Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| институт |
| Программная инженерия |
| кафедра |

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №5**

|  |
| --- |
| Реализация и описание паттернов проектирования |
| тема |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  | Д. В. Грузенкин |
|  | |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ22-16/2б, 032211631 |  |  |  | М.Н. Карыбеков |
|  | номер группы, зачетной книжки |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2025

СОДЕРЖАНИЕ

[Цель 3](#_Toc190524544)

[Задачи 3](#_Toc190524545)

[UML-диаграмма вариантов использования 4](#_Toc190524548)

[Бизнес-требования 4](#_Toc191249806)

[Требования пользователей 6](#_Toc191249807)

[Функциональные требования 6](#_Toc191249808)

[Нефункциональные требования 7](#_Toc191249811)

[Источники получения требований 8](#_Toc191249812)

Цель

Описать выбранную систему.

Задачи

Выбрать (придумать) систему (название и назначение), архитектуру которой вы будете разрабатывать в течение семестра. Оформить отчёт по СТУ, где привести помимо цели и задач работы название системы, краткое описание её назначения и характеристик, а также UML-диаграмму вариантов использования описываемой системы.

Описание системы

Название: «CookGuide».

Описание и характеристики: это мобильное приложение для поиска, сохранения и организации кулинарных рецептов. Приложение предназначено для помощи пользователям в быстром подборе рецептов по доступным ингредиентам, диетическим предпочтениям и категориям блюд. Пользователь может сохранять любимые рецепты, создавать собственные подборки и получать рекомендации на основе своих вкусов. Информацию о заведениях и магазинах можно будет просмотреть на картах интернет, что также предусмотрено.

UML-диаграмма вариантов использования

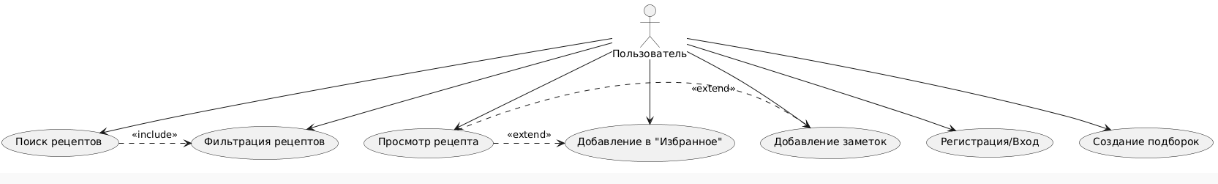


Рисунок 1 — Диаграмма вариантов использования системы поиска и добавления рецептов авторизованного пользователя



Рисунок 2 — Диаграмма вариантов использования системы поиска и добавления рецептов невторизованного пользователя

**Бизнес-требования**

1.Бизнес-возможности и проблемы

Как заказчик и потенциальный пользователь системы, я сталкиваюсь со следующими проблемами:

* Сложность в быстром подборе подходящих рецептов без необходимости длительных поисков;
* Отсутствие удобного способа фильтрации рецептов по доступным

ингредиентам, сложности приготовления или предпочтениям;

* Нет возможности создавать свои подборки рецептов для разных

случаев (например, "завтраки", "праздничный стол");

* Потребность в приложении, которое работает без постоянного подключения к Интернету и хранит рецепты локально.

2. Бизнес-цели

Разработка и внедрение системы «CookGuide» должно привести к

* Сокращению времени поиска подходящего рецепта на 70% за счёт интеллектуальной фильтрации
* Повышению доступности рецептов офлайн, что позволит использовать приложение даже без интернета;
* Улучшению пользовательского опыта за счёт удобного интерфейса и быстрой навигации.
* Повышение разнообразия блюд на каждый день

**Функциональные требования**

1. Авторизация и управление пользователями

* Регистрация нового пользователя через email или Google-аккаунт;
* Вход в приложение с сохранением сессии;
* Управление учетной записью (редактирование профиля, выход из
* аккаунта).

1. Работа с рецептами

* Поиск рецептов по ключевым словам;
* Фильтрация рецептов по категориям, сложности и времени приготовления;
* Просмотр детальной информации о рецепте (ингредиенты, шаги приготовления, время готовки);
* Сохранение рецептов в "Избранное";
* Создание пользовательских подборок рецептов
* Добавление заметок к сохранённым рецептам.

1. Работа офлайн

* Доступ к ранее сохранённым рецептам без подключения к Интернету

1. Уведомления

* Отправка уведомлений о новых рецептах и обновлениях подборок.

**Нефункциональные требования**

* Поддержка оффлайн-доступа без задержек;
* Оптимизированный интерфейс под разные размеры экранов;
* Совместимость с Android 10 и выше;
* Работа без сложной регистрации (простой вход через Google или email без дополнительных подтверждений).

**Источники и методы получения требований**

1. Интервью с заказчиком

**Разработчик**: Расскажите, пожалуйста, какой функционал вы хотели бы видеть в вашем приложении рецептов?

**Заказчик**: Мы хотим, чтобы пользователь мог быстро находить нужные рецепты по ингредиентам и сохранять их. Также важно, чтобы была возможность создавать свои подборки например, «любимые рецепты», «на ужин» и так далее.

**Разработчик**: То есть поиск и фильтрация рецептов по категориям и ингредиентам, а также сохранение в избранное и собственные подборки?

**Заказчик**: Да, именно. И ещё чтобы пользователь мог добавлять свои заметки к рецептам например, «добавить больше соли» или «не готовить дольше 10 минут».

**Разработчик**: А пользователи должны как-то регистрироваться?

**Заказчик**: Да, регистрация должна быть хотя бы через почту или Google. Без неё только просмотр в демо-режиме. Но чтобы сохранять рецепты и подборки нужен аккаунт.

**Разработчик**: Какие основные действия пользователь должен иметь возможность выполнять?

**Заказчик**: Он должен:

* Возможность зарегистрироваться и войти в систему через Google или электронную почту;
* Искать рецепты по названию или ингредиентам;
* Использовать фильтры по типу кухни, сложности и времени приготовления;
* Сохранять понравившиеся рецепты в "Избранное";
* Создавать тематические подборки рецептов;
* Добавлять личные заметки к рецептам;
* Использовать сохранённые рецепты без подключения к Интернету;
* Получать уведомления о новых рецептах и подборках;
* Использовать удобный и быстрый интерфейс без сложных настроек.

**Разработчик**: Нужно ли предусмотреть редактирование и удаление своих подборок и заметок?

**Заказчик**: Да, обязательно. Люди должны иметь возможность изменять свои данные, если их предпочтения поменялись.

**Разработчик**: А что насчёт администраторов? Или система будет полностью пользовательской?

**Заказчик**: Для начала можно обойтись без них, но в будущем можно предусмотреть управление базой рецептов через административную панель.

**Разработчик**: На каких устройствах планируется использование приложение?

**Заказчик**: Это будет мобильное приложение в первую очередь под Android. Важно, чтобы оно работало без подключения к Интернету для сохранённых рецептов.

1. Анализ существующих решений

* Проведён обзор популярных кулинарных приложений, таких как Tasty, Kitchn Stories, Yummly:
* Выявлены их сильные стороны (например, удобный поиск);
* Обнаружены недостатки (например, необходимость подписок для сохранения рецептов)
* Учтены лучшие практики и избегание перегрузки рекламы.

**Выбор архитектурного паттерна**

При проектировании системы отслеживания транспорта для Android было принято решение использовать архитектурный паттерн Client-Server + MVVM.

**Обоснование выбора архитектурного паттерна**

1. Клиент-Серверная архитектура

Система состоит из мобильного клиента (Android-приложение) и серверной части Firebase.

Firebase – облачная платформа, обеспечивающая:

1. Базу данных для хранения информации о пользователях и транспортных средствах;
2. Аутентификацию через Firebase Authentication;
3. Реальное обновление данных, что особенно важно для трекинга транспорта;
4. Простоту интеграции с Android-приложением через SDK;

Благодаря этим возможностям Firebase отлично вписывается в архитектуру Client-Server и позволяет избежать сложностей с ручным развертыванием серверной части.

**Обоснование выбора MVVM (Model-View-ViewModel**

* 1. Разделить ответственность между слоями приложения:
     + Model – управляет данными и бизнес-логикой (работа с Firebase);
     + View (Activity/Fragment) – отвечает за отображение данных и обработку пользовательского ввода.
     + View (Activity / Fragment) – отображает данные, реагирует на пользовательский ввод и делегирует обработку ViewModel.
  2. Упрощённое масштабирование и сопровождение:
* Благодаря слабой связанности компонентов, добавление новых функций (например, фильтрация рецептов по калорийности) минимально влияет на уже реализованные части.

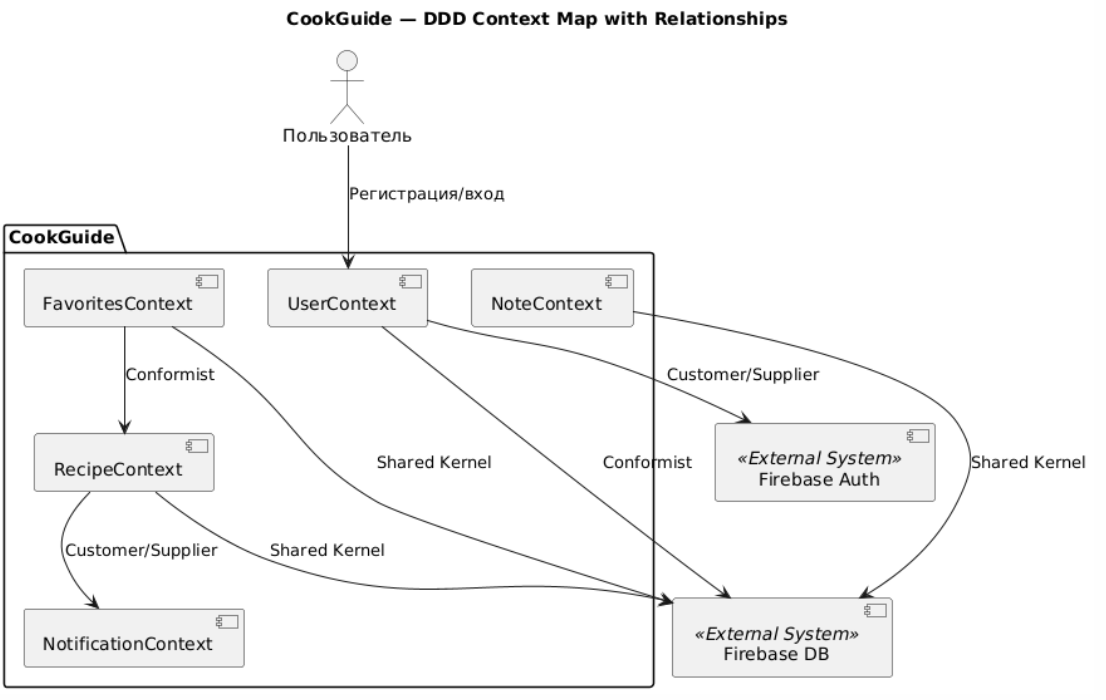


Рисунок 3 – Карта контекстов по методолгии DDD

### 1. **RecipeContext** (Смысловое ядро — Core Domain)

Основной контекст системы. Содержит бизнес-логику, связанную с рецептами: создание, фильтрация, поиск, категоризация.  
**Почему ядро:**  
Именно вокруг рецептов строится всё приложение — это центральная модель и основная пользовательская ценность.

**Тип взаимодействия:**

* Shared Kernel с FavoritesContext, NoteContext — они используют те же структуры данных рецепта.
* Customer по отношению к NotificationContext — инициирует уведомления.

### 2. **UserContext** (Вспомогательный контекст — Supporting)

Обеспечивает регистрацию, авторизацию и хранение пользовательской информации.

**Тип взаимодействия:**

* Conformist по отношению к Firebase Auth — принимает интерфейс внешнего поставщика как есть.
* Customer/Supplier по отношению к RecipeContext — пользовательский запрос инициирует работу с ядром.

### 3. **FavoritesContext** (Вспомогательный контекст)

Содержит логику работы с «Избранным»: сохранение, удаление, отображение подборок рецептов.

**Тип взаимодействия:**

* Shared Kernel с RecipeContext — оперирует идентификаторами и типами рецептов.
* Customer по отношению к RecipeContext — получает данные по ID рецептов.

### 4. **NoteContext** (Вспомогательный контекст)

Позволяет пользователю добавлять к рецептам личные заметки.

**Тип взаимодействия:**

* Shared Kernel с RecipeContext — должен «знать» рецепт, к которому добавляется заметка.

### 5. **NotificationContext** (Вспомогательный контекст — Event-Driven)

Обрабатывает события и уведомляет пользователя (например, о новых рецептах, обновлениях в подборках).

**Тип взаимодействия:**

* Supplier по отношению к RecipeContext — получает событие и реагирует (рассылка уведомлений).
* Может быть обособлен и асинхронен — что делает его удобным для масштабирования.

### 6. **Внешние системы** (не являются доменными контекстами, но входят в карту):

* **Firebase Auth** — используется UserContext, тип связи: Customer/Supplier
* **Firebase DB** (если включали ранее) — обычно доступен всем контекстам как Conformist (единое API-хранилище)

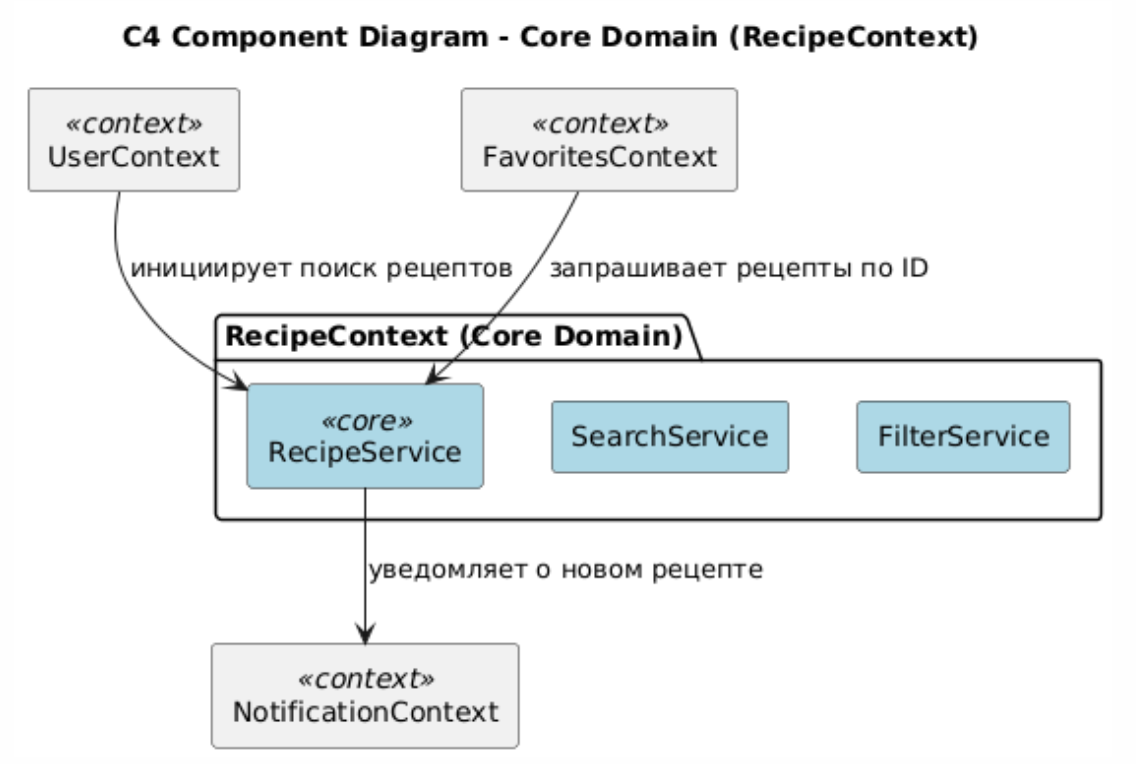


Рисунок 4 – Диаграмма контекстов

### Контексты на диаграмме:

#### 1. **RecipeContext (ядро)**

Главный модуль, содержащий бизнес-логику работы с рецептами. Представляет собой **смысловое ядро (core domain).  
Компоненты:**

* RecipeService — предоставляет рецепты по ID или категории;
* SearchService — выполняет полнотекстовый поиск по названию/ингредиентам;
* FilterService — фильтрует рецепты по параметрам (время, диета, т.п.).

#### 2**. UserContext (вспомогательная область)**

Отвечает за действия пользователя, включая авторизацию и выбор рецептов.  
**Роль в этой диаграмме:** инициирует поиск и обращается к RecipeService.

#### 3. **FavoritesContext (вспомогательная область)**

Управляет избранным.  
**Роль:** обращается к RecipeService для получения данных о сохранённых рецептах по ID.

#### **4. NotificationContext (вспомогательная область)**

Уведомляет пользователя о событиях в системе.  
**Роль:** получает сигналы от RecipeService о добавлении новых рецептов и рассылает уведомления.

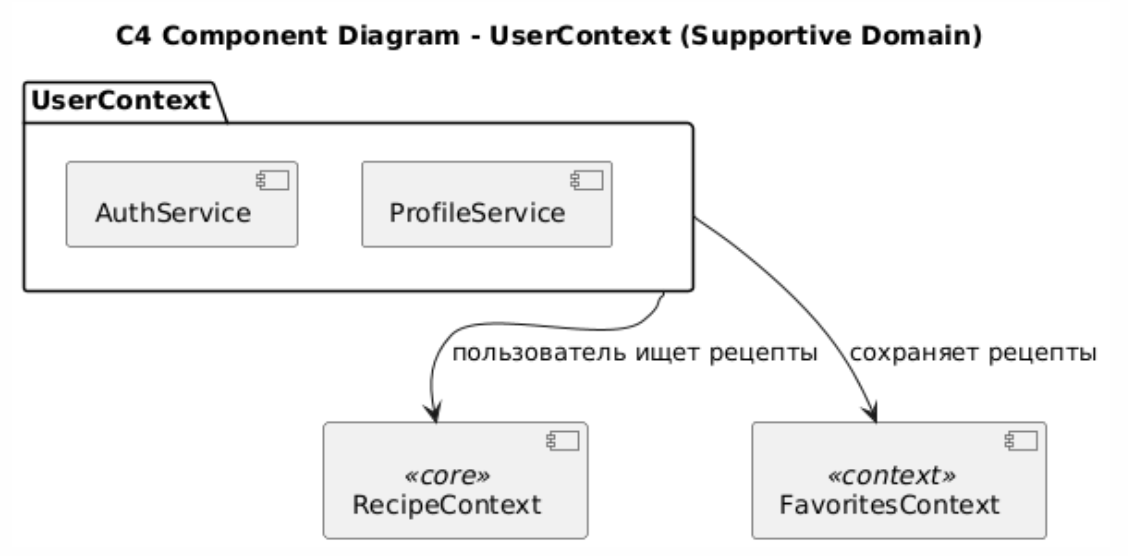


Рисунок 5 – Диаграмма

#### **1. UserContext (вспомогательный контекст)**

Отвечает за управление пользователем и его действия в системе.  
**Компоненты:**

* AuthService — обеспечивает регистрацию и вход пользователя;
* ProfileService — управление пользовательским профилем (имя, предпочтения, настройки и т.п.).

#### **2. RecipeContext (ядро)**

Контекст рецептов, принимает запросы от пользователя на поиск, просмотр и фильтрацию.

#### **3. FavoritesContext**

Контекст избранного, взаимодействует с пользователем для сохранения/удаления рецептов в подборки.

**Реализация паттернов проектирования**



Рисунок 6 - Observer

## **Описание реализованных паттернов проектирования**

### 1. **Observer (Наблюдатель)** – поведенческий паттерн

**Применение в проекте CookGuide:**  
Используется для оповещения подписчиков (пользователей) о добавлении новых рецептов. В классической архитектуре MVVM это аналог LiveData, где ViewModel уведомляет View (фрагмент) об изменениях.

**Возможные сценарии:**

* Пользователь получает уведомление о новом рецепте;
* UI обновляется автоматически при изменении данных;
* Уведомления рассылаются всем подписанным наблюдателям.

**Преимущества:**

* Разделение логики уведомлений и интерфейса;
* Упрощённое обновление UI;
* Безопасность: избегаются утечки памяти (как у LiveData в Android).
* 
* Рисунок 7 – паттерн Singleton

### 2. **Singleton (Одиночка)** – порождающий паттерн

**Применение в проекте CookGuide:**  
Используется для единственного объекта менеджера рецептов (RecipeManager). Обеспечивает единый доступ к данным из разных частей приложения.

**Возможные сценарии:**

* Загрузка всех рецептов при старте приложения;
* Централизованное хранилище или кеш рецептов.

**Преимущества:**

* Гарантируется единый экземпляр;
* Централизация логики;
* Экономия ресурсов (например, при инициализации сервисов или API).

Реализация паттерна Decorator

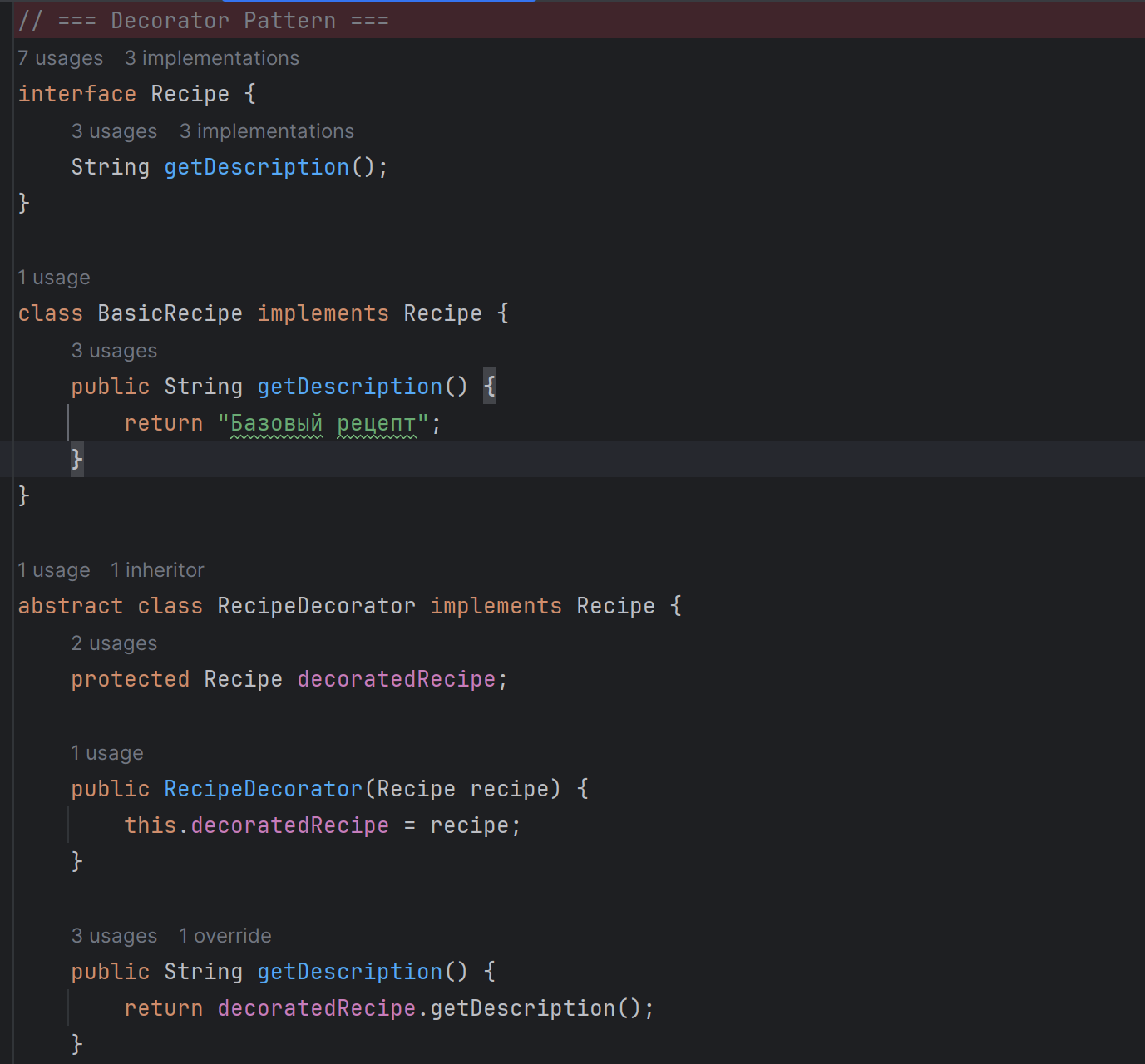
* 

Рисунок 8 – паттерн decorator

### 3. **Decorator (Декоратор)** – структурный паттерн

**Применение в проекте CookGuide:**  
Позволяет "обернуть" рецепт дополнительным описанием (например, пометкой «вегетарианский», «без глютена» и т.д.).

**Возможные сценарии:**

* Динамическое добавление признаков рецепта без изменения основного класса;
* Расширение функциональности без создания подклассов.

**Преимущества:**

* Гибкость и масштабируемость;
* Сохранение открытости к расширению без модификации базового класса;
* Удобно использовать при настройке отображения или фильтрации.

## **Паттерны, которые нецелесообразно использовать в проекте**

### 1. **Abstract Factory** – порождающий паттерн В проекте не требуется создавать семейства связанных объектов. Все рецепты представлены одним типом объекта (Recipe), и для его создания достаточно обычной фабрики. Abstract Factory привнёс бы избыточные интерфейсы и усложнил архитектуру без явной пользы.

### 2 Bridge (структурный)

Потому что в приложении нет необходимости абстрагировать реализацию рецепта от интерфейса

3.**Chain of Responsibility** – поведенческий паттерн  
Процессы, такие как загрузка рецепта или добавление изображения, выполняются линейно и централизованно. Нет необходимости в наборе обработчиков, каждый из которых может обработать или передать запрос. Весь контроль логики происходит в одном месте — это делает цепочку ненужной.

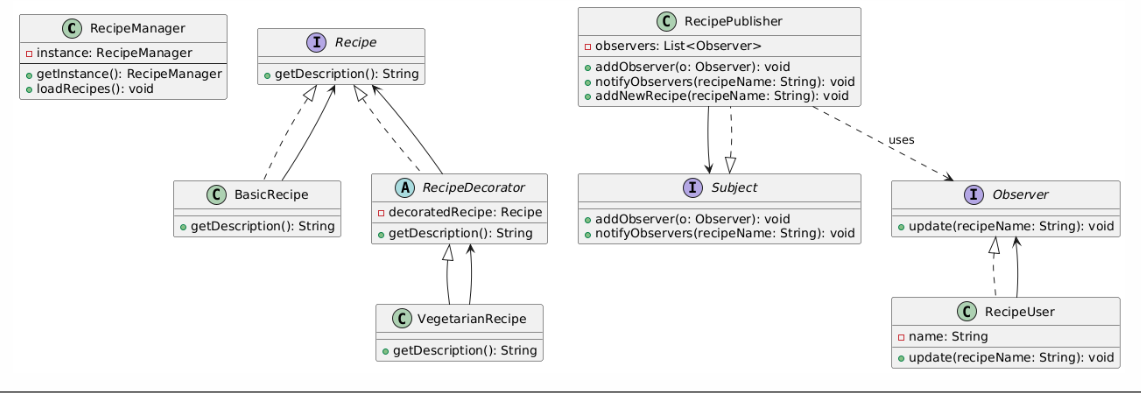


Рисунок 9 – UML диаграмма для реализованных классов

Вывод

В ходе выполнения данной практической работы была придумана и описана разрабатываемая в ходе семестра программная система, создана диаграмма вариантов использования системы, были составлены и задокументированы бизнес-требования, требования пользователей, функциональные и нефункциональные требования, выбран архитектурный паттерн и обосновано его применение, была составлена карта контекстов по методологии DDD и диаграммы по модели C4, были выбраны и реализованы паттерны проектирования.