной для конечного пользователя по сравнению с системами, требующими ручного выбора десятков фильтров.
2. **Эффективность персонализации:** Система успешно объединяет сиюминутные и долгосрочные предпочтения пользователя, что является ее главным функциональным преимуществом. Модель обработки запретов показала свою надежность в тестовых сценариях.
3. **Высокая модульность и чистота архитектуры:** Реализация по многослойному принципу (API Layer, Service Layer, Data Access Layer) значительно упрощает поддержку, тестирование и дальнейшее расширение проекта. Например, для добавления нового источника данных потребуется изменить только Data Access Layer, не затрагивая бизнес-логику и API.
4. **Интерпретируемость результатов:** Возвращая детализированную информацию о рецепте, включая ингредиенты и нутриенты, система позволяет пользователю понять, почему ему была предложена та или иная рекомендация, что повышает доверие к сервису.

**Выявленные ограничения и направления для развития:**

1. **Точность NLP-модуля:** spaCy является мощным инструментом, однако его точность при разборе сложных или нестандартно сформулированных предложений может быть недостаточной. Например, запросы с двойным отрицанием или сложными подчинительными связями могут быть интерпретированы неверно.
2. **Обработка сложных типов ограничений:** Текущая реализация сфокусирована на категориальных ограничениях (наличие/отсутствие ингредиента).
3. **Производительность и масштабирование:** Загрузка нейросетевой модели BAMnet и языковой модели spaCy при старте сервера является ресурсоемкой операцией.
4. **Проблема «холодного старта»:** для новых пользователей система функционирует как обычный семантический поиск, так как их профиль пуст.

Реализованный проект успешно демонстрирует высокий потенциал применения KBQA-подхода для создания интеллектуальных рекомендательных систем. Он эффективно решает задачу персонализации и предоставляет гибкий интерфейс для пользователя. Выявленные ограничения не являются концептуальными недостатками, а представляют собой четко очерченные направления для дальнейшей итеративной доработки и совершенствования системы.

**Глава 4.3 Возможности для улучшения и дальнейшего развития**

Несмотря на успешную реализацию прототипа и демонстрацию жизнеспособности выбранного подхода, текущая версия системы имеет ряд ограничений, которые открывают широкие возможности для дальнейшего совершенствования и развития проекта. Эти направления можно сгруппировать по нескольким ключевым областям.

**Повышение точности и гибкости NLP-модуля**: текущий QueryProcessor на базе spaCy эффективно справляется со стандартными запросами, однако его возможности ограничены при работе со сложными лингвистическими конструкциями.

* Сложные синтаксические структуры: Запросы с двойными отрицаниями («посоветуй что-нибудь, что не является не-вегетарианским»), условными конструкциями или неоднозначными формулировками могут быть разобраны некорректно.
* Извлечение неявных сущностей: Система может не распознать синонимы или описательные характеристики (например, «что-нибудь из постного мяса» вместо прямого указания «курица» или «индейка»).

Путь развития:

* + Интеграция моделей на основе Трансформеров: Использование более мощных моделей, таких как BERT или его производных (например, RoBERTa, DistilBERT), для задач извлечения именованных сущностей (NER) и анализа отношений. Эти модели способны улавливать более глубокий семантический контекст и могут быть дообучены на специфической кулинарной лексике для повышения точности.
  + Расширение онтологий: Интеграция дополнительных тезаурусов и онтологий для распознавания синонимов и гипонимов (например, понимание, что «говядина» и «телятина» — это виды мяса).

**Поддержка сложных типов ограничений**: текущая система отлично обрабатывает категориальные ограничения (наличие/отсутствие ингредиента), но ее функционал можно значительно расширить.

* Числовые и сравнительные ограничения: Система пока не поддерживает запросы вида «рецепты с содержанием калорий *не более 500*» или «блюдо, *менее жирное, чем лазанья*».
* Временные и контекстуальные ограничения: Запросы типа «что приготовить *за 20 минут*» или «что-нибудь, что *хорошо сочетается с белым вином*» требуют более сложного моделирования связей в графе знаний.

Путь развития:

* + Реализация KG Augmentation: Внедрение механизма, описанного в статье PFoodReq, который динамически добавляет в граф знаний новые сущности и отношения, представляющие числовые диапазоны (например, узел «калории\_400-500»). Это позволит модели обрабатывать числовые запросы как стандартные категориальные ограничения.
  + Развитие модели рассуждений (Reasoning Module): Усложнение архитектуры нейронной сети для выполнения сравнительных операций и многошаговых логических выводов (multi-hop reasoning) на графе.

**Улучшение пользовательского опыта (UX)**

* Обратная связь: В текущей версии пользователь не получает объяснения, почему тот или иной рецепт был отфильтрован (например, «Этот рецепт не показан, так как содержит арахис из вашего списка запретов»).

Путь развития:

* + Создание интерактивного интерфейса: Добавление всплывающих подсказок или специальных иконок на карточках рецептов, которые бы объясняли, какие именно ограничения были применены. Это повысит прозрачность работы системы и доверие к ней со стороны пользователя.
  + Диалоговый интерфейс: Развитие системы в сторону чат-бота или голосового ассистента, который мог бы вести диалог с пользователем, уточнять запросы и предлагать альтернативы.

**Техническая оптимизация и масштабирование:** прототип системы реализован с упором на функциональность, но для промышленного использования потребуется техническая оптимизация.

* Производительность модели: Загрузка модели BAMnet и spaCy при старте сервера является ресурсоемкой и может приводить к длительному времени холодного старта.
* Актуальность данных: Граф знаний FoodKG является статичным. Новые рецепты и данные о продуктах появляются постоянно.

Путь развития:

* + Выделенный инференс-сервис: Вынесение нейросетевой модели в отдельный микросервис с использованием специализированных инструментов, таких как NVIDIA Triton Inference Server или TorchServe. Это позволит независимо масштабировать веб-сервер и модель, а также использовать техники квантизации и оптимизации для ускорения инференса.
  + Разработка конвейера обновления данных (Data Pipeline): Создание автоматизированного процесса для регулярного парсинга популярных кулинарных сайтов, извлечения новых рецептов и их интеграции в граф знаний FoodKG для поддержания его актуальности.

Реализация этих улучшений позволит превратить текущий прототип в полноценный, высокопроизводительный и еще более интеллектуальный продукт, способный занять свою нишу на рынке цифровых сервисов для здорового питания.

# Заключение

В ходе выполнения данной дипломной работы была успешно решена поставленная задача по разработке и реализации прототипа интеллектуальной системы для персонализированной рекомендации рецептов. Проект продемонстрировал высокий потенциал применения современных подходов из области обработки естественного языка и машинного обучения для решения актуальной и социально значимой проблемы подбора индивидуального рациона питания.

Для реализации была спроектирована и создана комплексная программная система с четким разделением ответственности. Серверная часть, реализованная на Python с использованием фреймворка FastAPI, выступила в роли центрального координатора, управляющего бизнес-логикой, доступом к данным и взаимодействием с моделью. Клиентская часть, разработанная на React, обеспечила современный, отзывчивый и удобный пользовательский интерфейс для регистрации, управления профилем и получения рекомендаций.

Интеллектуальным ядром системы стала адаптированная архитектура BAMnet, которая с помощью двунаправленных механизмов внимания эффективно сопоставляет семантику пользовательского запроса со структурированными данными из масштабного графа знаний FoodKG. Для анализа текста и извлечения сущностей был успешно интегрирован NLP-модуль на базе библиотеки spaCy.

В результате были достигнуты все поставленные цели и задачи:

* Проведен анализ и выбрана релевантная нейросетевая архитектура.
* Спроектирована и реализована многослойная архитектура программного комплекса.
* Создан полнофункциональный бэкенд с REST API, аутентификацией и доступом к базе данных.
* Разработано интерактивное клиентское приложение, демонстрирующее все ключевые сценарии использования.
* Проведена успешная интеграция всех компонентов в единую работающую систему.

Качественный анализ работы прототипа на репрезентативных примерах подтвердил его работоспособность и эффективность. Система корректно обрабатывает как простые тематические запросы, так и сложные персонализированные сценарии, предоставляя пользователю релевантные и безопасные рекомендации.

В то же время, в ходе работы были выявлены и четко очерчены направления для дальнейшего развития, включая повышение точности NLP-модуля, поддержку более сложных числовых и сравнительных ограничений, улучшение пользовательского опыта и техническую оптимизацию для промышленного использования.

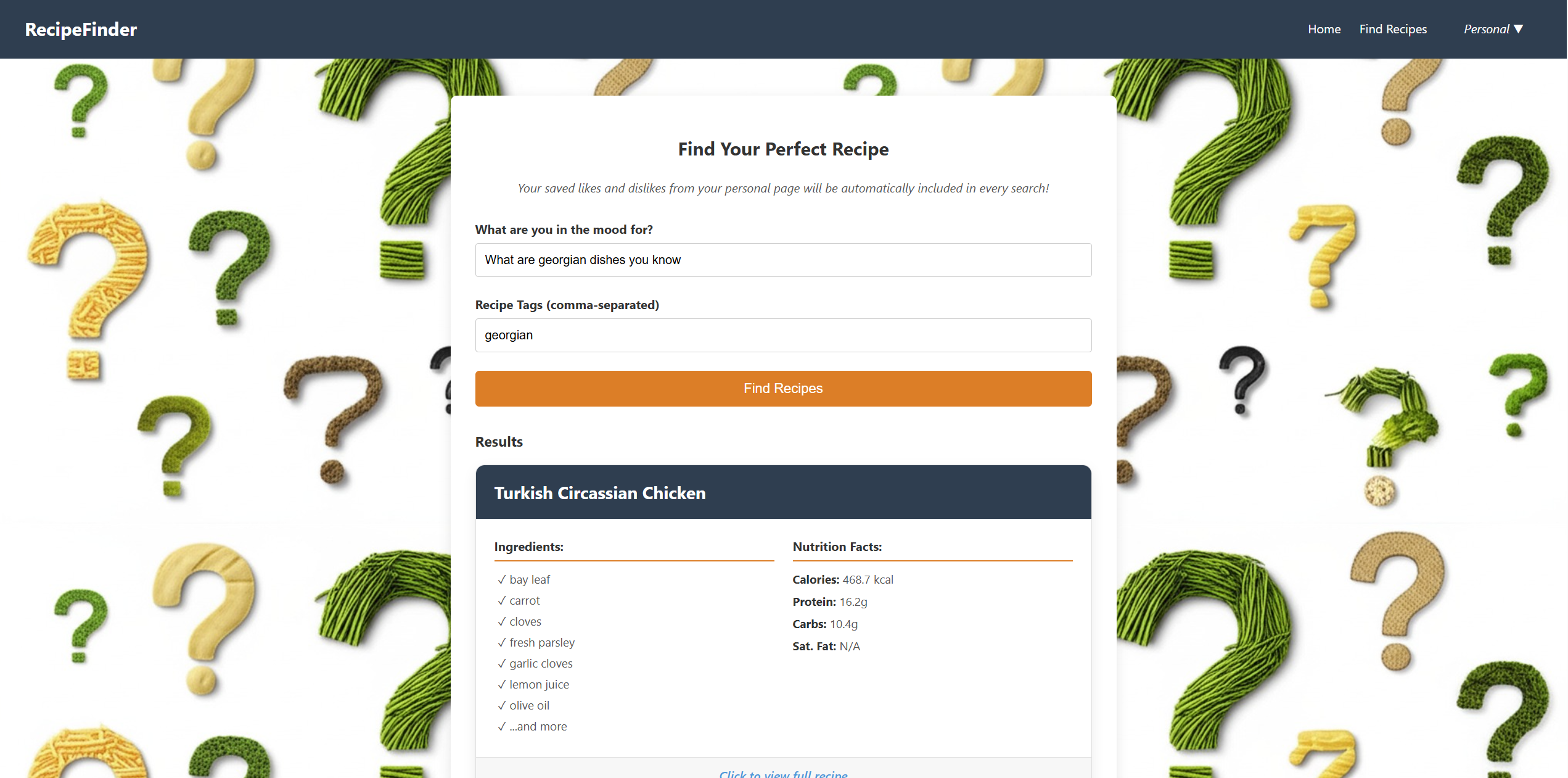
Таким образом, данная работа не только представляет собой успешное решение сложной инженерной задачи, но и закладывает прочный архитектурный и методологический фундамент для создания нового поколения интеллектуальных сервисов в области здорового питания и персонализированной диетологии.

# Использованная литература

1. **Chen, Y., Wu, L., & Zaki, M. J. (2019).** *Bidirectional Attentive Memory Networks for Question Answering over Knowledge Bases.* Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT).
2. **Chen, Y., Subburathinam, A., Chen, C. H., & Zaki, M. J. (2021).** *Personalized Food Recommendation as Constrained Question Answering over a Large-scale Food Knowledge Graph.* Proceedings of the Fourteenth ACM International Conference on Web Search and Data Mining (WSDM '21).
3. **Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017).** *Attention is all you need.* Advances in neural information processing systems.
4. **Haussmann, S., Seneviratne, O., Chen, Y., Ne'eman, Y., Codella, J., Chen, C. H., ... & Zaki, M. J. (2019).** *FoodKG: A semantics-driven knowledge graph for food recommendation.*
5. **Berant, J., Chou, A., Frostig, R., & Liang, P. (2013).** *Semantic parsing on freebase from question-answer pairs.*
6. **Bordes, A., Chopra, S., & Weston, J. (2014).** *Question answering with subgraph embeddings.*

# Приложения

**Приложение 1. Графический интерфейс приложения**

Figure 1 - Внешний вид приложения

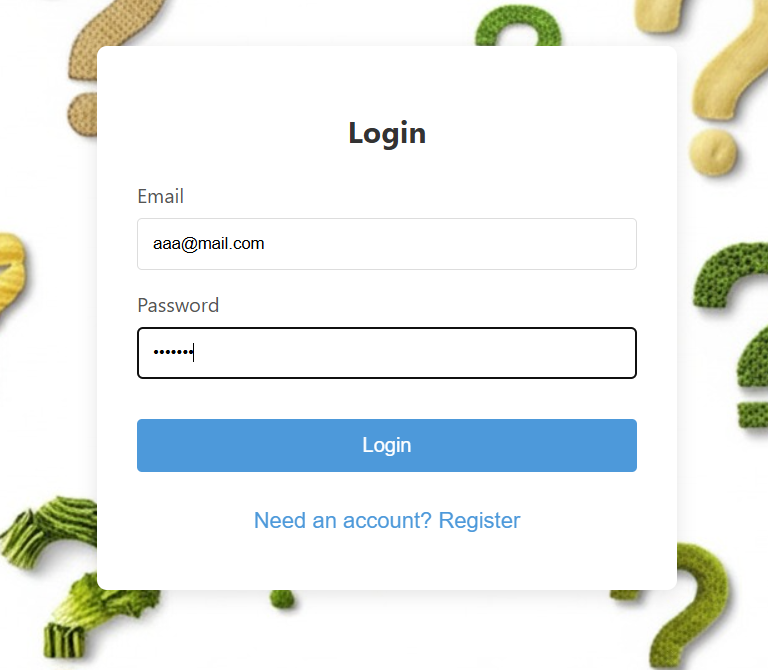
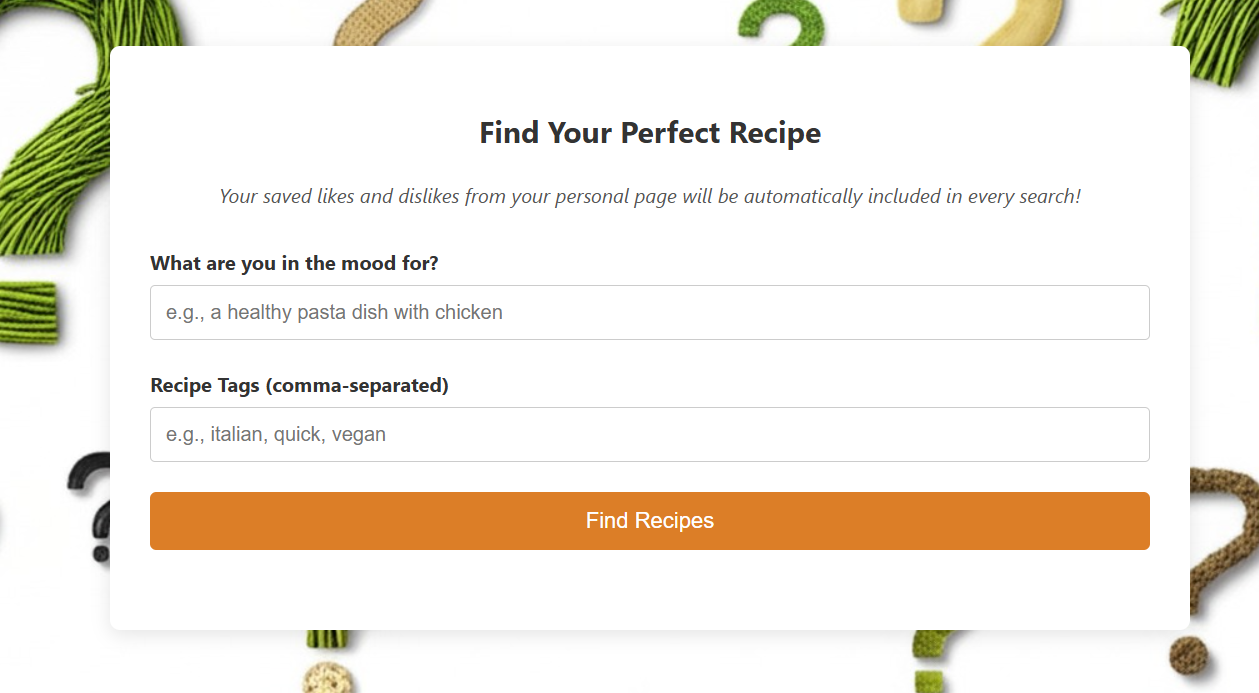
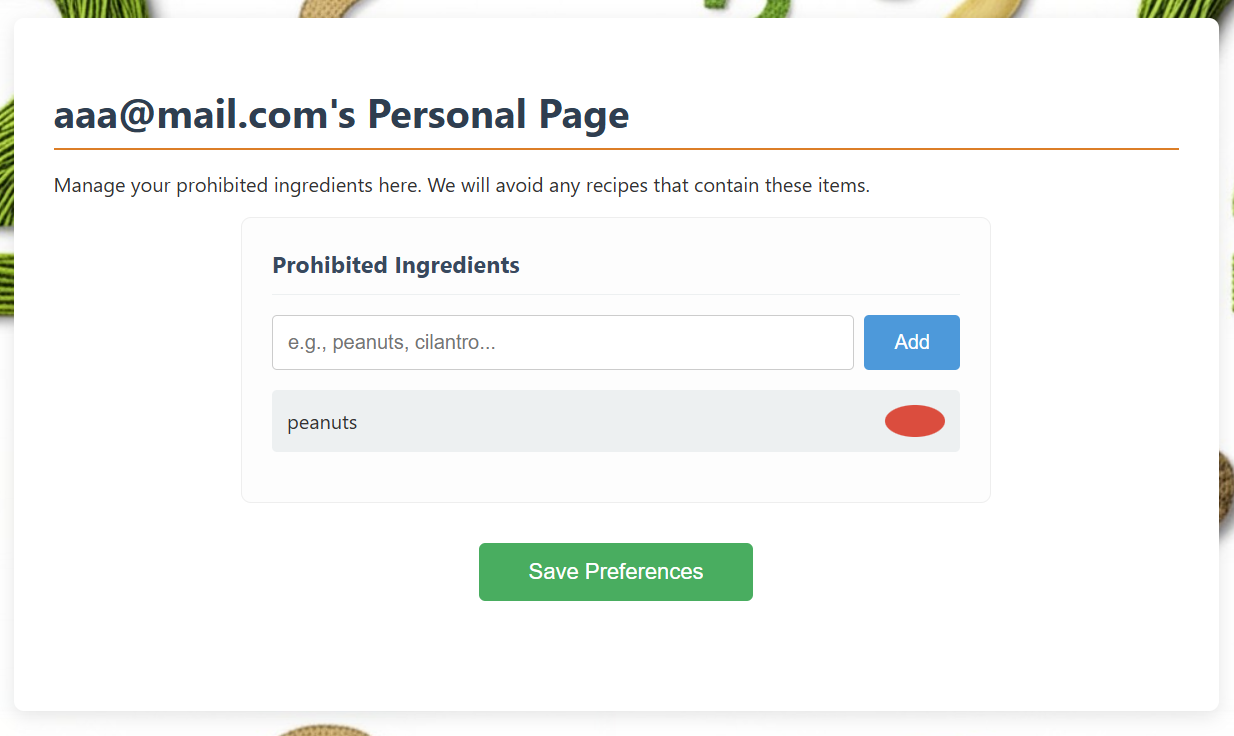
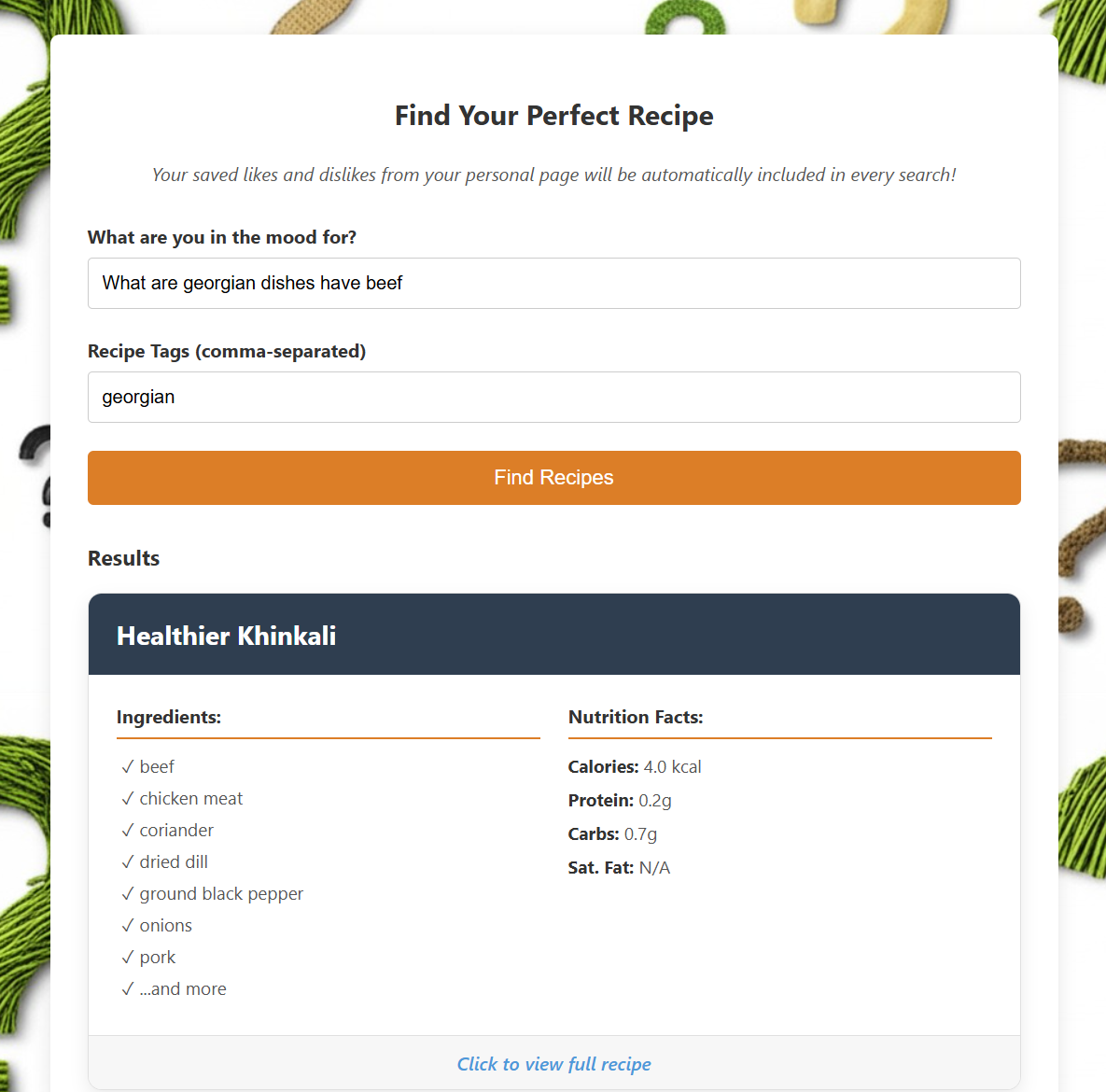


Figure 2 - Форма для входа

Figure 3 - Форма для поиска рецептов

Figure 4 - Личный кабинет пользователя

Figure 5 - Окно поиска и результаты