

Discord-бот Dolix

Архитектурный документ

Авторы документа:

Абрамов Александр Сергеевич (БПИ213)

Преподаватель группы:

Мицюк Алексей Александрович

0. Раздел регистрации изменений

Версия документа	Дата изменения	Описание изменения	Автор изменения
1.0.0	26.02.2024	Создание документа	Абрамов Александр

1. Введение

1.1. Название проекта

Многофункциональный Discord-бот «Dolix»

1.2. Задействованные архитектурные представления

Настоящий документ содержит следующие архитектурные представления:

1. Представление прецедентов, показывающее основные услуги системы и ключевых акторов, взаимодействующих с системой при оказании этих услуг.
2. Логическое представление, описывающее проектное устройство системы, её основные классы и их разделение на пакеты и подсистемы.
3. Представление архитектуры процессов, показывающее ключевые аспекты взаимодействия классов и акторов системы во время её работы.
4. Представление развёртывания, демонстрирующее распределение подсистем и пакетов по физическим и логическим вычислительным узлам.
5. Представление архитектуры данных, описывающее принципы получения, обработки и хранения данных системой, позволяющие обеспечить необходимый уровень надёжности системы с точки зрения работы с данными.
6. Представление архитектуры безопасности, описывающее ключевые способы авторизации запросов, обеспечения безопасности системы и защиты данных пользователей, что является важными требованиями к разрабатываемому продукту.
7. Представление реализации и разработки, дающее рекомендации по организации процессов разработки и развёртывания системы.
8. Атрибуты качества системы, представляющие набор качественных и количественных характеристик, позволяющих оценить, насколько разработанная система удовлетворяет установленным нефункциональным требованиям.
9. Спецификации прецедентов первого этапа проекта, подробно описывающие актёров, потоки, условия, требования, ограничения и атрибуты качества услуг системы, реализуемых в рамках первого этапа проекта.

Физическое представление архитектуры не включено в настоящий документ, так как для развёртывания системы будут применяться облачные технологии, структура которых описана в представлении развёртывания.

Представление производительности не задействовано в настоящем документе, так как специальных требований к производительности системы не предъявляется.

Структура документа представлена в содержании:

0. Раздел регистрации изменений.....	2
1. Введение.....	3
1.1. Название проекта.....	3
1.2. Задействованные архитектурные представления.....	3
1.3. Контекст задачи и среда функционирования системы.....	5
1.4. Рамки и цели проекта.....	6
2. Архитектурные факторы.....	8
2.1. Ключевые заинтересованные лица.....	8
2.2. Ключевые требования к системе.....	8
2.3. Ключевые ограничения.....	9
3. Общее архитектурное решение.....	11
3.1. Принципы проектирования.....	11
4. Архитектурные представления.....	13
4.1. Представление прецедентов.....	13
4.2. Логическое представление.....	13
4.3. Представление архитектуры процессов.....	13
4.4. Представление развертывания.....	13
4.5. Представление архитектуры данных.....	13
4.6. Представление архитектуры безопасности.....	14
4.7. Представление реализации и разработки.....	14
4.8. Атрибуты качества системы.....	15
4.8.1. Объем данных и производительность системы.....	16
4.8.2. Гарантии качества работы системы.....	16
4.9. Спецификации прецедентов первого этапа проекта.....	16
4.9.1. Спецификация прецедента “настройка приложения”.....	16
4.9.2. Спецификация прецедента “покупка монет за фиат”.....	21
5. Технические описания отдельных ключевых архитектурных решений.....	25
5.1. Техническое решение №1: микросервисная архитектура.....	25
5.1.1. Проблема.....	25
5.1.2. Идея решения.....	25
5.1.3. Факторы.....	25
5.1.4. Решение.....	25
5.1.5. Мотивировка.....	26
5.1.6. Неразрешенные вопросы.....	26
5.1.7. Альтернативы.....	26

5.2. Техническое решение №2: сервис взаимодействия с Discord.....	26
5.2.1. Проблема.....	26
5.2.2. Идея решения.....	26
5.2.3. Факторы.....	26
5.2.4. Решение.....	26
5.2.5. Мотивировка.....	27
5.2.6. Неразрешенные вопросы.....	27
5.2.7. Альтернативы.....	27
5.3. Техническое решение №3: использование реляционных баз данных.....	28
5.3.1. Проблема.....	28
5.3.2. Идея решения.....	28
5.3.3. Факторы.....	28
5.3.4. Решение.....	28
5.3.5. Мотивировка.....	28
5.3.6. Неразрешенные вопросы.....	28
5.3.7. Альтернативы.....	29
5.4. Техническое решение №4 : авторизация пользователей через Discord.....	29
5.4.1. Проблема.....	29
5.4.2. Идея решения.....	29
5.4.3. Факторы.....	29
5.4.4. Решение.....	30
5.4.5. Мотивировка.....	30
5.4.6. Неразрешенные вопросы.....	30
5.4.7. Альтернативы.....	30
6. Приложения.....	31
6.1. Словарь терминов.....	31
6.2. Задействованные диаграммы.....	31

1.3. Контекст задачи и среда функционирования системы

Discord – одна из крупнейших социальных сетей на данный момент. Сервисом ежемесячно пользуются более 200 миллионов человек на более, чем 20 миллионах серверах. Большая часть серверов – крупные сообщества, объединяющие тысячи пользователей, которыми невозможно управлять без автоматизации основных процессов для повышения удобства как администраторов, так и участников.

К счастью, платформа предоставляет удобный API для создания ботов, которым многие сообщества успешно пользуются для решения различных задач. Тем не менее “порог входа” при создании новых сообществ достаточно высок: необходимо либо настраивать совместную работу большого количества ботов, выполняющих конкретные функции, либо заказывать разработку уникального приложения под свои потребности. Оба решения требуют большого количества как человеческих, так и финансовых ресурсов.

В результате анализа было замечено, что, на самом деле, большинство сообществ используют похожие функции, поэтому предлагается разработать универсальное решение, которое объединит их в одном боте и предоставит возможность легкой настройки под нужды конкретного сообщества. Модель предметной области, показывающая основные сущности, с которыми должна работать система, приведена на рисунке 1 приложения 2.

1.4. Рамки и цели проекта

Целью проекта является разработка универсального Discord-бота, объединяющего основные функции в одном приложении с возможностью их лёгкой, гибкой настройки под нужды конкретного сообщества в удобном веб-интерфейсе. Продукт позволит владельцам крупных сообществ избежать больших временных и финансовых вложений на настройку совместного взаимодействия нескольких ботов или разработку уникального приложения для решения основных задач управления сообществом. В результате проекта будет создана система, эффективно решающая основные задачи и требующая для работы лишь простой единоразовой настройки.

В рамках первого этапа проекта требуется разработать общую архитектуру системы и реализовать функции, позволяющие убедиться в её работоспособности:

1. Получение взаимодействий от Discord и их обработку;
2. Экономическую систему; в частности, покупку виртуальных монет за фиатную валюту;
3. Веб-приложение с авторизацией через Discord для доступа к панели управления;
4. Панель управления для настройки приложения в целом и модуля покупки монет в частности.

Система должна быть легко расширяемой и должна поддерживать добавление новых модулей без значительных изменений архитектуры в целом. В будущем планируется разработка следующих функций:

1. Обширная система экономики для повышения и поддержания активности участников: получение внутренней валюты за активность, возможность приобретения привилегий на сервере (ролей, кланов и др.) за заработанную валюту, игры на валюту между участниками сообщества, проведение мероприятий администраторами сообществ с монетами в качестве призов;
2. Покупка привилегий (в т.ч. ролей) на сервере за фиатную валюту с начислением вырученных средств на баланс сообщества после взимания небольшой комиссии;

3. Рынок ролей, позволяющий участникам сообществ покупать и продавать Discord-роли за виртуальную валюту;
4. Инвентарь ролей, с помощью которого участники могут “спрятать” одну или несколько ролей, которыми они владеют;
5. Модерация: принятие и обработка жалоб участников, выдача и просмотр предупреждений, наказаний и блокировок;
6. Автоматическое создание голосовых каналов для участников сервера по необходимости;
7. Автоматизация проведения розыгрышей среди участников сервера.

Более того, для монетизации приложения потребуется реализовать подсистему подписок на премиальные модули. Выбор желаемой подписки и создание платежа будут происходить в панели управления.

2. Архитектурные факторы

2.1. Ключевые заинтересованные лица

Действующее лицо	Заинтересованность в системе
Бизнес / Владелец	Извлечение прибыли, простота добавления новых модулей, полнота предоставляемых функций, высокая безопасность, отказоустойчивость
Discord	Соблюдение пользовательского соглашения и политики разработчиков, защита данных пользователей, работа в пределах установленных лимитов
Разработчик	Простота поддержки, возможность введения новой функциональности без значительных изменений архитектуры, прозрачность внутренних API
Тестировщик	Удобство верификации и валидации продукта, простота проведения регрессионного тестирования и тестирования новых функций
Поддержка пользователей	Простота диагностирования ошибок, возможность просмотра состояния и лог-файлов системы, а также данных пользователей при необходимости
Администраторы сообществ Discord	Автоматизация основных процессов управления сервером, повышение и поддержание активности участников, возможность гибкой настройки
Обычные участники сообществ Discord	Удобство пользования и понятность интерфейсов, простота взаимодействия, производительность, защищённость данных

2.2. Ключевые требования к системе

Ключевые требования к системе определяются контекстом и целями проекта:

1. Система должна реализовывать основные функции, используемые многими сообществами, с возможностью настройки большинства их параметров. Добавление новых функций должно быть возможно на любом этапе проекта в изоляции от существующих модулей, т.е. без внесения значительных изменений в них. В рамках первого этапа проекта требуется реализовать одну функцию – покупку виртуальной валюты за фиат – для проверки работоспособности архитектуры системы.
 - a. Администраторы сообществ в панели управления должны иметь возможность настройки стоимости монет;
 - b. Участники сообществ должны иметь возможность использования команды “buycoints” в клиенте Discord с указанием желаемого количества монет. При этом система должна создать платёж и прислать ссылку на страницу созданного платежа в качестве ответа;

- c. После успешной оплаты система должна начислить на баланс пользователя указанное количество монет, а также увеличить баланс сообщества на долю размера платежа, задаваемую в файле конфигурации;
2. Система должна предоставлять возможность настройки большинства параметров всех функций в удобном веб-интерфейсе – в панели управления, которая доступна только авторизованным пользователям, являющимся администраторами настраиваемых сообществ. Настройка объектов, связанных с “сообществом” методом один-ко-многим, должна производиться с помощью таблиц с возможностью добавления, редактирования и удаления записей.

2.3. Ключевые ограничения

1. Веб-приложение должно иметь возможность изменения стиля и языка интерфейса. Добавление тем и локализаций должно быть возможно на любом этапе проекта без изменения архитектуры системы. На первом этапе проекта должны быть реализованы два стиля – светлый и тёмный – и две локализации – английская и русская.
2. Веб-приложение должно корректно работать во всех современных браузерах.
3. Система должна запрашивать у пользователя двойное подтверждение любых необратимых изменений. Обо всех изменениях объектов “сообщество” должны быть отправлены уведомления в соответствующие системные каналы.
4. При проведении платежей пользователь должен иметь возможность выбора способа оплаты. Добавление новых способов оплаты должно быть возможно на любом этапе проекта без значительного изменения архитектуры системы. В рамках первого этапа проекта система должна работать с платёжными системами Yandex Pay и PayPal.
5. Система должна обеспечивать высокий уровень надёжности и безопасности:
 - a. Программа должна обеспечивать проверку корректности полученных от пользователей данных;
 - b. Приложение не должно аварийно завершаться при любом наборе входных данных. В случае отказа программы, не связанного с техническими неполадками сервера, время восстановления системы не должно превышать 30 минут;
 - c. Программа должна производить обработку возникающих ошибок и отображать пользователю понятный интерфейс с сообщениями о неполадках;
 - d. Система не должна допускать несанкционированный доступ к функциям и командам. Система должна обеспечивать высокий уровень защиты персональных данных пользователей.
6. Система должна соблюдать пользовательское соглашение и политику разработчиков Discord, включая:
 - a. Система не должна намеренно превышать лимиты запросов к API: должны быть разработаны механизмы задержки или отклонения запросов, которые заранее превысят установленные ограничения;

- b. Система не должна получать доступ к данным пользователей, не требующимся для непосредственного оказания её услуг. Система не должна хранить данные пользователей Discord, если это не является необходимым для её работы, а при необходимости система должна обеспечивать высокий коммерчески-обоснованный уровень защиты хранимых данных;
 - c. Для получения взаимодействий и других данных система должна поддерживать параллелизацию соединения с Discord с помощью шардирования в соответствии с документацией API.
7. Система должна быть реализована на языке JavaScript в среде Node.JS. Для реализации веб-сервера должен быть использован фреймворк Fastify и шаблонизатор веб-страниц EJS. Клиентская часть должна быть реализована на языке разметки HTML, языке стилей CSS и языке программирования JavaScript. Хранение данных должно производиться в базах PostgreSQL и Redis. Для использования API Discord должна быть использована библиотека Discord.JS.

3. Общее архитектурное решение

Общее решение основано на микросервисной архитектуре со следующими идеями:

1. Система содержит нескольких ключевых сервисов: основная база данных, клиент Discord, сервис проведения платежей, основное веб-приложение и веб-приложение панели управления;
2. За реализацию каждой функции отвечает один или несколько сервисов-модулей, каждый из которых реализует строгий интерфейс, позволяющий эффективно интегрироваться в систему: набор методов чтения и редактирования настроек, а также метод обработки взаимодействий, получаемых Discord-клиентом от пользователя;
3. Ключевые сервисы реализуют общие алгоритмы, необходимые для работы многих функций, и фактически служат для связывания сервисов-модулей между собой и распределения запросов пользователей между ними;
4. Сервисы-модули в свою очередь могут иметь сколь угодно сложную архитектуру, которая не влияет на общее архитектурное решение системы, выходит за рамки настоящего документа и должна разрабатываться отдельно;
5. Помимо прочего, каждый сервис, являющийся веб-приложением, реализует заданный набор методов, позволяющих пользователю изменять стиль и язык интерфейса в соответствии с требованиями.

Описанная архитектура позволяет без труда добавлять новые сервисы-модули, которые интегрируются в систему исключительно путём редактирования параметров конфигурации, что соответствует интересам всех заинтересованных лиц и одному из ключевых архитектурно-значимых требований к системе – расширяемости. Более того, микросервисная архитектура позволяет разделить задачи и данные по независимым, изолированным логическим узлам, что способствует обеспечению защищённости и сохранности данных, а разработка унифицированных протоколов взаимодействия сервисов позволяет обеспечить простоту и понятность пользовательских и программных интерфейсов.

3.1. Принципы проектирования

При проектировании основной целью стояло снижение связности сервисов и пакетов системы. В рамках архитектуры требовалось явно выделить ключевые высоконагруженные сервисы и минимизировать их количество, а сервисы, непосредственно отвечающие за исполнение бизнес-логики приложения, было необходимо как можно лучше изолировать от остальной системы для обеспечения их высокой сплочённости и слабой связности. Более того, немаловажным считалось ослабление связей между сервисами и их реализация посредством общих интерфейсов и файлов конфигурации, что способствует расширяемости системы и позволяет добавлять и редактировать сервисы без существенного влияния на остальную систему.

Также, для повышения безопасности системы все её взаимодействия с “внешним миром” (пользователями, API Discord и др.) по возможности проектировались посредством небольшого набора сервисов, отвечающих исключительно за это. Помимо прочего, такая архитектура способствует унификации интерфейсов системы и, как следствие, повышению удобства пользования ею.

4. Архитектурные представления

4.1. Представление прецедентов

Основные прецеденты использования системы и задействованные в них акторы описываются диаграммой прецедентов, представленной на рисунке 2 приложения 2.

4.2. Логическое представление

Устройство системы представлено диаграммами классов и пакетов, изображенными на рисунках 3 – 4 приложения 2.

4.3. Представление архитектуры процессов

Основные процессы, выполняемые всеми частями системы, представлены на диаграммах последовательности (рис. 5 – 16 прил. 2). Специальная проработка организации процессов в системе не требуется.

4.4. Представление развертывания

Специальных действий для развёртвания клиентской части системы не требуется: продукт доступен посредством браузера и приложения Discord.

Для запуска серверной части системы необходимо разместить все программные модули – исходные коды (js, html, css, ejs, sql) и файлы конфигурации (json) – на физических или логических узлах в соответствии с диаграммой развёртывания, приведённой на рисунке 17 приложения 2.

Автоматическое обновление системы не предусмотрено. Обновление системы должно происходить не чаще, чем 1 раз в сутки, в ручном режиме по графику, устанавливаемому руководителем проекта с учётом трендов суточной динамики нагрузки на систему.

4.5. Представление архитектуры данных

Большую часть данных, получаемых и обрабатываемых системой, составляют объекты API Discord в соответствии с документацией, доступной по адресу <https://discord.com/developers/docs>. За их получение отвечает Discord-клиент, передающий данные в соответствии с параметрами конфигурации всем или некоторым сервисам-модулям, которые производят их обработку и возвращают ответ в формате Discord-сообщения. Более того, сервисы-модули могут вызывать Discord-клиент для совершения каких-либо действий (выдачи или снятия ролей, управления каналами и др.) независимо от обрабатываемых событий.

Для хранения информации, если это необходимо, должны использоваться реляционные базы данных PostgreSQL. При этом должна быть реализована одна общая БД, содержащая основную информацию, необходимую многим сервисам, а также отдельные независимые БД

некоторых модулей, содержащие информацию, обрабатываемую только этими модулями. Для обеспечения сохранности данных в случае внештатных ситуаций должно регулярно производиться резервное копирование всех баз данных.

Гонок данных, требующих особого внимания, в системе не ожидается. Все случаи, когда возможно их возникновение, будут автоматически обрабатываться системой баз данных или выбранным для реализации проекта языком программирования. Специальных алгоритмов управления гонками данных разрабатывать не требуется.

4.6. Представление архитектуры безопасности

Один из ключевых аспектов разрабатываемой системы – безопасность данных пользователей, для обеспечения которой при разработке необходимо следовать ряду принципов:

1. Доступ к веб-приложению, за исключением одной – лендинговой – страницы, должен быть доступен только пользователям, прошедшим авторизацию через Discord по протоколу OAuth2.
2. Все взаимодействия с веб-приложениями должны происходить исключительно по протоколу HTTPS с использованием SSL-сертификата защищённого соединения.
3. Информация обо всех ключевых изменениях данных должна сохраняться в системе и доводиться до администраторов соответствующих сообществ посредством отправки уведомлений в указанные системные каналы.
4. Система должна собирать и обрабатывать исключительно данные, непосредственно необходимые для её работы. Система в целом и Discord-клиент в частности не должны запрашивать доступ к сообществам и аккаунтам пользователей, который им не требуется для оказания заявленных услуг.
5. Наиболее чувствительные данные (токены, коды доступа и др.) должны храниться в зашифрованном виде без возможности их расшифровки даже в случае утечки.
6. Обращения пользователей, связанные с вопросами безопасности, должны обрабатываться в первую очередь. Любые обнаруженные уязвимости должны устраняться немедленно; соответствующие задачи должны иметь приоритет над любыми другими задачами.

4.7. Представление реализации и разработки

При выполнении проекта допускается использование любых инструментов и сред разработки на усмотрение исполнителя. Рекомендованная среда разработки – Visual Studio Code.

Для организации процесса разработки возможно использование любого сервиса на усмотрение руководителя проекта. Рекомендуется использование системы Yandex Tracker.

Для хранения исходного кода и удобной работы с ним должна быть использована система контроля версий Git и репозиторий <https://github.com/TTPO100AJIEX/DOLIX>. Для получения доступа следует обращаться к руководителю проекта. При внесении изменений в исходный код разработчик должен создать в репозитории новую ветку и вести разработку в ней. По окончании

реализации новая версия исходного кода должна пройти все существующие процессы тестирования. Только в этом случае она может быть соединена с основной версией.

Развёртывание актуальной версии должно происходить регулярно, но не чаще, чем 1 раз в сутки в штатном режиме. Точный график “релизов” не регламентируется настоящим документом и должен быть установлен руководителем проекта с учётом трендов суточной динамики нагрузки на систему.

4.8. Атрибуты качества системы

Наиболее важными для системы считаются следующие атрибуты качества:

1. Модифицируемость и распределаемость процесса разработки – система должна поддерживать добавление новых модулей без значительных изменений архитектуры в целом. Работу над независимыми модулями должно быть возможно производить параллельно.
2. Защищённость – система должна обеспечивать высокий уровень защиты данных пользователей. Возможность несанкционированного доступа к базам и хранилищам должна быть исключена; утечка данных клиентов должна быть невозможна. Собираться и обрабатываться должны только данные, которые непосредственно необходимы для выполнения системой заявленных функций.
3. Надёжность – программа не должна аварийно завершаться при любом наборе входных данных, а также должна производить обработку возникающих ошибок и отображать пользователю понятный интерфейс с сообщениями о неполадках.
4. Простота использования – система должна иметь интерфейс, понятный пользователю, обладающему базовыми навыками работы с компьютером или смартфоном. Приложение не должно совершать “неожиданных” операций, а необратимые действия должны требовать двойного подтверждения от пользователя.
5. Доступность – система не должна аварийно завершаться при любых действиях пользователей. В случае возникновения любых ошибок система должна сообщить пользователю о неполадках, но корректно продолжить работу.
6. Переносимость – взаимодействие с Discord-клиентом должно быть возможно на любом устройстве, соответствующем требованиям к оборудованию приложения Discord; интерфейс веб-приложения должен корректно работать в следующих браузерах:
 - a. Google Chrome версии 119 или выше;
 - b. Microsoft Edge версии 119 или выше;
 - c. Mozilla Firefox версии 121 или выше;
 - d. Опера версии 106 или выше;
 - e. Safari версии 16.4 или выше.

4.8.1. Объем данных и производительность системы

Объём обрабатываемых данных зависит от количества сообществ, использующих продукт. Ключевые сервисы системы, включая Discord-клиент и веб-приложения, должны успешно обрабатывать до 100 взаимодействий в минуту с каждого сервера, а также должны быть легко масштабируемы на любое количество пользователей. Большой нагрузки на сервисы-модули не ожидается.

Особенных требований к производительности системы не предъявляется. Программа должна отвечать на любой запрос не более, чем за 0.5 секунды без учёта затрат на взаимодействие со сторонними сервисами, а отображение ответа на экране пользователя не должно занимать более 3-х секунд.

4.8.2. Гарантии качества работы системы

Для обеспечения качества системы должны быть реализованы механизмы тестирования и мониторинга:

1. Для каждого сервиса должна быть разработана программа и методика испытаний, а исходный код должен быть покрыт интеграционными и юнит-тестами для удобства проведения и автоматизации регрессионного тестирования. Внесение изменений в исходный код не должно быть возможно при невыполнении новым исходным кодом хотя бы одного из установленных тестовых случаев.
2. Для наблюдения за состоянием системы во время её работы должны быть реализованы механизмы мониторинга, позволяющие оценивать динамику и выявлять аномалии изменения нагрузки на все сервисы (количество запросов, время обработки запросов, утилизация ресурсов и др.) не менее, чем за последние 7 дней.

4.9. Спецификации прецедентов первого этапа проекта

4.9.1. Спецификация прецедента “настройка приложения”

Название ◇	Настройка приложения
Аннотация ◇	Одной из ключевых конкурентных особенностей продукта является возможность гибкой настройки всех функций под потребности конкретного сообщества, в связи с чем важно уделить особенное внимание разработке удобной для пользователей панели управления, которая бы предоставила возможность детальной настройки приложения при простоте и понятности своего интерфейса и принципов работы. Хотя настройку всех существующих параметров можно реализовать и на поздних этапах проекта, панель управления для настройки минимального

	набора основных параметров должна присутствовать уже на самых ранних этапах жизненного цикла.
Автор ◇	Абрамов Александр Сергеевич, программный инженер продукта, студент группы БПИ213
Рамки применения	Ко всей системе
Значимость	Ключевая задача
Приоритет	Высокий
Статус реализации	Проработан, не реализован
Первичный актёр ◇	Администратор сообщества
Вторичные актёры и их требования	<p>Discord:</p> <ol style="list-style-type: none"> 1. Авторизация пользователей - https://discord.com/developers/docs/topics/oauth2 2. Отправка лога изменений - https://discord.com/developers/docs/resources/channel#create-message
Базовый поток ◇	<ol style="list-style-type: none"> A. Пользователь заходит на сайт. B. Система формирует пользовательский интерфейс с общей информацией о продукте и кнопкой авторизации через Discord. C. Пользователь авторизуется через Discord по протоколу OAuth2. D. Система формирует пользовательский интерфейс с меню выбора сообщества для настройки. E. Пользователь выбирает сообщество, на котором хочет произвести настройку бота. F. Система убеждается, что пользователь действительно является администратором выбранного сообщества. G. Система формирует пользовательский интерфейс с описанием общих принципов использования панели управления и с меню навигации по ней. H. Пользователь переходит на страницу с настройками, которые хочет отредактировать. I. Система генерирует пользовательский интерфейс с полями ввода, необходимыми для настройки параметров соответствующих функций, и заполняет их текущими значениями. J. Пользователь редактирует значения всех или некоторых элементов. K. Пользователь нажимает кнопку сохранения изменений. L. Система запрашивает подтверждение, что пользователь

	<p>действительно хочет сохранить новые значения.</p> <p>M. Пользователь подтверждает намерение сохранить изменения нажатием соответствующей кнопки.</p> <p>N. Система проверяет корректность введённых значений. Значения корректны.</p> <p>O. Система производит сохранение новых значений.</p> <p>P. Система отправляет лог изменений в системный канал сообщества, если таковой задан, посредством запроса к API Discord.</p> <p>Q. Система сообщает пользователю об успешном выполнении операции.</p> <p>R. Пользователь завершает настройку приложения.</p>
Альтернативные потоки ◇	<p>Альтернативный поток АТ1. Условие начала: пользователь хочет прекратить работу.</p> <p>A. Пользователь покидает страницы панели управления.</p> <p>Альтернативный поток АТ2. Условие начала: пользователь нажимает кнопку изменения стиля интерфейса.</p> <p>A. Система формирует меню выбора стиля интерфейса.</p> <p>B. Пользователь выбирает желаемый стиль.</p> <p>C. Система вносит соответствующие корректировки в интерфейс.</p> <p>D. Продолжение основного потока.</p> <p>Альтернативный поток АТ3. Условие начала: пользователь нажимает кнопку изменения языка интерфейса.</p> <p>A. Система формирует меню выбора языка интерфейса.</p> <p>B. Пользователь выбирает желаемый язык.</p> <p>C. Система формирует интерфейс на выбранном языке.</p> <p>D. Продолжение основного потока.</p> <p>Альтернативный поток С1. Условие начала: на шаге С нет связи с Discord.</p> <p>A. Система переходит в аварийный режим и формирует соответствующий пользовательский интерфейс с сообщением об ошибке.</p> <p>Альтернативный поток F1. Условие начала: пользователь не является администратором выбранного сообщества.</p> <p>A. Система формирует пользовательский интерфейс с сообщением об ошибке.</p> <p>B. Переход на шаг D основного потока.</p> <p>Альтернативный поток G1. Условие начала: пользователь хочет настроить шаблоны сообщений, доступные для использования в настройках.</p>

- A. Пользователь переходит на страницу управления шаблонами сообщений.
- B. Переход к прецеденту “Создание шаблонов сообщений”.

Альтернативный поток G2. Условие начала: пользователь хочет изменить сообщество, на котором производит настройку бота.

- A. Пользователь нажимает кнопку изменения сообщества.
- B. Переход на шаг D основного потока.

Альтернативный поток I1. Условие начала: пользователь хочет перейти на другую страницу панели управления.

- A. Переход на шаг H основного потока.

Альтернативный поток I2. Условие начала: пользователь хочет изменить сообщество, на котором он производит настройку бота.

- A. Пользователь нажимает кнопку изменения сообщества.
- B. Переход на шаг D основного потока.

Альтернативный поток J1. Условие начала: на шаге J пользователь редактирует поле ввода строки добавления элементов в таблицу.

- A. Система создаёт кнопку удаления отредактированной строки.
- B. Система создаёт новую строку с пустыми полями ввода для дальнейшего добавления элементов в таблицу.
- C. Переход на шаг J основного потока.

Альтернативный поток J2. Условие начала: на шаге J пользователь нажимает кнопку удаления строки таблицы.

- A. Система удаляет все элементы, связанные с соответствующей строкой.
- B. Переход на шаг J основного потока.

Альтернативный поток L1. Условие начала: пользователь отменяет сохранение изменений нажатием соответствующей кнопки.

- A. Переход на шаг J основного потока.

Альтернативный поток N1. Условие начала: на шаге N введённые значения некорректны.

- A. Система формирует пользовательский интерфейс с сообщением об ошибке.
- B. Переход на шаг J основного потока.

Альтернативный поток Q1. Условие начала: пользователь хочет продолжить работу.

- A. Переход на шаг I основного потока.

Предусловие ◇

Пользователь является администратором сообщества, на котором хочет произвести настройку приложения, и бот добавлен на соответствующий сервер Discord.

Постусловия ◇	<ol style="list-style-type: none"> В сообществе установлены новые настройки приложения, удовлетворяющие всем требованиям и ограничениям. Лог изменений отправлен в системный канал сообщества, если он задан.
Специальные бизнес-требования	<ol style="list-style-type: none"> На шаге С основного потока система не должна запрашивать больше доступа к аккаунту пользователя, чем ей необходимо для работы. Система должна обеспечивать высокий уровень надёжности. Интерфейс должен быть прост, удобен и понятен. Интерфейс должен быть реализован в двух стилях (светлый, тёмный) на двух языках (русский, английский). Добавление стилей и языков должно быть возможно на любом этапе жизненного цикла проекта без изменения архитектуры системы. Внесённые пользователем на шаге J основного потока изменения не должны “пропадать” при переходе к альтернативным потокам L1 и N1.
Связь с атрибутами качества	<ol style="list-style-type: none"> Система не должна допускать несанкционированный доступ к панели управления. Интерфейс должен быть понятен конечному пользователю, обладающему базовыми навыками обращения с компьютером. Интерфейс должен быть снабжен достаточным для понимания типов и ограничений ожидаемых значений, а также для интерпретации возникающих ошибок количеством документации и подсказок.
Проектные ограничения	Панель управления для настройки минимального набора основных параметров должна быть реализована на самых ранних этапах жизненного цикла. Поддержка других существующих параметров может быть реализована позже.
Список технологий реализации	<p>Серверная часть:</p> <ol style="list-style-type: none"> Язык программирования Node.JS; Фреймворк Fastify; База данных PostgreSQL, библиотека pg; Ханилище данных Redis, библиотека ioredis; Шаблонизатор веб-страниц EJS; <p>Клиентская часть:</p> <ol style="list-style-type: none"> Язык разметки HTML; Язык стилей CSS; Язык программирования JavaScript; <p>Клиент-серверное и межсервисное взаимодействие:</p> <ol style="list-style-type: none"> Протокол HTTPS;

	2. Протокол OAuth2;
Открытые проблемы	Отсутствуют

4.9.2. Спецификация прецедента “покупка монет за фиат”

Название ◇	Покупка монет за фиат
Аннотация ◇	Одним из основных способов монетизации приложения будет взимание комиссии с продажи виртуальной валюты сообществами. Для этого необходимо реализовать удобную, “прозрачную” автоматизированную систему, в рамках которой пользователи смогут покупать монеты за фиатную валюту. При этом большая часть средств перечисляется на баланс сообщества, которым затем можно оплатить подписку на премиальные функции.
Автор ◇	Абрамов Александр Сергеевич, программный инженер продукта, студент группы БПИ213
Рамки применения	Ко всей системе
Значимость	Ключевая задача
Приоритет	Высокий
Статус реализации	Проработан, не реализован
Первичный актёр ◇	Участник сообщества
Вторичные актёры и их требования	<ol style="list-style-type: none"> 1. Discord - средство взаимодействия с пользователем, https://discord.com/developers/docs 2. Платёжные системы - обработка платежей. <ol style="list-style-type: none"> a. Yandex Pay - https://pay.yandex.ru/business b. PayPal - https://www.paypal.com/ru/webapps/mpp/partners-and-developers
Базовый поток ◇	<ol style="list-style-type: none"> A. Пользователь использует команду покупки монет в любом текстовом канале сервера, указывая количество монет к покупке. B. Бот вычисляет стоимость указанного количества монет с учётом настроек сообщества. C. Бот создаёт платёж на необходимую сумму и получает его уникальный номер. D. Бот отвечает на команду сообщением, в котором

	<p>подтверждается информация о платеже, с кнопкой, которая направляет пользователя на страницу платежа на сайте.</p> <p>E. Пользователь переходит по кнопке.</p> <p>F. Система формирует пользовательский интерфейс страницы с данными платежа.</p> <p>G. Пользователь проверяет, что вся информация верна.</p> <p>H. Пользователь выбирает способ оплаты.</p> <p>I. Система создаёт платёжную форму с помощью API выбранного способа оплаты и перенаправляет пользователя на неё.</p> <p>J. Пользователь переходит на страницу оплаты.</p> <p>K. Пользователь осуществляет оплату в стороннем сервисе.</p> <p>L. Система проверяет статус платежа с помощью API выбранного способа оплаты. Оплата произведена успешно.</p> <p>M. Система начисляет монеты на баланс пользователя.</p> <p>N. Система увеличивает баланс сообщества на долю размера платежа, задаваемую в файле конфигурации.</p> <p>O. Система устанавливает платежу статус "Processed".</p> <p>P. Система отправляет уведомление о покупке в системный канал сообщества, если таковой задан, посредством запроса к API Discord.</p>
Альтернативные потоки ◇	<p>Альтернативный поток G1. Условие начала: на шаге G пользователь обнаружил ошибку в информации о платеже.</p> <p>A. Переход на шаг A основного потока.</p> <p>Альтернативный поток I1. Условие начала: на шаге I нет связи с платёжной системой.</p> <p>A. Система переходит в аварийный режим и формирует соответствующий пользовательский интерфейс с сообщением об ошибке.</p> <p>Альтернативный поток L1. Условие начала: на шаге L получен статус, указывающий, что оплата еще не была произведена, или нет связи с платёжной системой.</p> <p>A. Перейти на шаг L.</p> <p>Альтернативный поток L2. Условие начала: на шаге L получен статус, указывающий, что оплата не была произведена успешно (платёж отклонён).</p> <p>A. Система устанавливает платежу статус "Rejected".</p>
Предусловие ◇	Бот добавлен в сообщество, функция экономики активна, и пользователь может использовать команду покупки монет на необходимом сервере.
Постусловия ◇	1. На виртуальный счёт пользователя начислено приобретённое

	<p>количество монет.</p> <ol style="list-style-type: none"> 2. На баланс сообщества начислено соответствующее количество средств. 3. Уведомление о покупке отправлено в системный канал сообщества, если он задан.
Специальные бизнес-требования	<ol style="list-style-type: none"> 1. Система не должна сохранять информацию, которая не является необходимой для её непосредственной работы, включая персональные данные пользователей и реквизиты использованных способов оплаты. 2. Система должна обеспечивать высокий уровень надёжности. 3. Интерфейс страницы платежа должен быть прост, удобен и понятен. 4. Для повышения доверия пользователей к системе обработка платежа после его оплаты должна производиться быстро.
Связь с атрибутами качества	<ol style="list-style-type: none"> 1. Система не должна допускать “накрутку” монет в обход оплаты. 2. Интерфейс страницы платежа должен быть понятен конечному пользователю, обладающему базовыми навыками обращения с компьютером. 3. Обработка платежа - начисление монет на баланс пользователя - при отсутствии технических неполадок сервера или платёжной системы не должно занимать более 5 минут после оплаты счёта пользователем на шаге К основного потока.
Проектные ограничения	<p>На начальном этапе проекта должна быть реализована поддержка только платёжных систем Yandex Pay и PayPal. В будущем список может расширяться; добавление новых сервисов приёма платежей не должно требовать изменения архитектуры системы.</p>
Список технологий реализации	<p>Бот (Discord-клиент):</p> <ol style="list-style-type: none"> 1. Язык программирования Node.JS; 2. Библиотека Discord.JS; <p>Серверная часть:</p> <ol style="list-style-type: none"> 1. Язык программирования Node.JS; 2. Фреймворк Fastify; 3. База данных PostgreSQL, библиотека pg; 4. Шаблонизатор веб-страниц EJS; <p>Клиентская часть:</p> <ol style="list-style-type: none"> 1. Язык разметки HTML; 2. Язык стилей CSS; 3. Язык программирования JavaScript; <p>Клиент-серверное и межсервисное взаимодействие:</p> <ol style="list-style-type: none"> 1. Протокол HTTPS;

Открытые
проблемы

Отсутствуют

5. Технические описания отдельных ключевых архитектурных решений

5.1. Техническое решение №1: микросервисная архитектура

5.1.1. Проблема

Система должна быть расширяемой и должна способствовать добавлению модулей с похожими интерфейсами на любом этапе проекта.

5.1.2. Идея решения

Следует использовать микросервисную архитектуру, в которой модули реализуются в виде отдельных изолированных сервисов, взаимодействующих с остальной системой посредством небольшого набора ключевых сервисов.

5.1.3. Факторы

1. Бизнес-логика отдельных модулей должна быть изолирована от остальной системы. Добавление новых функций должно быть возможно на любом этапе проекта без внесения значительных изменений в архитектуру системы в целом. Должна быть возможность параллельной работы над разными модулями.
2. Система должна обеспечивать высокий уровень надёжности. Программа не должна аварийно завершаться при любом наборе входных данных, а при возникновении ошибок должна корректно их обрабатывать и отображать пользователю понятный интерфейс с сообщениями о неполадках.
3. Хотя специальных требований к производительности не предъявляется, обработка запросов должна быть достаточно эффективной для удовлетворения требованиям п. 4.8.1.

5.1.4. Решение

Общее архитектурное решение подробно описано в п. 3 настоящего документа и заключается в использовании микросервисной архитектуры с выделением двух типов сервисов:

1. Ключевые сервисы, отвечающие за получение запросов от пользователей и организацию их обработки соответствующими модулями;
2. Шаблонизированные сервисы-модули, непосредственно реализующие бизнес-логику приложения – обработку запросов, получаемых от ключевых сервисов – и следующие набору строгих интерфейсов для успешной интеграции в систему.

Для взаимодействия микросервисов должен использоваться GRPC, который реализует строгую типизацию на протокольном уровне и обеспечивает высокую производительность и надёжность соединений.

5.1.5. Мотивировка

Микросервисная архитектура – стандартный подход при наличии требований к расширяемости системы, который хорошо подходит для решения поставленной задачи. Использование высокопроизводительного, надёжного протокола взаимодействия микросервисов – GRPC – также способствует удовлетворению всех установленных требований. Подробное обоснование выбора представлено в п. 3 настоящего документа.

5.1.6. Неразрешенные вопросы

Одним из вопросов, которому потребуется уделить особое внимание во время разработки, является обеспечение надёжного и эффективного взаимодействия микросервисов. Необходимо оценить применимость GRPC как основного протокола взаимодействия микросервисов в системе и выбрать или разработать другой протокол, если это необходимо.

5.1.7. Альтернативы

Использование монолитной архитектуры с разделением на модули. Тем не менее такой подход снижает потенциал расширяемости системы, усложняет её развёртывание и масштабирование, хотя и незначительно упрощает архитектуру.

5.2. Техническое решение №2: сервис взаимодействия с Discord

5.2.1. Проблема

Система должна связываться с API Discord по протоколу WebSocket для взаимодействия с пользователями в приложении Discord.

5.2.2. Идея решения

Взаимодействие с Discord следует организовать посредством отдельного микросервиса, основной обязанностью которого является связь с Discord: получение событий и отправка ответов.

5.2.3. Факторы

1. Количество обрабатываемых событий может быть достаточно большим и будет неограниченно увеличиваться с ростом числа пользователей.
2. С увеличением количества клиентов потребуется параллелизация соединения с Discord посредством шардирования.
3. Система не должна допускать превышения установленных Discord лимитов запросов к API. Чрезмерные запросы должны задерживаться или отклоняться системой до отправки.

5.2.4. Решение

Для взаимодействия с Discord должен быть реализован отдельный микросервис, отвечающий за получение событий, их распределение по сервисам-модулям на основе

параметров конфигурации и отправку ответов. При этом сервис должен содержать набор точек доступа, дублирующих API Discord, для отправки запросов, не связанных с ответами на взаимодействия.

5.2.5. Мотивировка

Именно такое решение наилучшим образом подходит для решения поставленной задачи с учётом всех факторов. Альтернативные решения и их недостатки приведены в п. 5.2.7.

5.2.6. Неразрешенные вопросы

1. Ожидается, что количество событий, получаемых системой от Discord, линейно зависит от количества сообществ, использующих приложение, что может потребовать большого количества вычислительных ресурсов для успешной обработки всех запросов с ростом популярности продукта.
2. Следует следить за количеством отправляемых к API Discord запросов и не допускать превышения установленных лимитов. На первых этапах проекта с этой задачей справятся внутренние методы библиотеки Discord.JS, но по мере роста количества пользователей и распределения вычислений по разным логическим и физическим узлам может потребоваться разработка собственного решения.
3. Для успешного взаимодействия с описанным микросервисом потребуется фактически продублировать части достаточно обширного API Discord: реализовать методы, необходимые для выполнения основных операций, и описать схемы структур данных, запросов и ответов для получения взаимодействий, их распределения по сервисам-модулям и отправки ответов.

5.2.7. Альтернативы

1. Реализация всех взаимодействий с Discord и всей логики обработки событий в одном сервисе не потребует разработки большого количества точек доступа, фактически дублирующих API Discord, но у такого сервиса будет множество обязанностей и, как следствие, низкая сплочённость, что противоречит требованию расширяемости системы. Более того, такой подход реализует ещё большее количество логики в и так высоконагруженному Discord-клиенте, что может потребовать серьёзной проработки вопросов производительности, а также усложнит организацию распределённых вычислений путём шардирования, что неизбежно потребуется с увеличением количества пользователей.
2. Получение взаимодействий в сервисах-модулях напрямую от Discord путём открытия параллельных соединений из каждого сервиса. Такой подход также не потребует дублирования API Discord в точках доступа системы, но приведёт к снижению сплочённости сервисов-модулей, к большому количеству повторяющегося кода и к повышению нагрузки на всю систему, а не только на отдельный сервис. Более того, возможность организации нескольких параллельных соединений не представлена в документации API Discord, хотя и, по-видимому, работает в текущей версии интерфейса.

5.3. Техническое решение №3: использование реляционных баз данных

5.3.1. Проблема

Система должна обеспечивать высокую защищённость и надёжность хранилищ данных. Утеря данных или получение несанкционированного доступа к ним не должны быть возможны.

5.3.2. Идея решения

Следует использовать набор реляционных баз данных PostgreSQL, каждая из которых содержит определённую небольшую часть данных и служит для хранения информации конкретным набором сервисов.

5.3.3. Факторы

1. Система должна обеспечивать высокий уровень защиты данных пользователей в соответствии с требованиями п. 4.8.
2. По мере добавления модулей в систему может потребоваться хранение большого количества данных.
3. Хотя специальных требований к производительности и не предъявляется, обработка запросов должна быть достаточно эффективной для удовлетворения требованиям п. 4.8.1.

5.3.4. Решение

Для организации хранения информации должен быть использован набор реляционных баз данных в соответствии с представлениями архитектуры, описанными п. 4 настоящего документа: одно высоконагруженное хранилище, содержащее ключевую информацию о пользователях и сообществах, и некоторое число независимых баз данных для хранения информации отдельными модулями системы.

5.3.5. Мотивировка

1. Использование реляционных баз данных позволяет точно знать, какая информация хранится и обрабатывается в системе, что способствует упрощению процессов её разработки и поддержки.
2. Распределение информации по нескольким базам данных не только способствует обеспечению производительности, надёжности и отказоустойчивости отдельных хранилищ, но и повышает безопасность системы: компрометация одного хранилища не повлечет утечку всех данных пользователей.

5.3.6. Неразрешенные вопросы

При организации нескольких изолированных баз данных установление внешних ключей не представляется возможным, из-за чего потребуется уделить особенное внимание обеспечению

целостности информации путём организации внесения изменений в несколько БД одновременно или путём разработки запросов и логики особенным образом, не полагающимся на целостность данных.

5.3.7. Альтернативы

1. Разработка единой базы данных, содержащей всю информацию, с которой работает система. Такой подход позволит обеспечить целостность данных, но может привести к проблемам с производительностью и безопасностью.
 - a. Ожидается, что нагрузка на некоторые базы данных будет достаточно большой, из-за чего при объединении всех хранилищ в одно с увеличением числа пользователей БД может оказаться узким местом системы с точки зрения производительности и потребует масштабирования, что всегда является достаточно сложной задачей, столкновения с которой следует избегать при отсутствии крайней необходимости.
 - b. Хранение информации в одном месте приведёт к потере всех данных пользователей в случае утечки, если использованных мер защиты окажется недостаточно. Разделение хранилища на несколько баз данных усложняет работу злоумышленников и способствует обеспечению сохранности данных, хотя, безусловно, и не гарантирует её.
2. Использование нереляционных подходов к хранению информации (напр., MongoDB) позволит не описывать структуру данных для создания БД, что способствует ускорению процесса разработки, но усложняет поддержку системы и требует ещё более детальной проработки механизмов обеспечения целостности данных.

5.4. Техническое решение №4 : авторизация пользователей через Discord

5.4.1. Проблема

Доступ к панели управления должен быть возможен только администраторами сообществ после авторизации в веб-приложении.

5.4.2. Идея решения

Следует реализовать авторизацию через Discord по протоколу OAuth2.

5.4.3. Факторы

1. Возможность доступа к панели управления без авторизации или без необходимых прав должна быть исключена.
2. Вероятность утечки авторизационных данных пользователей (токенов, логинов, паролей и др.) должна быть сведена к минимуму.
3. Другие требования безопасности, указанные в настоящем документе, включая п. 4.6.

5.4.4. Решение

Идентификацию пользователей следует производить с помощью стороннего провайдера авторизации – Discord – по протоколу OAuth2 в соответствии с документацией, доступной по адресу <https://discord.com/developers/docs/topics/oauth2>. При авторизации система должна запрашивать минимальный необходимый для оказания услуг набор уровней доступа к аккаунту пользователя (OAuth2 Scopes). Система не должна запрашивать доступ, а также получать, обрабатывать и сохранять данные, не являющиеся необходимыми для её работы.

Для проверки наличия у пользователя доступа к выбранному сообществу на уровне “Администратор” система должна запрашивать данные пользователя и список его серверов с помощью точек доступа /users/@me и /users/@me/guilds, реализуемых API Discord. Соответствующая документация доступна по адресам <https://discord.com/developers/docs/resources/user#get-current-user> и <https://discord.com/developers/docs/resources/user#get-current-user-guilds>. В случае отсутствия у пользователя необходимого доступа система не должна допустить его к панели управления, а должна отобразить пользовательский интерфейс с сообщением об ошибке доступа.

5.4.5. Мотивировка

Выбранное решение фактически избавляет систему от ряда механизмов, связанных с процессом авторизации, перекладывая обязанности обработки персональных данных пользователей на внешнюю систему – Discord, что упрощает разработку и повышает безопасность системы, уменьшая количество конфиденциальных данных, которые необходимо сохранять и обрабатывать. Более того, использование авторизации через сторонний сервис способствует упрощению пользовательского пути, улучшению интерфейса и повышению доверия пользователей к системе.

5.4.6. Неразрешенные вопросы

Вопросов, требующих дополнительной проработки, не выявлено.

5.4.7. Альтернативы

Собственная авторизация с помощью логина и пароля с последующей привязкой аккаунта Discord даёт больше контроля над процессом авторизации и способствует повышению безопасности системы, если реализована корректно. Тем не менее вероятность ошибки, которая может привести к нарушению уровней доступа или утечке информации, при таком подходе значительно выше, чем при использовании стороннего провайдера авторизации. Более того, при использовании такого подхода пользователю всё равно потребуется привязать аккаунт Discord для получения системой информации о его ролях в сообществе, из-за чего ввод логина и пароля фактически оказывается лишней операцией, лишь усложняющей пользовательский путь.

6. Приложения

6.1. Словарь терминов

БД – (сокр.) база данных

Канал – (конт. Discord) текстовый или голосовой канал сервера Discord

Модуль – часть системы, непосредственно отвечающая за оказание какой-либо услуги, выполнение какой-либо функции. Каждый модуль описывается некоторой логикой ответов на взаимодействия, получаемые от Discord, набором настраиваемых параметров этой логики и, возможно, необходимостью хранения и обработки каких-либо данных

Монеты – виртуальная валюта, реализуемая модулями экономики

Панель управления – веб-приложение, позволяющее администраторам сообществ производить настройку параметров модулей в удобном веб-интерфейсе

Роль – (конт. Discord) роль сервера Discord, выдаваемая участникам сообщества администраторами

Сервер/сообщество – сущность предметной области, содержащая информацию о серверах Discord в контексте их взаимодействия с системой

6.2. Задействованные диаграммы

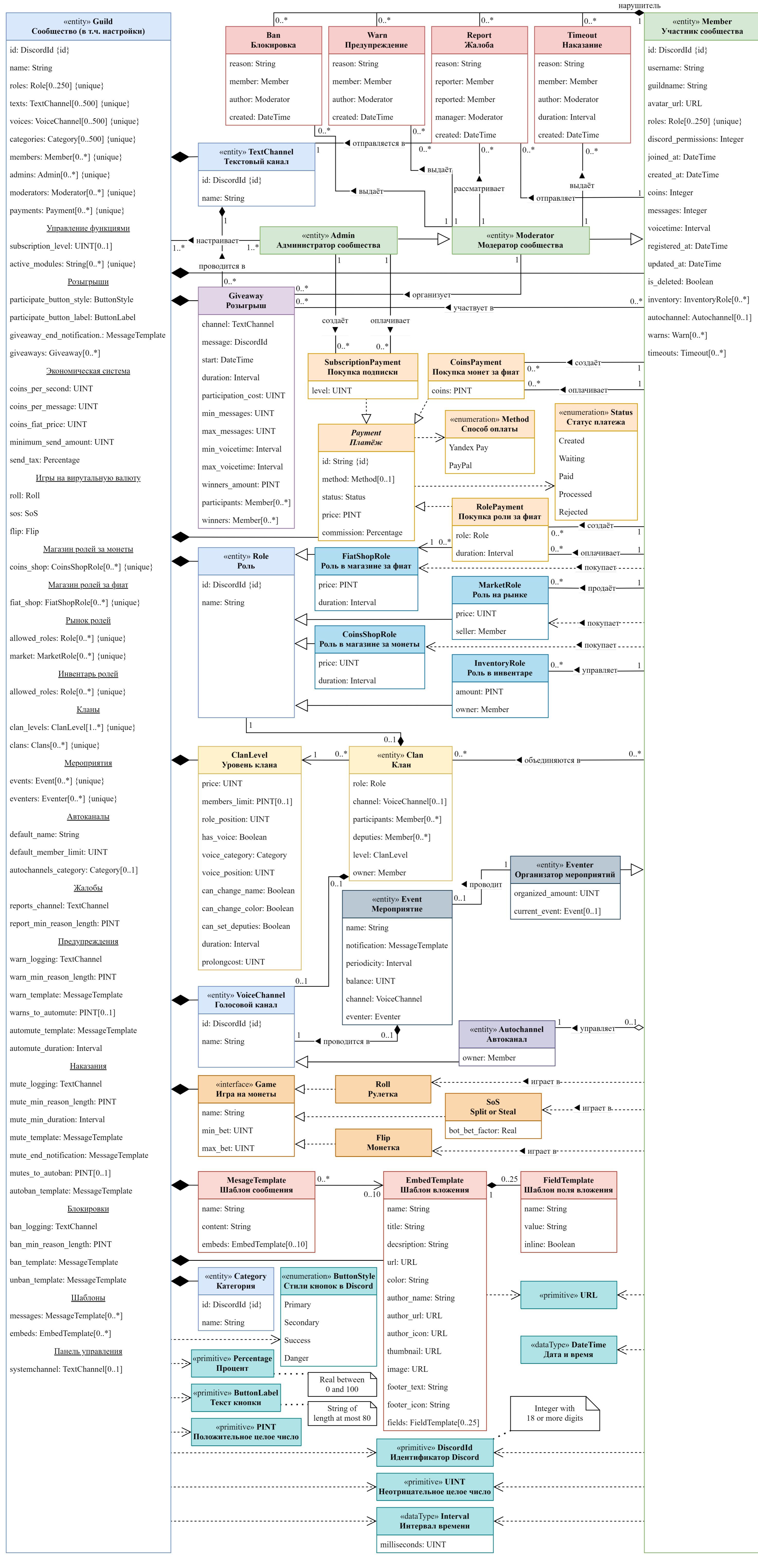


Рисунок 1 - Модель предметной области

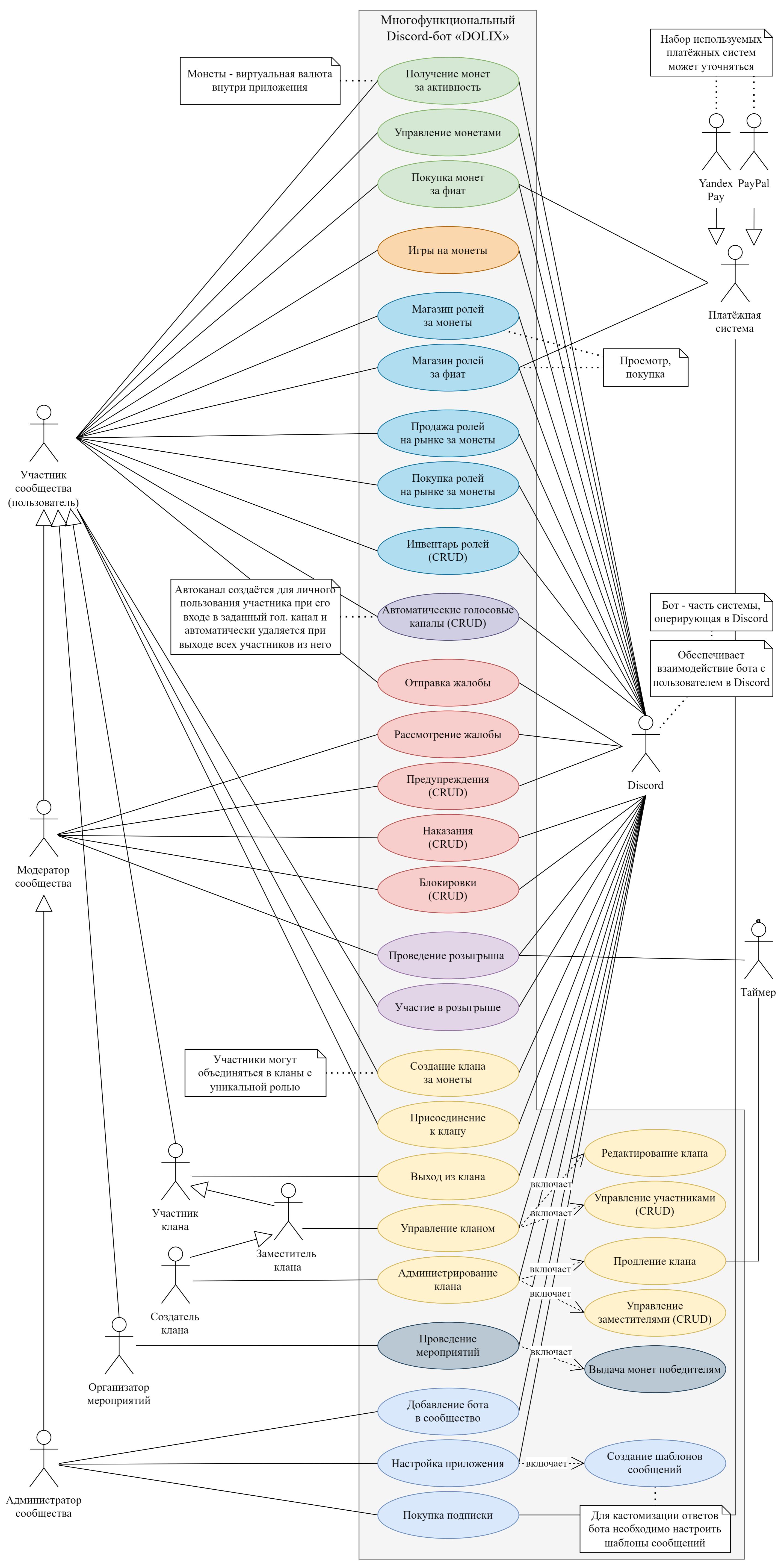


Рисунок 2 - Модель прецедентов

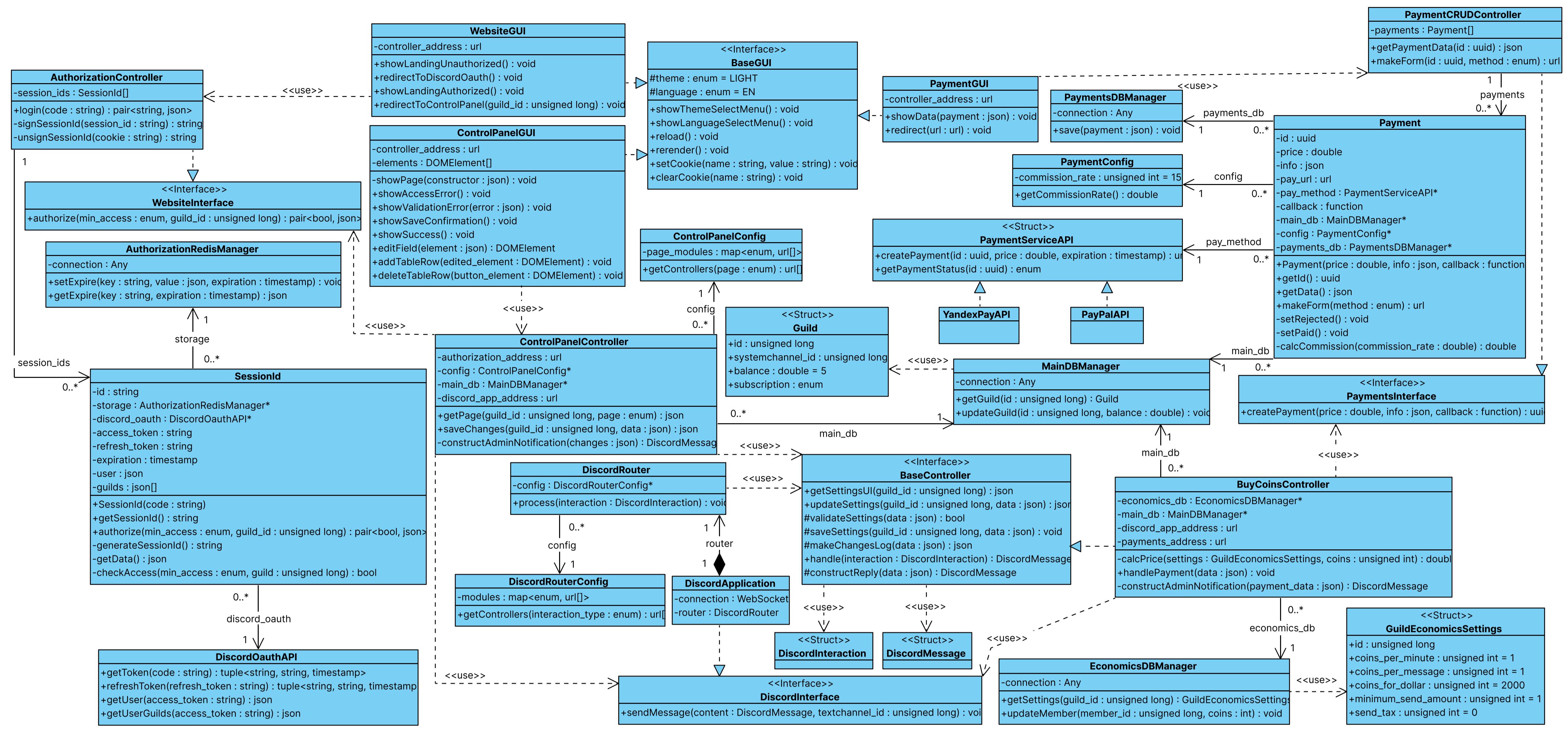


Рисунок 3 - Проектная диаграмма классов

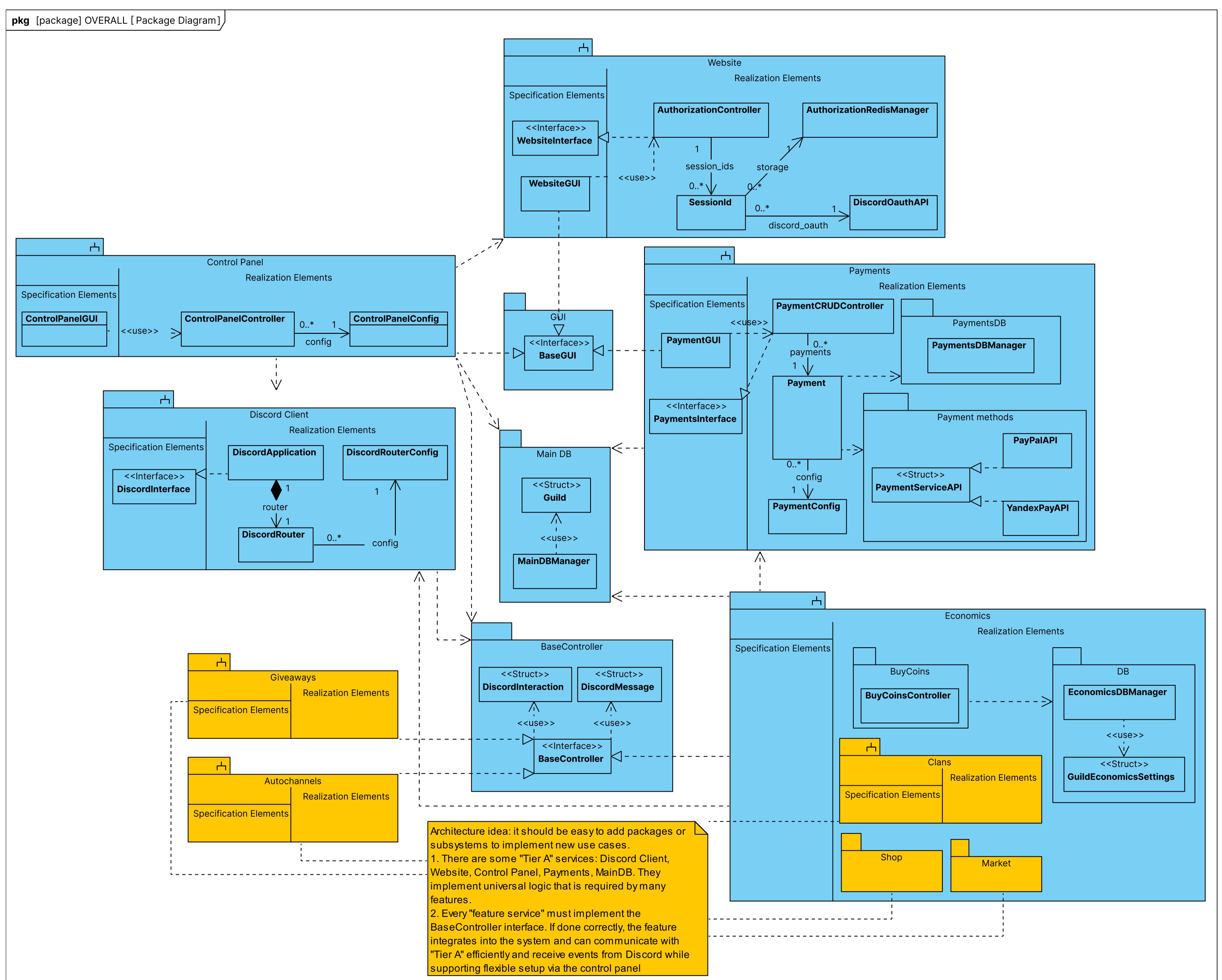


Рисунок 4 - Проектная диаграмма пакетов

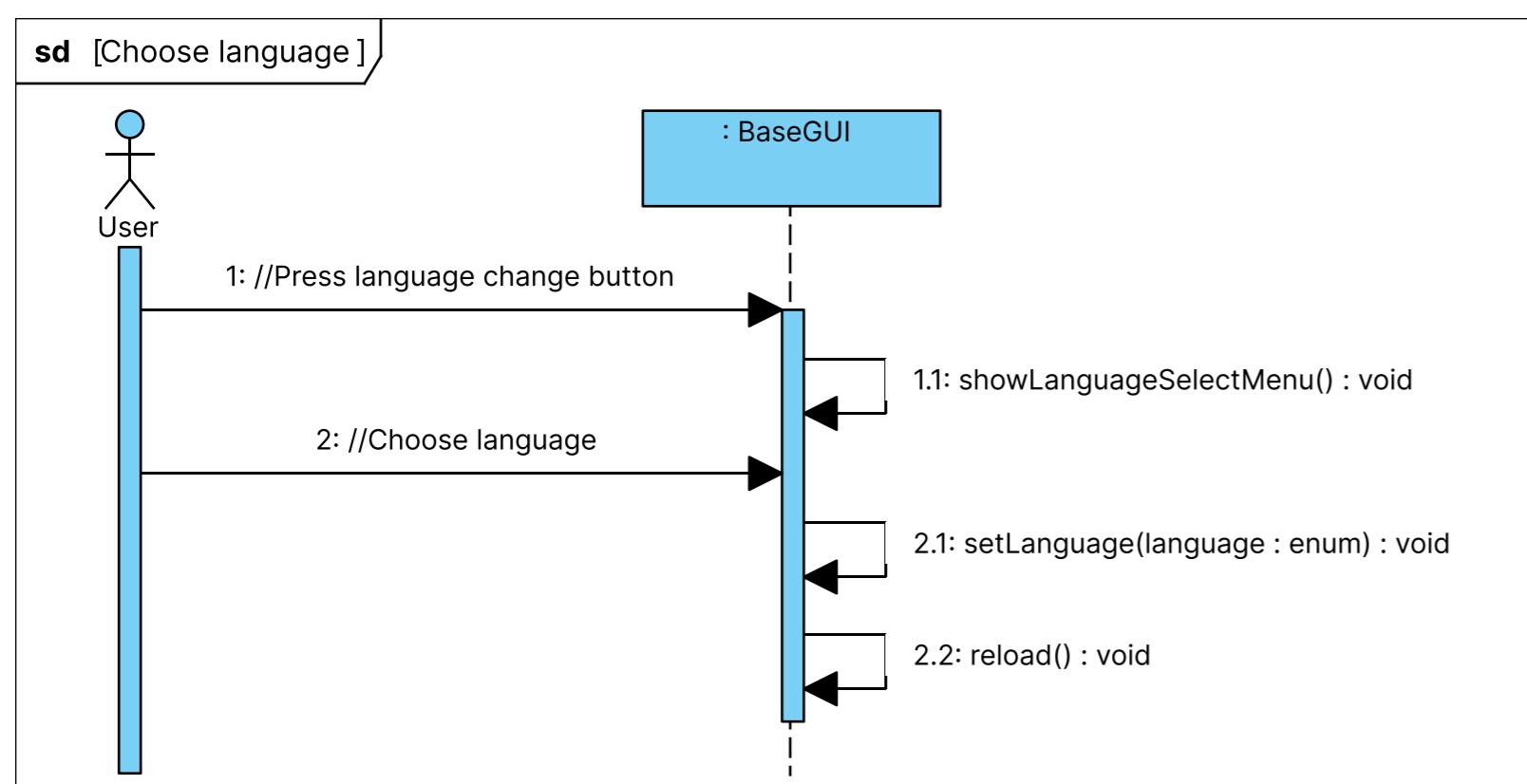


Рисунок 5 - Диаграмма последовательности выбора языка

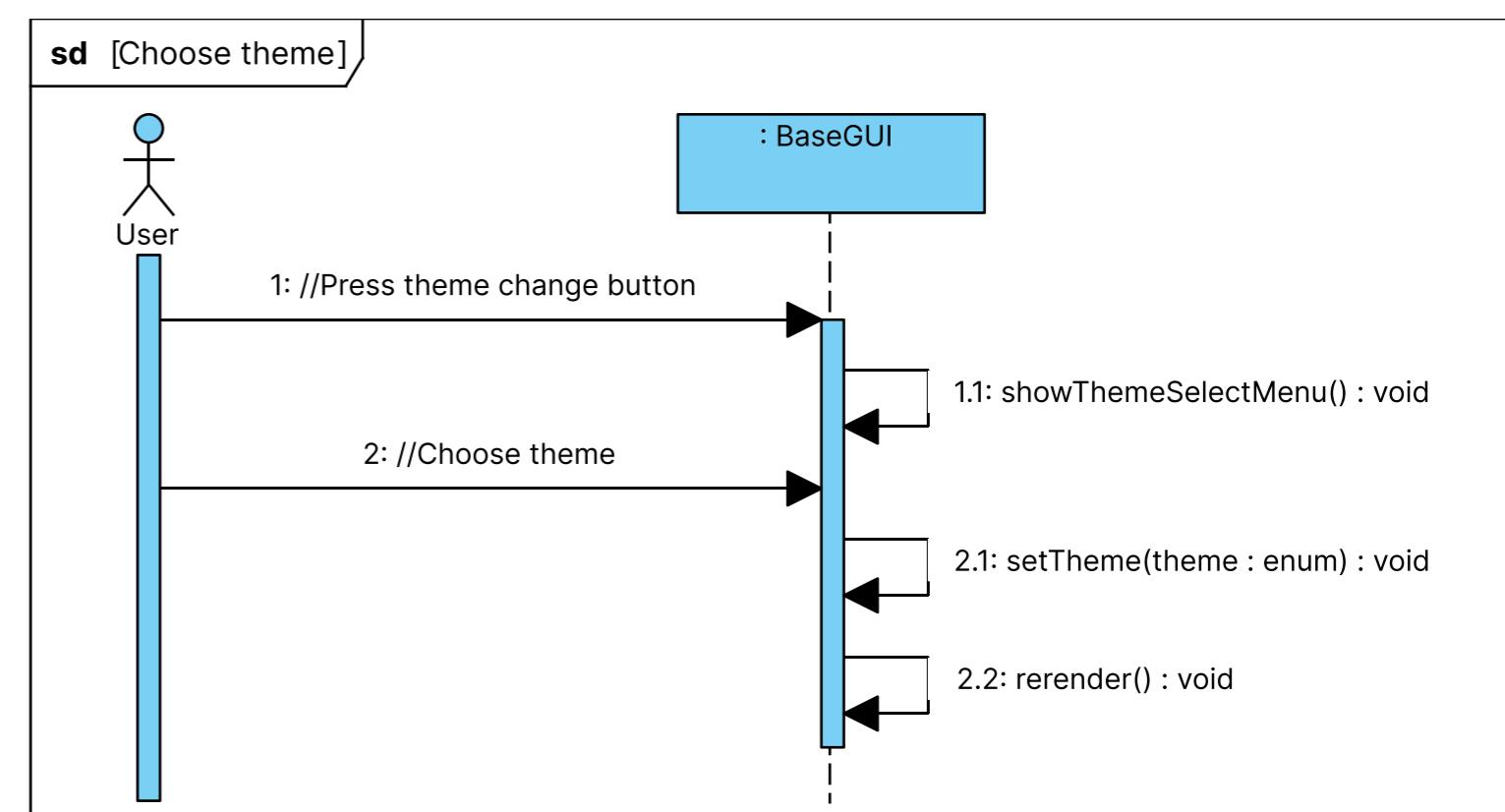


Рисунок 6 - Диаграмма последовательности выбора темы интерфейса

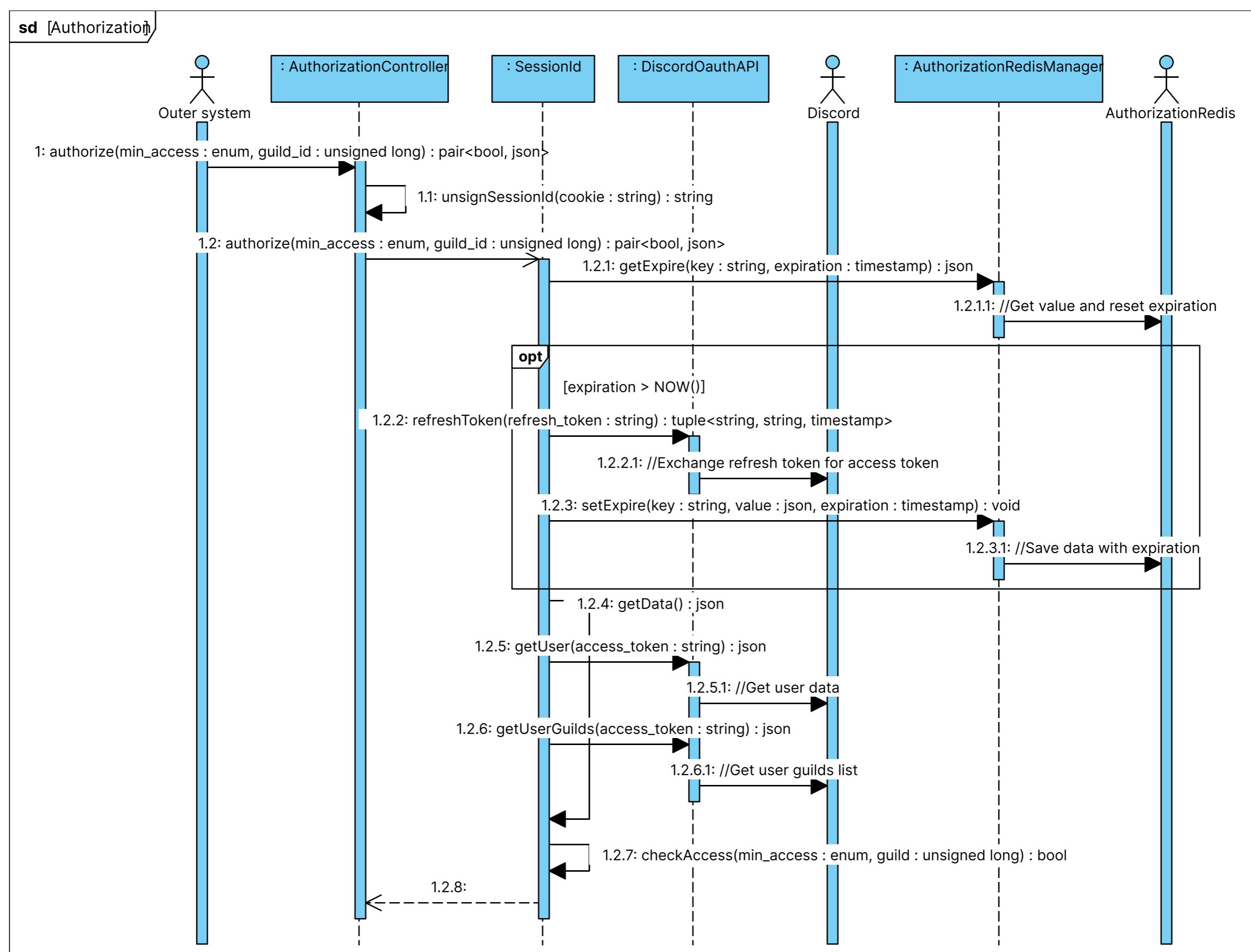


Рисунок 7 - Диаграмма последовательности авторизации

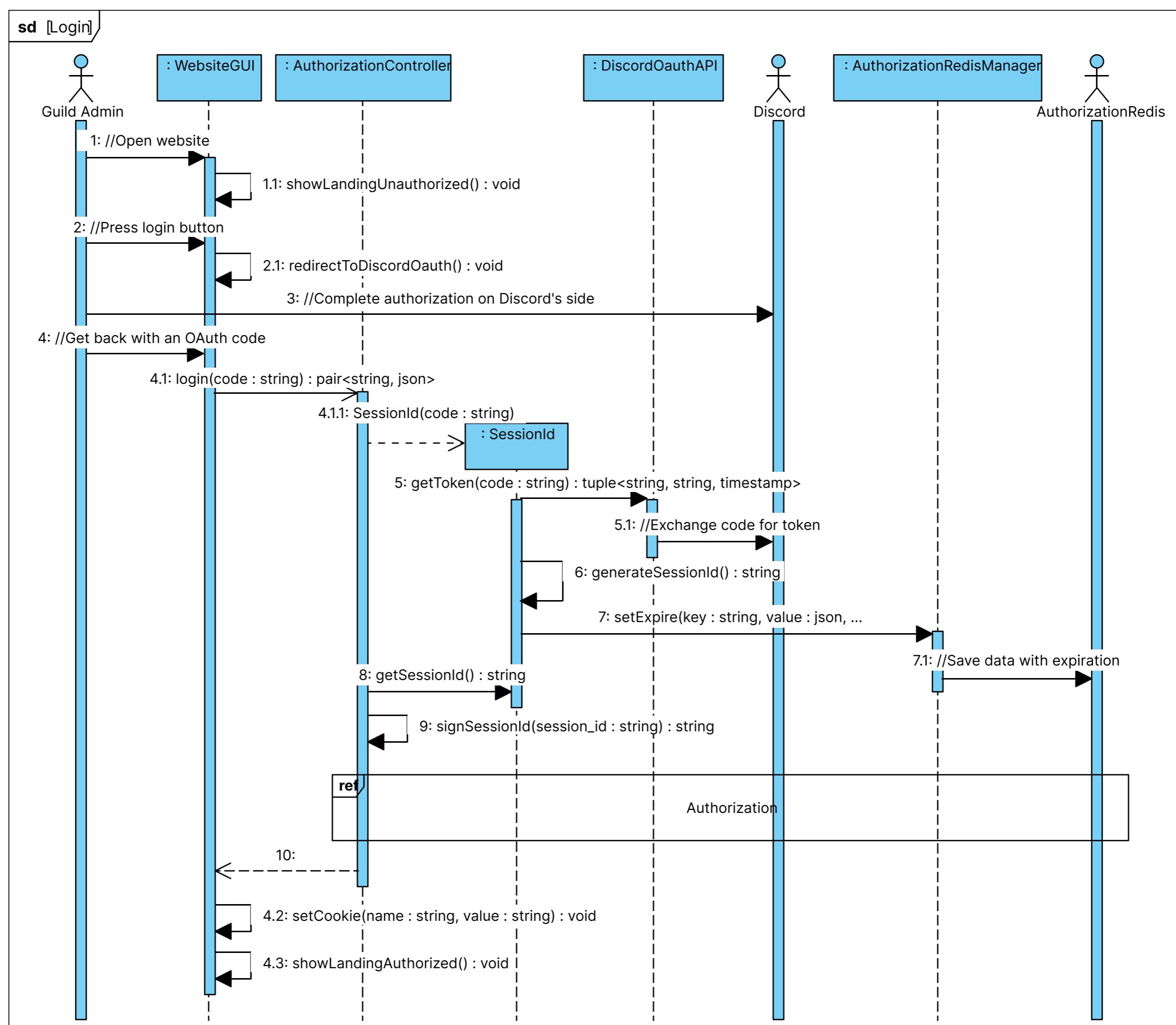


Рисунок 8 - Диаграмма последовательности входа в аккаунт

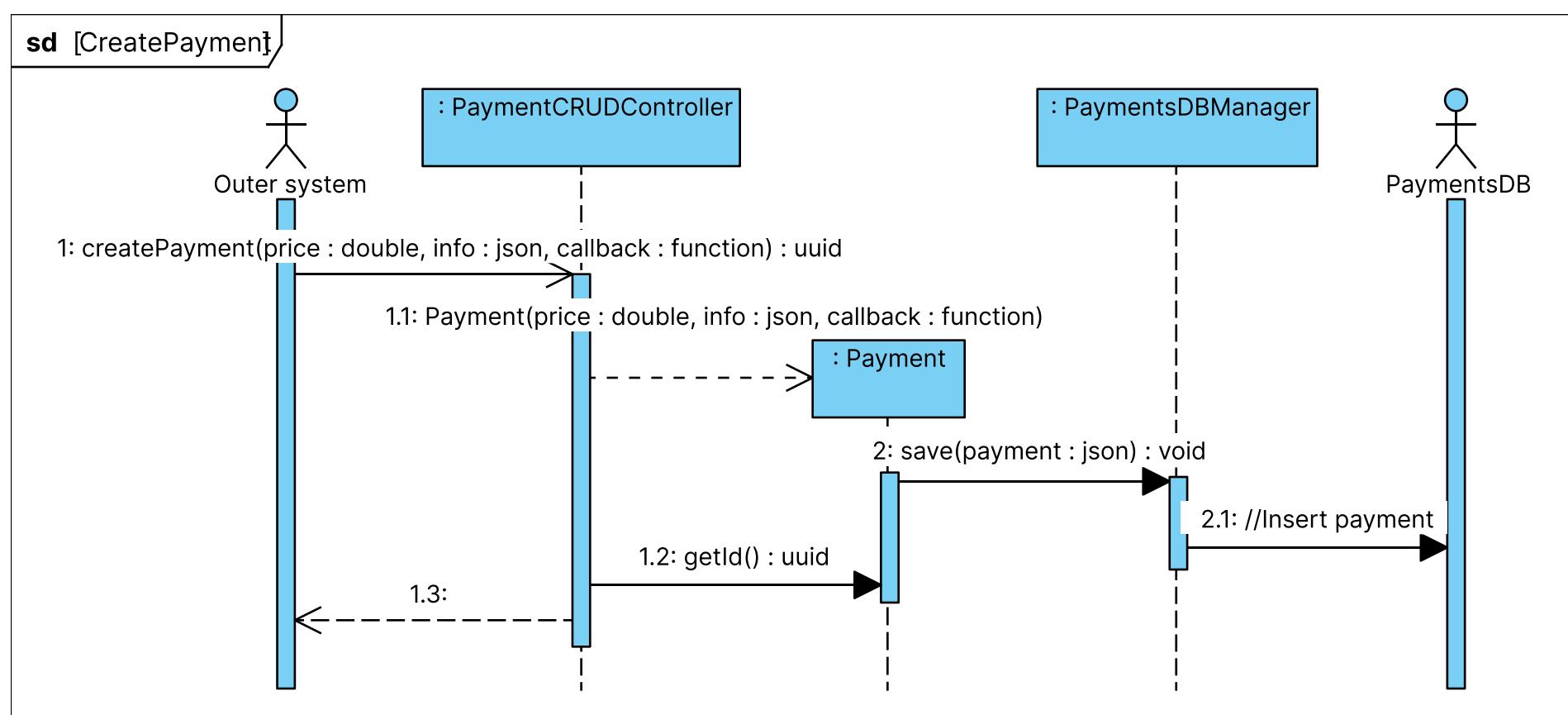


Рисунок 9 - Диаграмма последовательности создания платежа

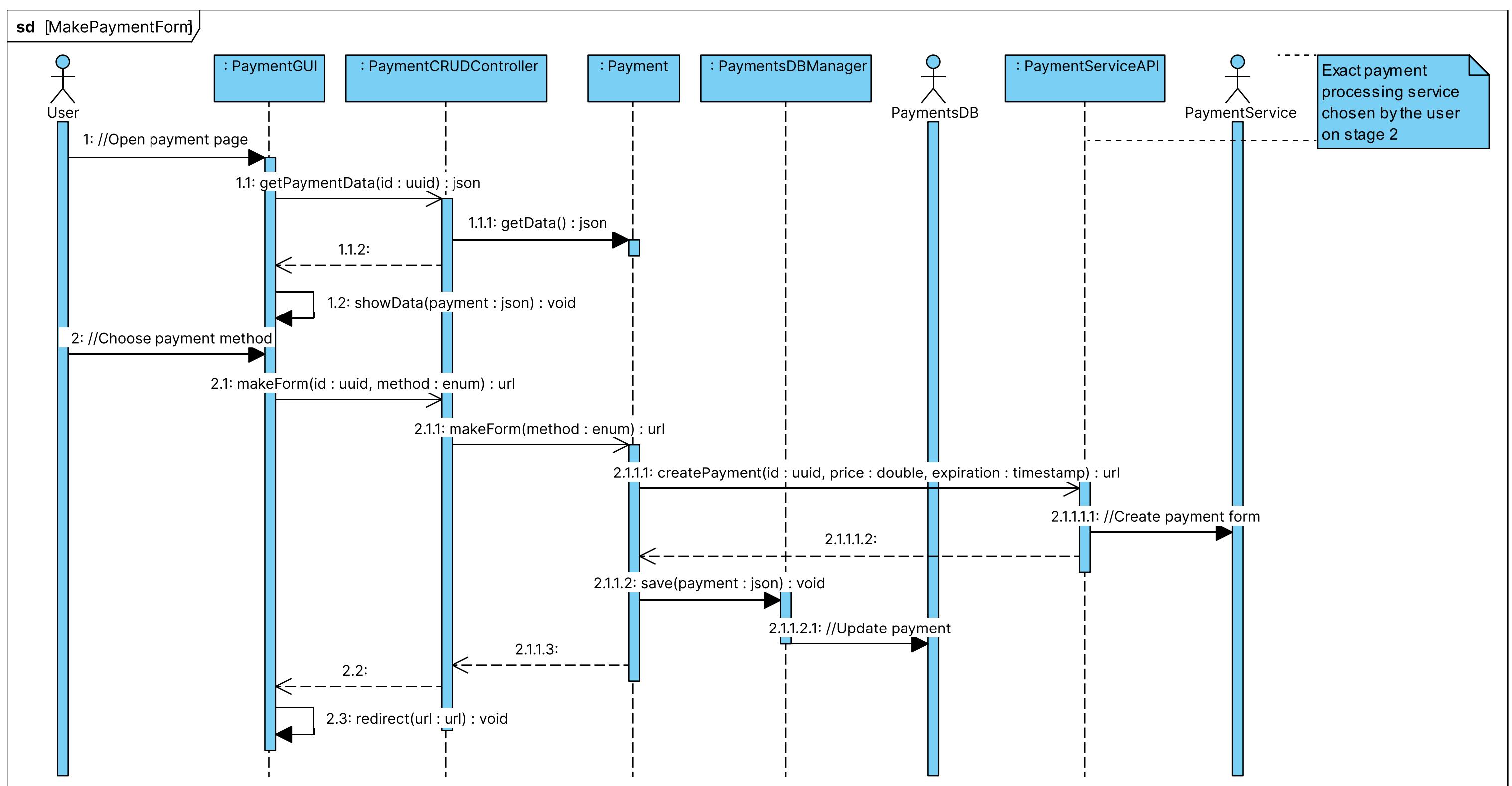


Рисунок 10 - Диаграмма последовательности создания формы оплаты

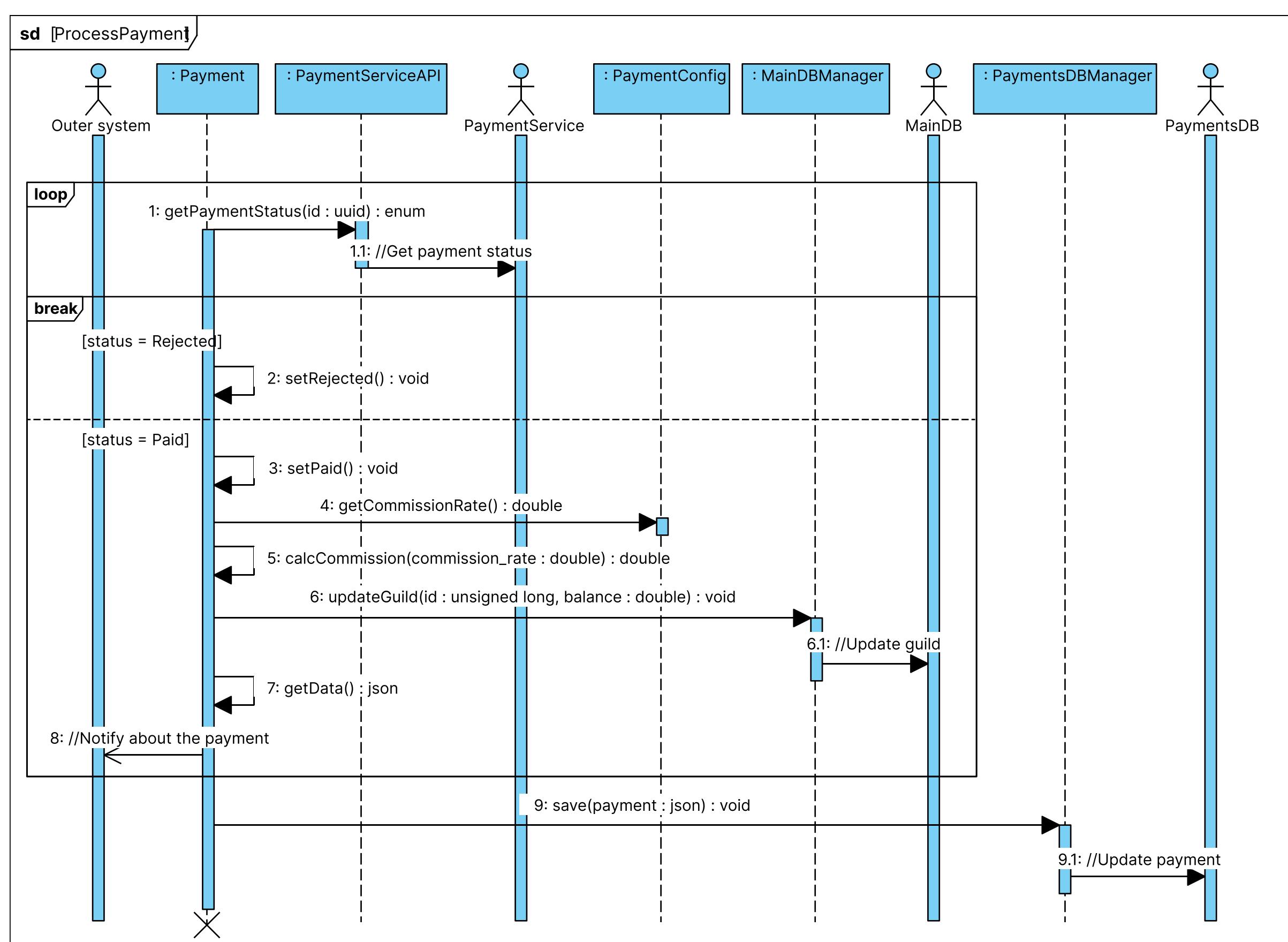


Рисунок 11 - Диаграмма последовательности обработки платежа

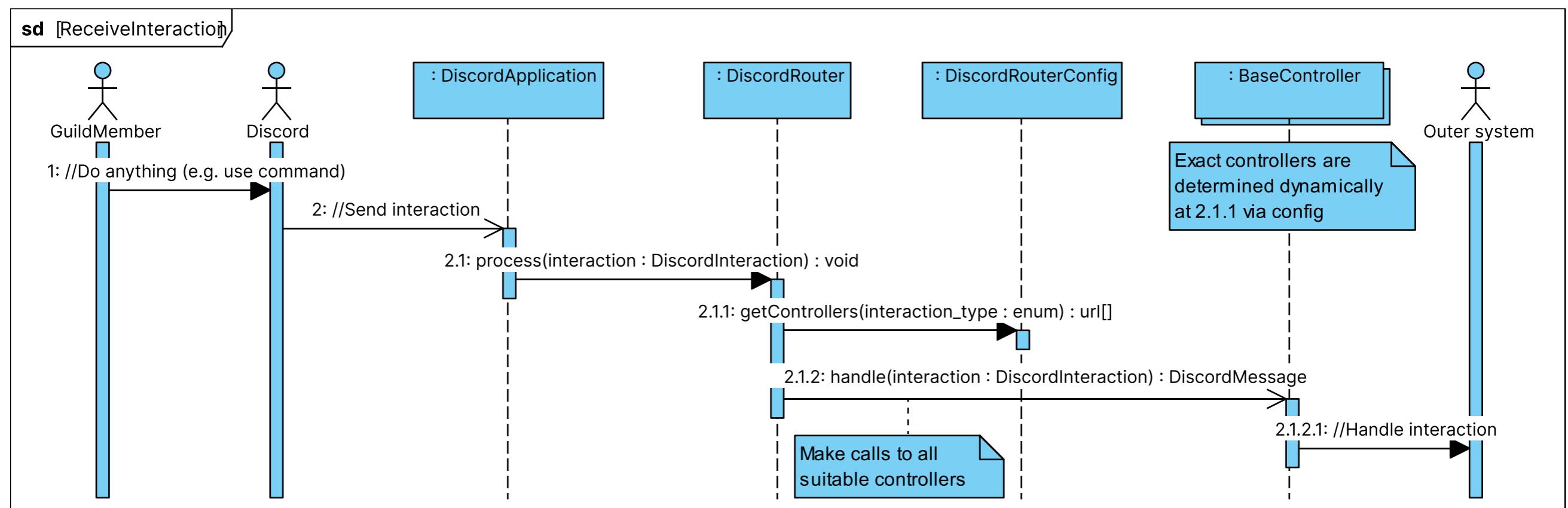


Рисунок 12 - Диаграмма последовательности приёма взаимодействия от Discord

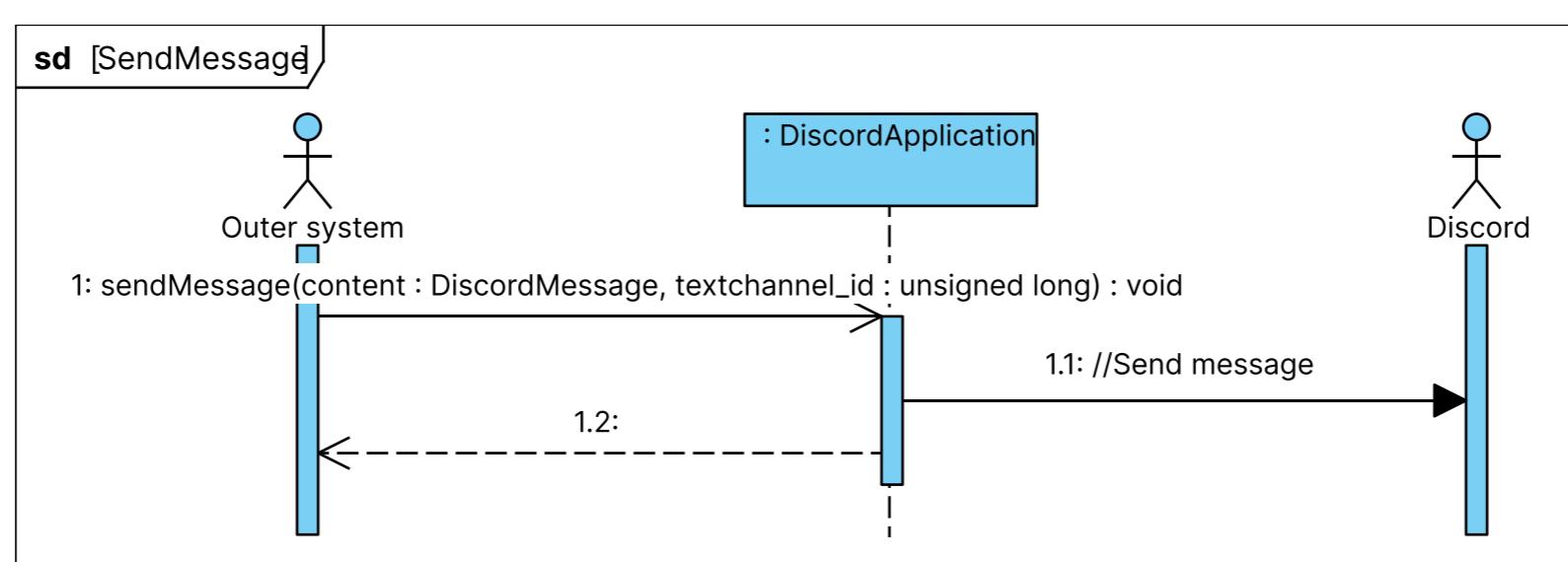


Рисунок 13 - Диаграмма последовательности отправки сообщения

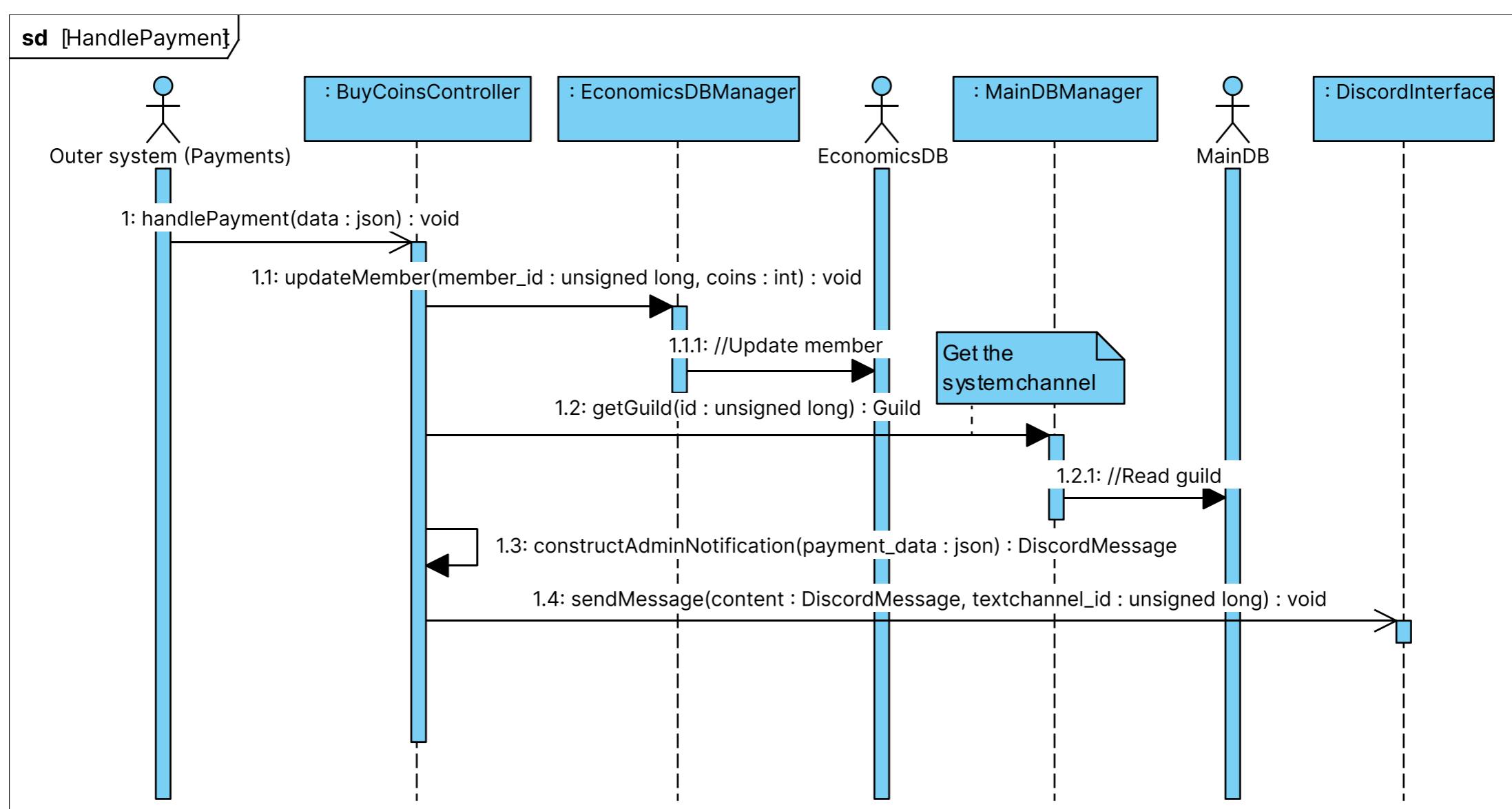


Рисунок 14 - Диаграмма последовательности обработки платежа при покупке монет

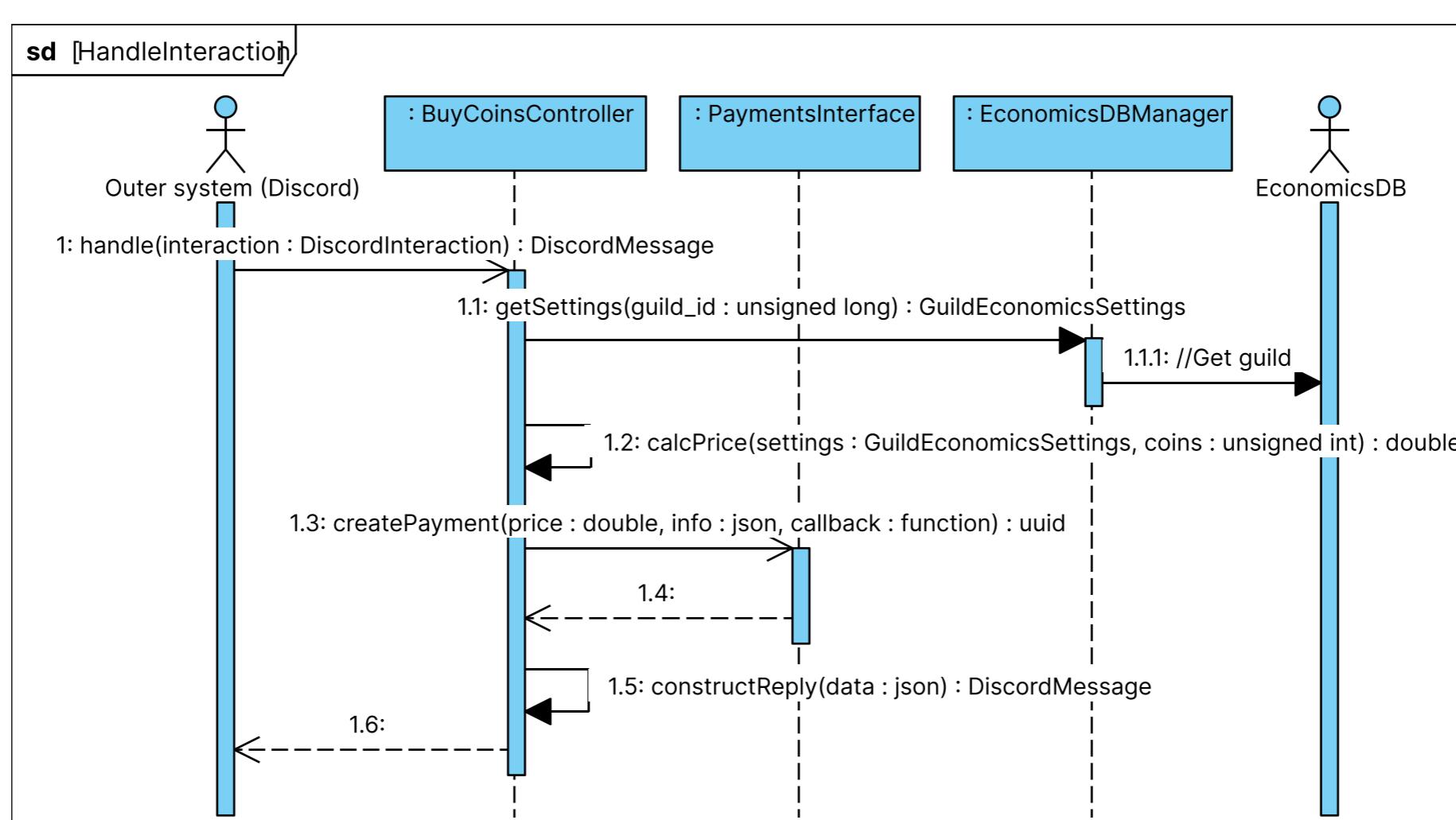


Рисунок 15 - Диаграмма последовательности обработки взаимодействия при покупке монет

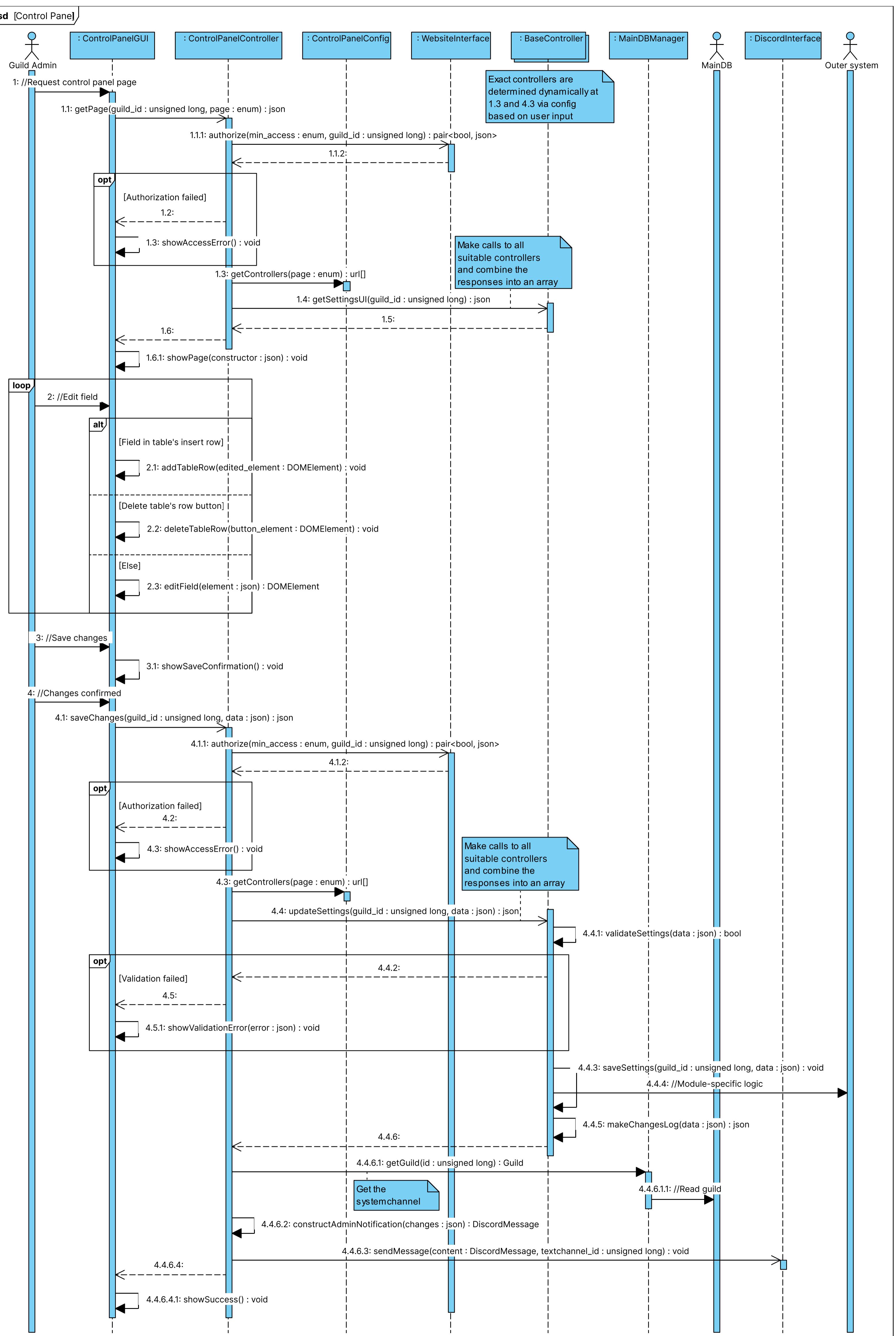


Рисунок 16 - Диаграмма последовательности настройки приложения в панели управления

Deployment Diagram

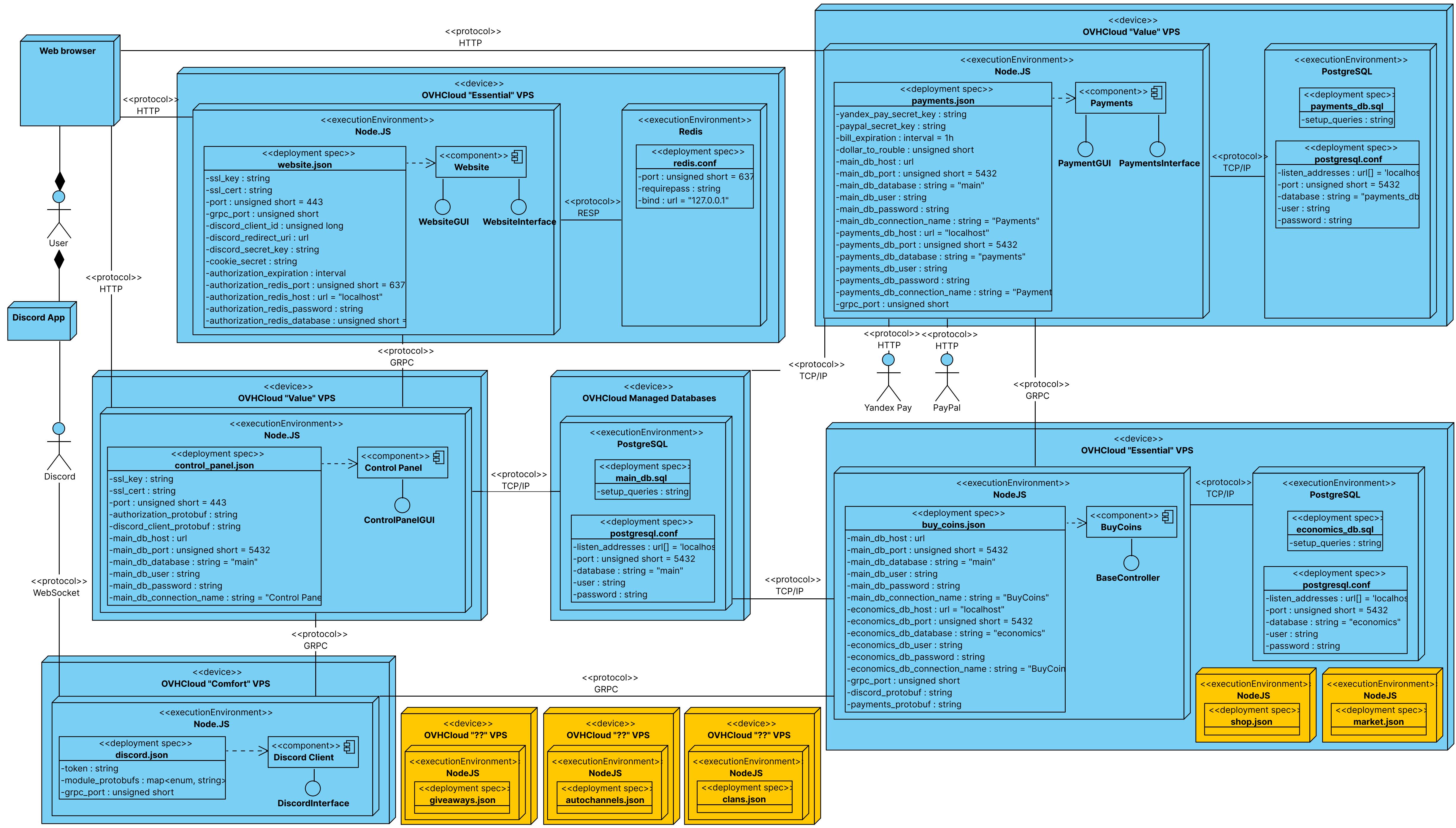


Рисунок 17 - Диаграмма развёртывания