

Диаграмма классов

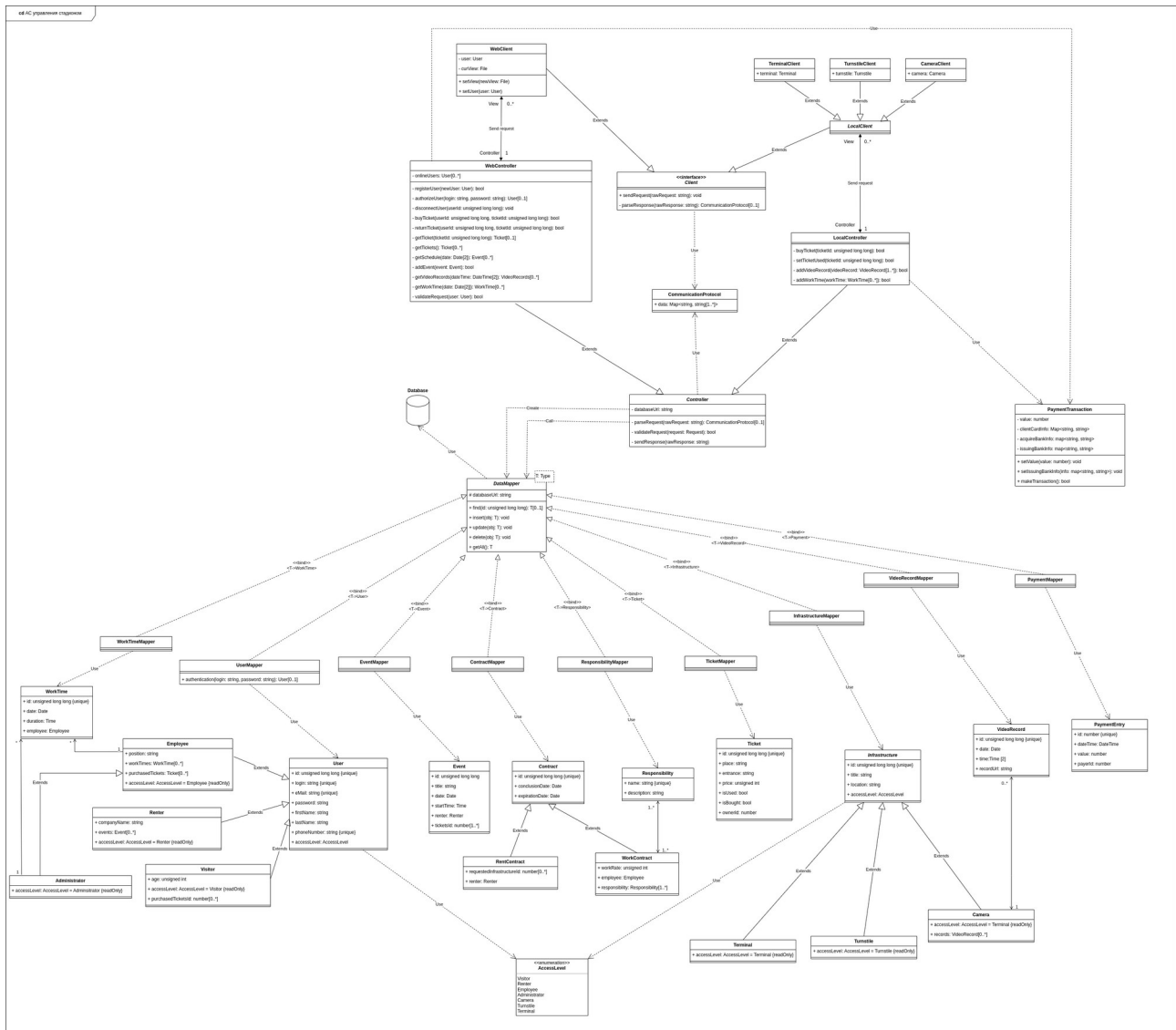


Рисунок 1: Диаграмма классов

Описание:

Диаграмма изображена на рис. 1.

При создании диаграммы классов использовалась концепция Model-View-Controller (MVC).

View:

- WebClient — класс, представляющий собой сущность, которая будет находиться в браузере пользователя, либо с которой будет взаимодействовать сторонние сервисы для покупки билетов.
- TerminalClient, TunstleClient, CameraClient — классы, представляющие собой терминал, турникет и камеру соответственно, которые подключены к локальной сети и могут взаимодействовать с контроллером.
- Интерфейс Client позволяет отправлять запросы по локальной сети к контроллеру и парсить его ответы.

Controller:

- WebController, LocalController — контроллеры для запросов по сети Интернет и локальной сети соответственно. Разделение необходимо, так как функционал для локальных и сетевых клиентов отличается.
- Оба контроллера используют класс PaymentTransaction, который представляет собой банковскую транзакцию для оплаты билета.

- Абстрактный класс Controller предоставляет общий функционал контроллеров по отправке ответов, парсинга запросов и их валидации. Также задается URL для подключения к базе данных.

Model:

- Для каждой сущности из модели предметной области в базе данных созданы соответствующие классы: Ticket, Employee, Renter и т. д. (используется РоЕАА-паттерн «Domain model»), для каждого из которых задан mapper, задача которого - инкапсулировать код для работы с самой базой данных. В данном случае использован промышленный паттерн проектирования «Data mapper».

Перечисление AccessLevel задает всевозможные уровни доступа, они используются в методах контроллеров для проверки, может ли пользователь системы выполнять выбранную операцию (методы сами «знают», какие пользователи могут их выполнять).

Описание использованных шаблонов проектирования:

- Шаблон MVC использован для повышения повторного использования кода, который будет написан при реализации системы, т. е. чтобы в случае необходимости, можно было бы изменить View, не меняя контроллер, и наоборот.
- В данной системе используется «простая» разновидность шаблона «Модель предметной области», так как одному объекту домена соответствует одна реляционная таблица. Использование данного шаблона позволяет упростить понимание диаграммы классов, причем «Модель предметной области» напоминает соответствующую базу данных. В качестве примера можно привести класс «Employee» (Работник), который соответствует реляционной таблице, в которой имеются атрибуты id, login, password, eMail, position, workTimes и т. д.
- Шаблон «Data mapper» позволяет упростить взаимодействие с БД и сделать его более гибким. Данный шаблон хорошо работает в связке с паттерном «Domain model». Рассмотрим, например, класс «Event» (Событие). Контроллер при необходимости создает соответствующий класс, в данном случае это EventMapper, который инкапсулирует в себе код обращения к БД и код выполнения определенных операции (поиск, вставка, обновление, удаление). То есть классы модели предметной области хранят только данные и не знают о существовании БД. Для каждого события выпускается несколько билетов, в БД это будет отражаться в виде внешних ключей в реляционной таблице «Событие», который ссылаются на ключи из таблицы «Билет». В задачи «преобразователя» (mapper) будет входить нахождение всех билетов, на которые ссылается определенное событие, и предоставление этой информации в виде поля-массива (в данном примере это ticketsId).

Диаграмма объектов

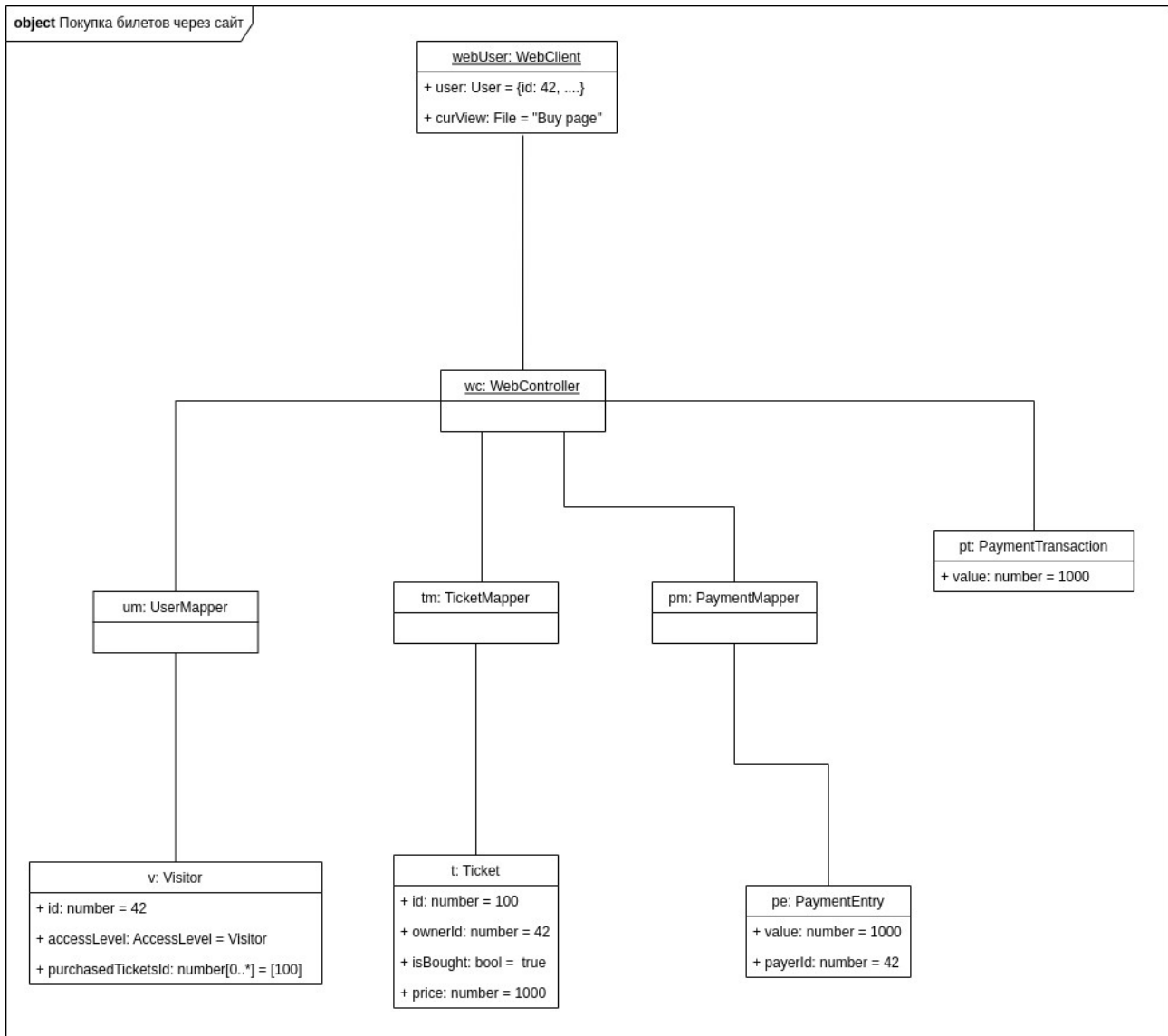


Рисунок 2: Диаграмма объектов

Описание:

Диаграмма (см. рис. 2) представляет собой «снимок» системы во время процесса покупки билета пользователем через веб-сайт. В объекте **webUser** хранится информация о текущем пользователе: в данном случае важен лишь **id** пользователя в БД — и название текущей HTML-страницы.

В объекте **v** хранится такой же **id** как и у объекта **webUser** (в данном случае роль пользователя — посетитель) и список купленных билетов.

В объекте билета **t** имеется идентификатор билета в БД, **id** владельца и его цена.

pe — объект, соответствующий записи в таблице БД с информацией о проведенных транзакциях. Хранящаяся информация: заплаченная сумма и **id** покупателя.

В объекте **pt** представляет собой экземпляр проведенной транзакции, в хранится заплаченная сумма.

Диаграмма состояний

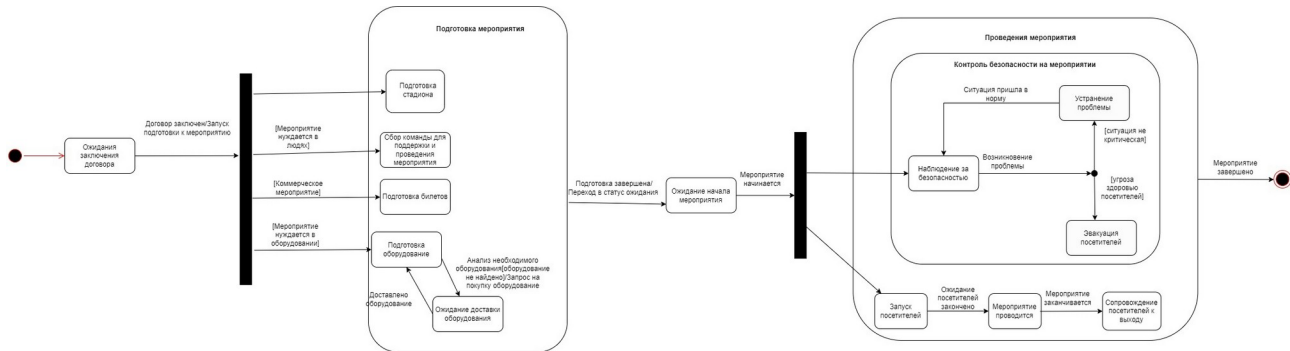


Рисунок 3: Диаграмма состояний

Описание:

На данной диаграмме (см. рис. 3) показано то, какие состояние принимает объект мероприятия. Сначала система ждёт, когда арендатор подпишет с администрацией договор аренды. В этом договоре прописаны требования к стадиону и дата проведения. Мероприятие находится в состоянии подготовки: подготавливается стадион, собирается команда для помощи проведения, подготавливаются билеты и устанавливается оборудование. Все указания и информация о подготовки содержится в разрабатываемой системе. Далее во время самого мероприятия система должна обеспечивать контроль билетов и безопасность посетителей.

Диаграмма деятельности

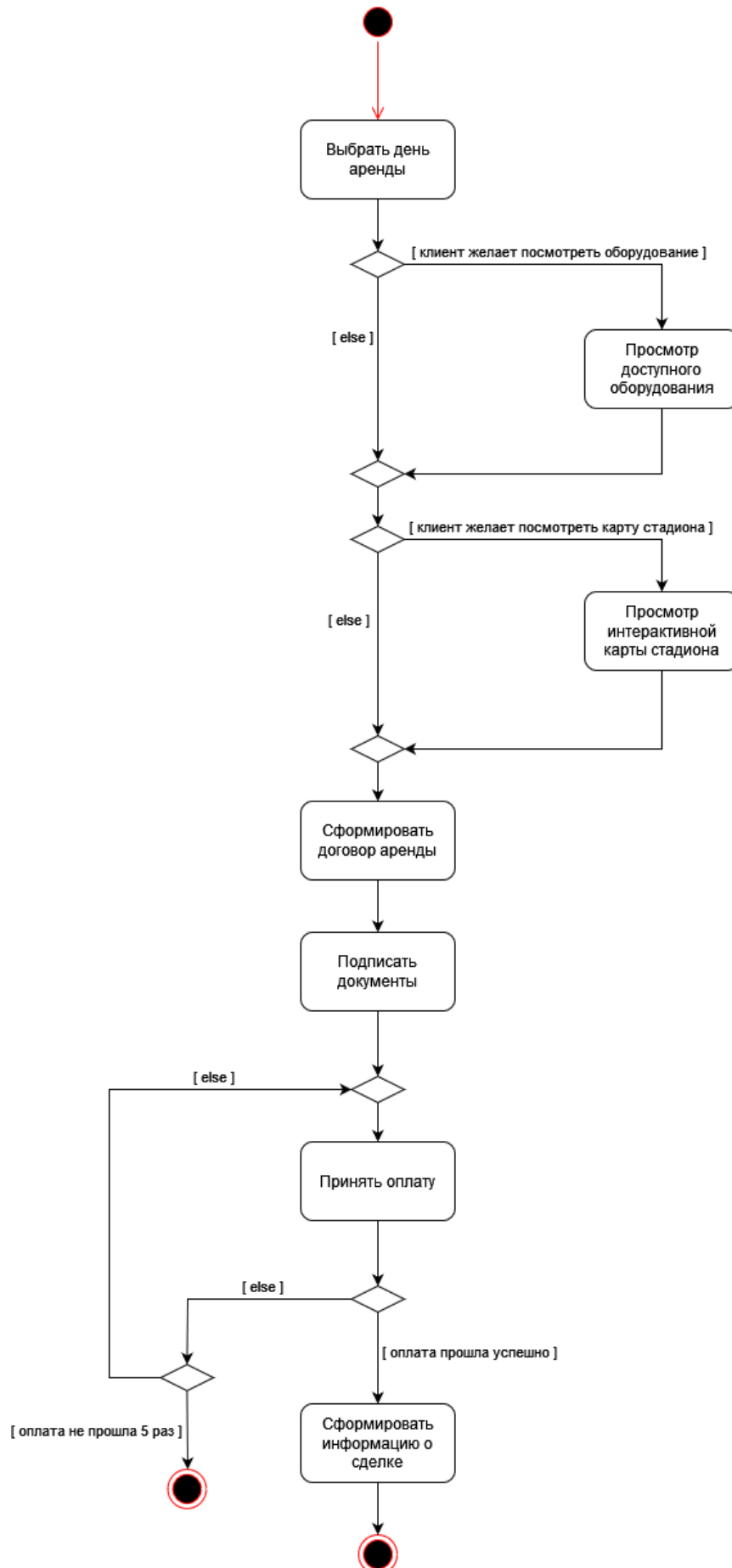


Рисунок 4: Диаграмма деятельности (аренда стадиона)

Описание:

Диаграмма деятельности, представленная на рис. 4, описывает алгоритм аренды стадиона:

- «Выбрать день аренды» - пользователь выбирает день для проведения мероприятия
- «Просмотр доступного оборудования» - пользователь может посмотреть все имеющееся на стадионе оборудование для проведения мероприятия
- «Просмотр интерактивной карты стадиона» - пользователь может посмотреть карту стадиона для оценки количества мест и посадки
- «Сформировать договор аренды» - подготовка необходимых документов для предоставления аренды
- «Подписать документы» - окончательное оформление документов на аренду
- Потом происходит процесс оплаты. «Принять оплату» - внесение суммы клиентом. Если при оплате возникли трудности, то даётся 5 попыток произвести оплату, после которых процесс аренды прерывается.
- Если оплата прошла успешно, формируется и обрабатывается информация о сделке

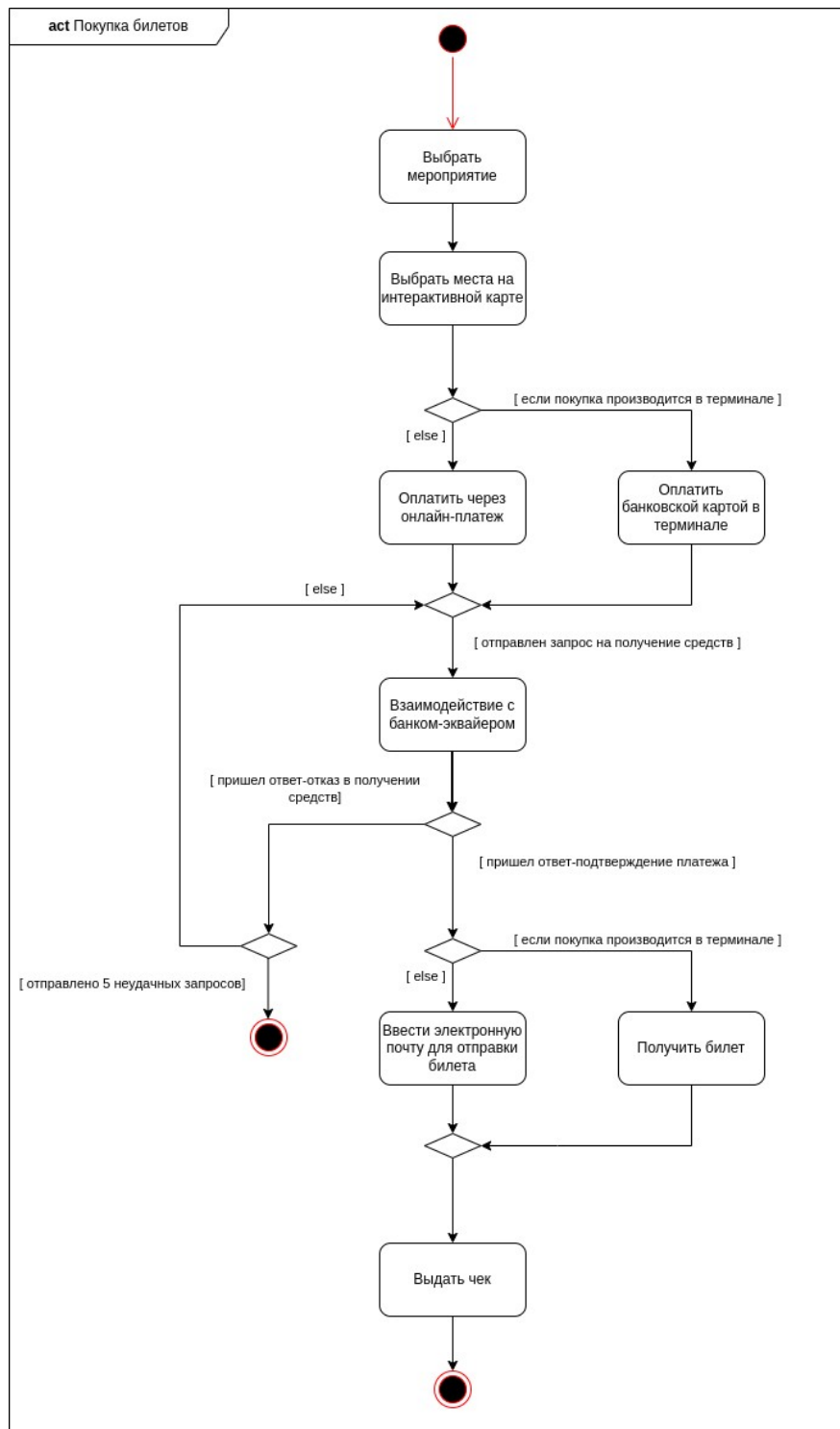


Рисунок 5: Диаграмма деятельности (покупка билета)

Описание:

Диаграмма деятельности, представленная на рис. 5, описывает алгоритм покупки билета:

- «Выбрать мероприятие» - пользователь выбирает мероприятие, на которое хочет купить билет
- «Выбрать места на интерактивной карте» - пользователь выбирает от 1 до 10 свободных мест на выбранном мероприятии
- Если пользователь приобретает билет в терминале на стадионе, он должен «Оплатить банковской картой в терминале» стоимость билетов. Если же пользователь приобретает билет онлайн, то он должен «Оплатить через онлайн-платеж»

- Далее идет взаимодействие с банковской системой: отправляется запрос в банк-эквайер на получение средств от клиента, в это время банк-эквайер налаживает связь с банком-эмитентом клиента, через который клиент производит оплату. После этого банк-эквайер возвращает ответ – результат операции. Ответ-подтверждение означает. Что операция прошла успешно, а ответ-отказ – что при оплате возникли проблемы. После 5 попыток на запрос средств покупка билетов останавливается.
- Если оплата произведена успешно, то пользователь «Получает билет», если пользовался терминалом, или «Вводит электронную почту для отправки билета»
- После этого пользователь «Получает чек»

Диаграмма последовательности

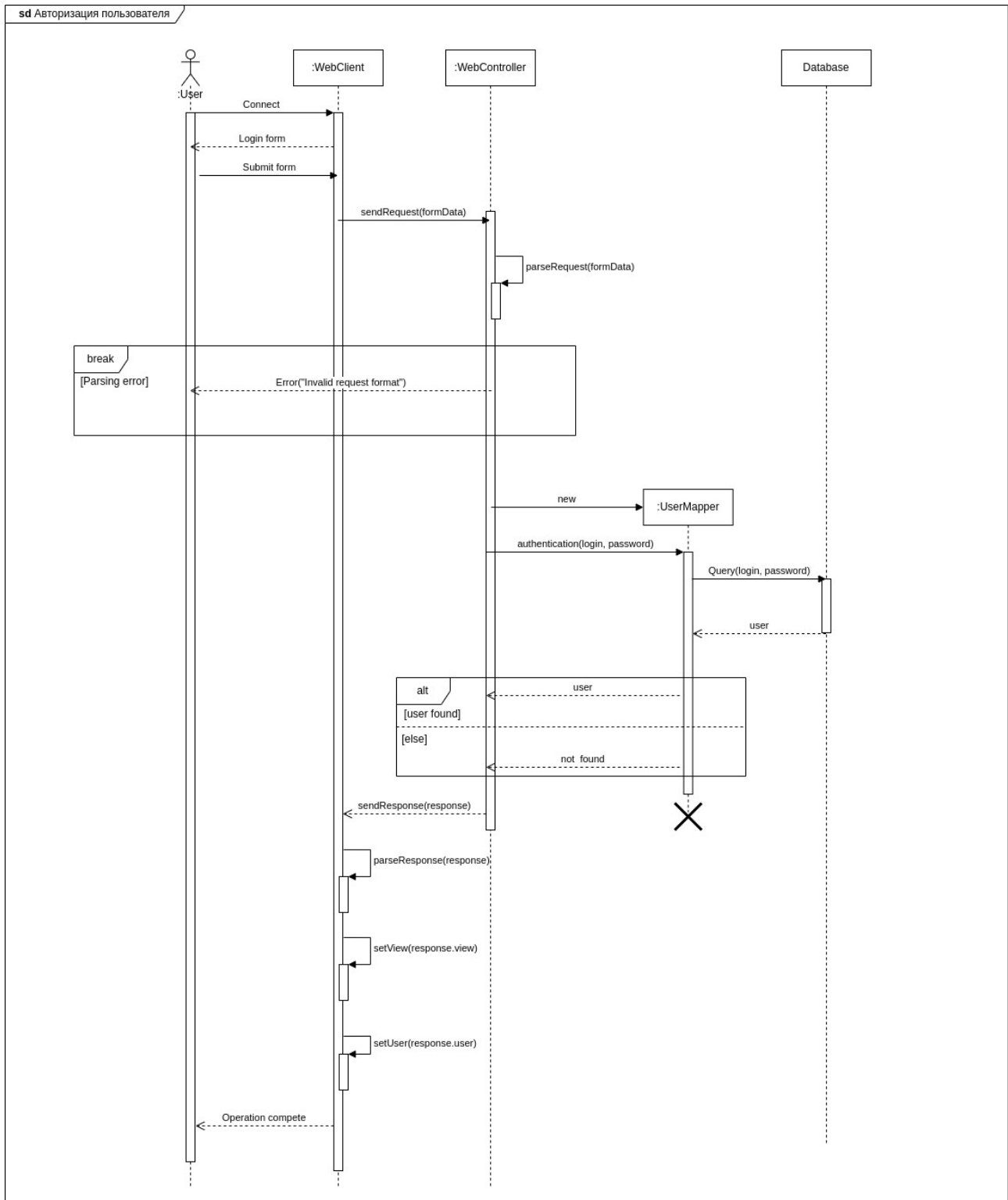


Рисунок 6: Диаграмма последовательности (авторизация пользователя)

Описание:

На рис. 6 приведена диаграмма последовательности, описывающая процесс авторизации пользователя. Сначала актер (User) подключается в веб-клиенту (WebClient) и заполняет форму для авторизации. Далее веб-клиент формирует запрос и отправляет его на веб-сервер приложения (WebController), где запрос парсится. В случае, если парсинг данных не удался, пользователю возвращается страница с информацией об ошибке. Затем контроллер создает экземпляр объекта UserMapper для поиска пользователя в БД по введенному логину и паролю.

Если соответствующей записи нет в БД, то пользователю возвращается информация об этом. В ином случае, на веб-клиент возвращается ответ от сервера, который парсится. Затем с помощью сеттеров обновляются соответствующие поля веб-клиента, и процесс авторизации завершен.

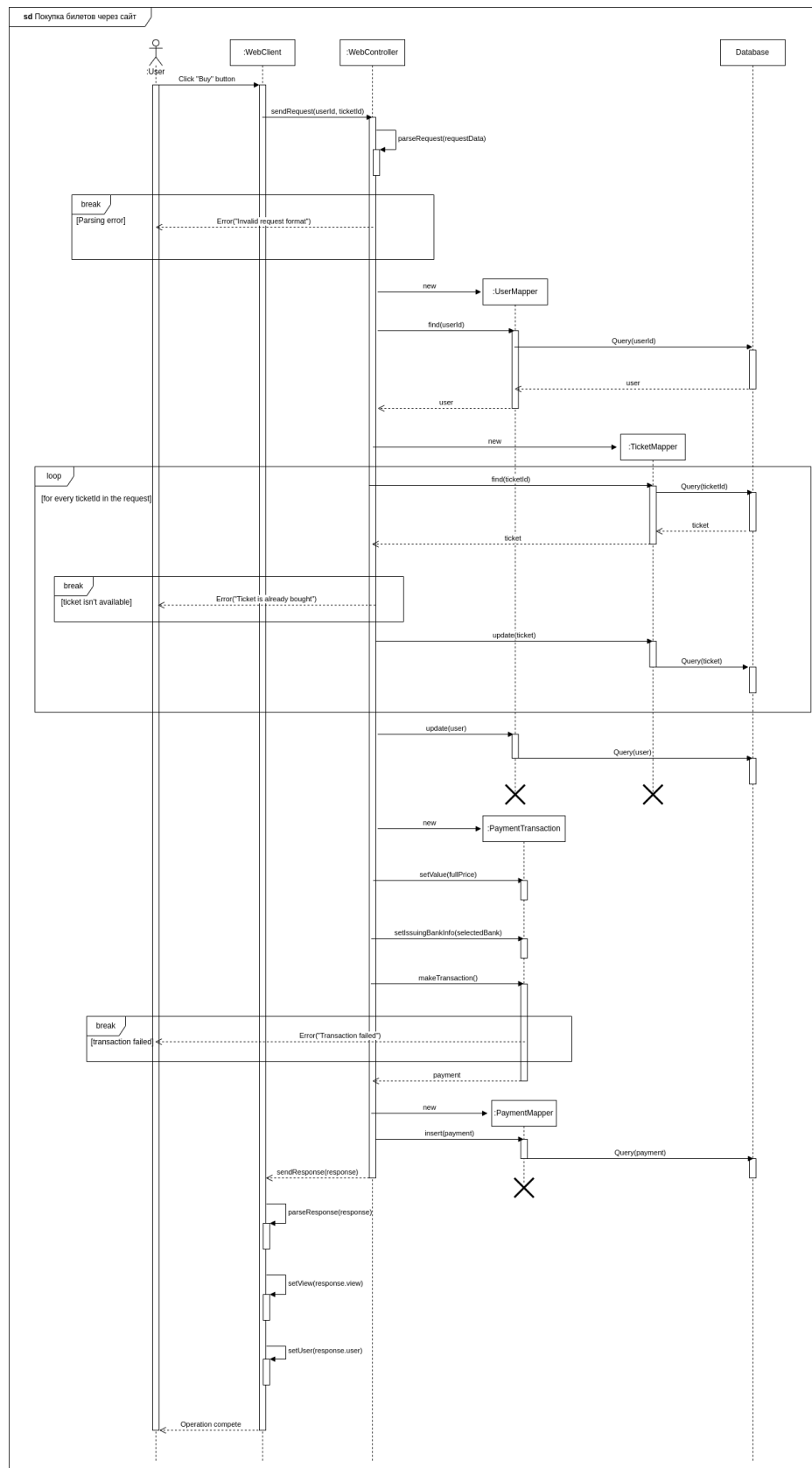


Рисунок 7: Диаграмма последовательности (покупка билета через сайт)

На рис. 7 представлена диаграмма последовательности, описывающая процесс покупки билета пользователем через веб-сайт. Сначала актер (User) нажимает на кнопку «Купить», затем веб-клиент формирует запрос и отправляет его на веб-сервер приложения (WebController), где запрос парсится. В случае, если парсинг данных не удался, пользователю возвращается страница с информацией об ошибке. Затем контроллер создает экземпляр объекта UserMapper для поиска пользователя, который отправил запрос. Далее контроллер создает экземпляр объекта TicketMapper для поиска выбранных пользователем билетов в БД. Если хотя бы один из выбранных билетов уже куплен, то пользователю возвращается ошибка с информацией об этом. В ином случае в БД обновляется информация о каждом выбранном пользователем билете (записываются данные о покупателе). Затем обновляется информация о самом пользователе, который выполнил запрос (обновляется список купленных билетов). Затем создается инстанс класса PaymentTransaction для проведения банковской транзакции, устанавливаются цена билета, выбранный пользователем банк-эмиттер и выполняется сама транзакция. Если во время транзакции произошла ошибка, то пользователю выводится сообщение об этом. Иначе в БД с помощью класса PaymentMapper заносятся данные о новой транзакции. Далее информация на стороне пользователя обновляется и выводится сообщение об успешном завершении операции.

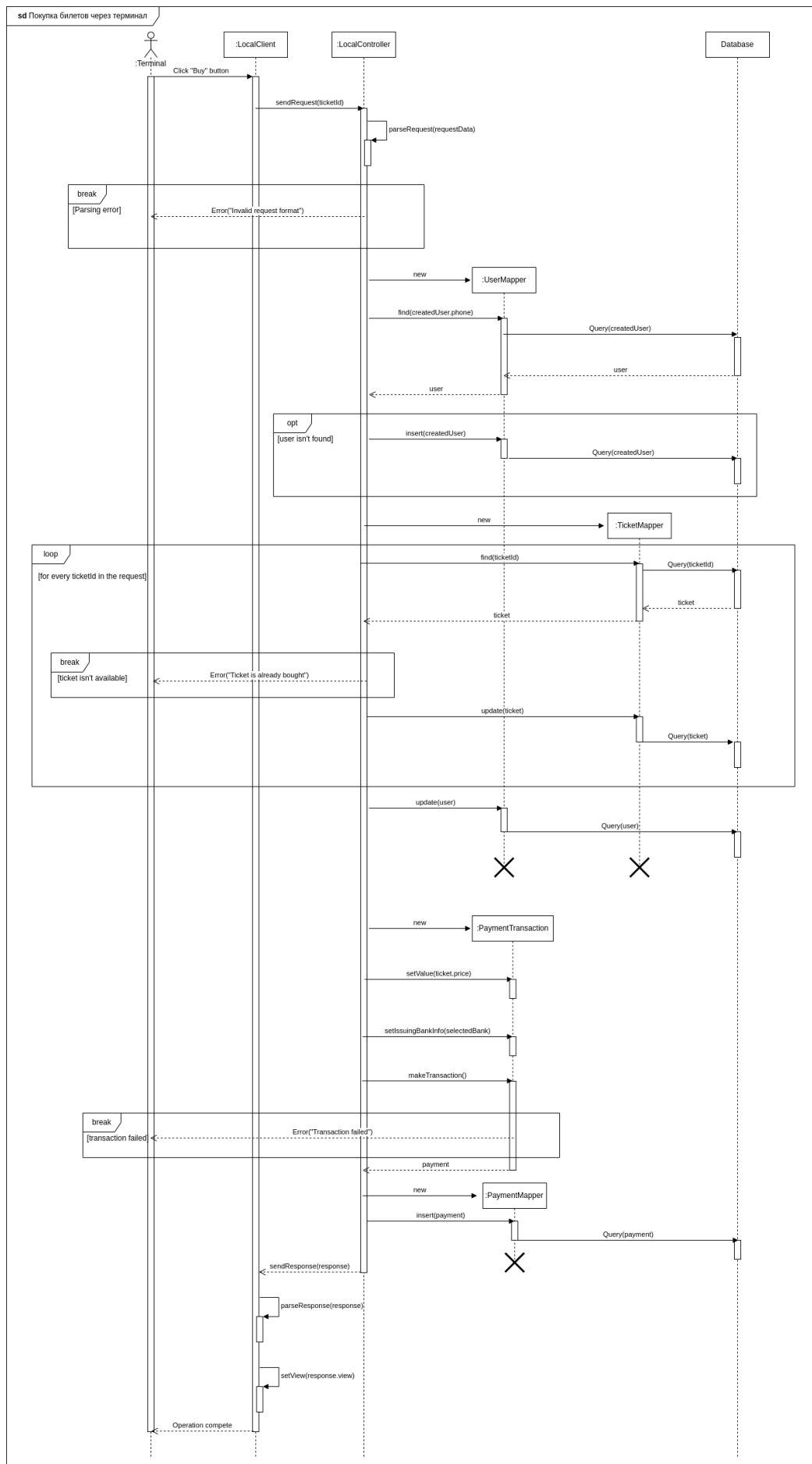


Рисунок 8: Диаграмма последовательности (покупка билета через терминал)

Описание:

Процесс покупки билета через терминал (см. рис. 8) отличается от покупки билета через веб-сайт тем, что если пользователь впервые покупает билеты, то предоставленные им данные (например, телефон) заносятся в БД. В ином случае, последовательность действий аналогична тем, которые представлены на предыдущей диаграмме (см. рис. 7).

Диаграмма коммуникации

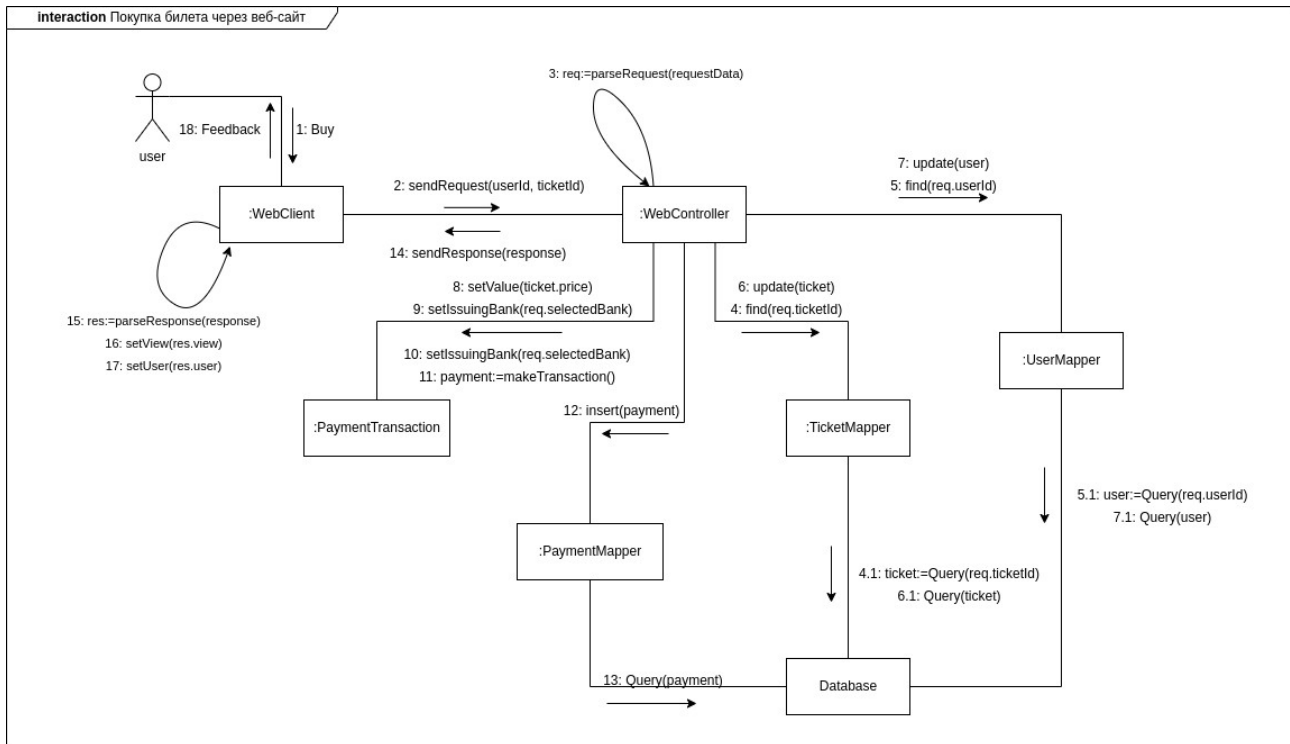


Рисунок 9: Диаграмма коммуникации (покупка билета через веб-сайт)

Описание:

Семантически предоставленная диаграмма (см. рис. 9) похожа на диаграмму последовательности для покупки билета через веб-сайт (см. рис. 7). Здесь лишь добавлены связи между экземплярами классов.

Диаграмма компонентов

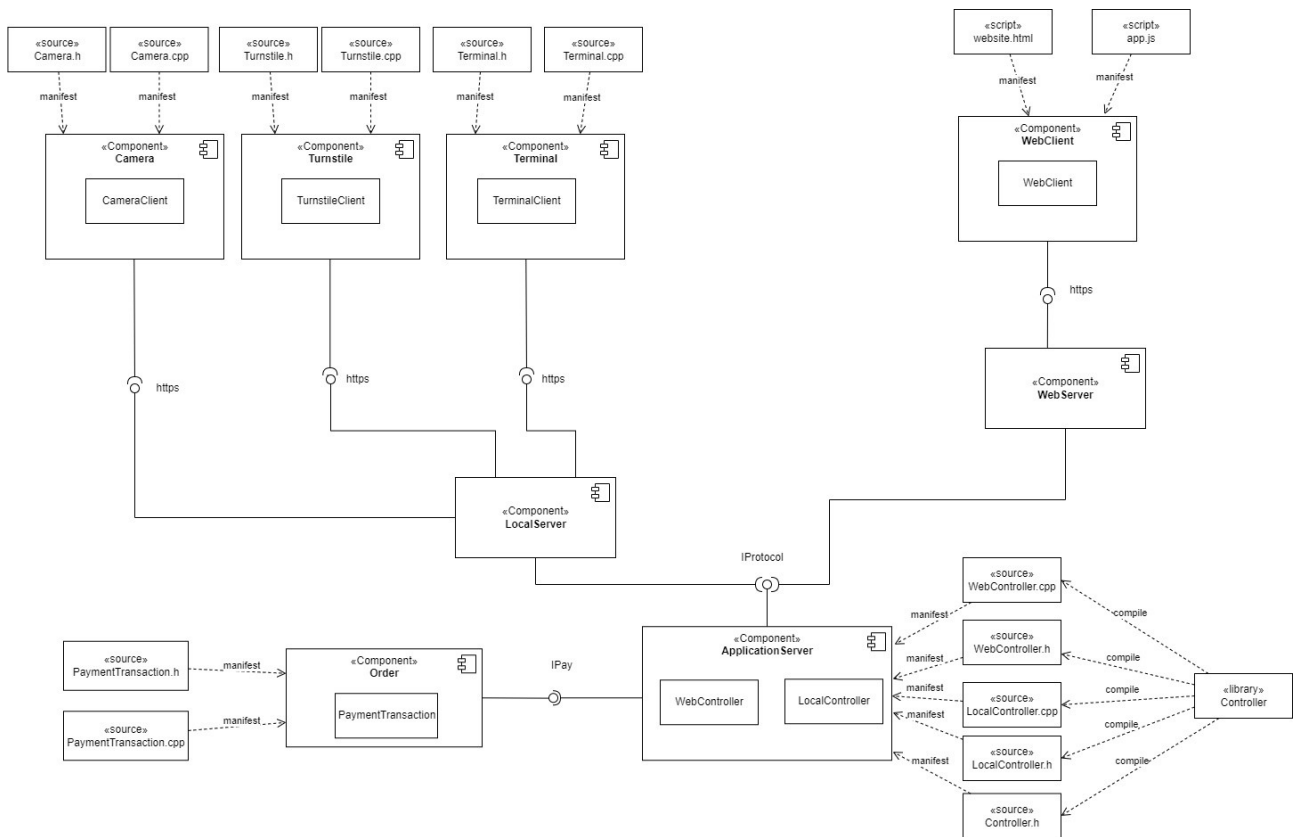


Рисунок 10: Диаграмма компонентов

Описание:

На данной диаграмме (см. рис. 10) показаны особенности физического представления системы. Компонент **WebClient**, представляющий клиентскую сторону веб-приложения, общается с компонентом **WebServer**, который возвращает клиенту ответы в виде html-страниц. Компоненты **Camera**, **Turnstile**, **Terminal** объединяются в одну локальную сеть с **LocalServer**. **LocalServer** и **WebServer** взаимодействуют с **ApplicationServer** для обработки запросов. Для взаимодействия с платежными системами **ApplicationServer** использует компонент **Order**.

Диаграмма развертывания

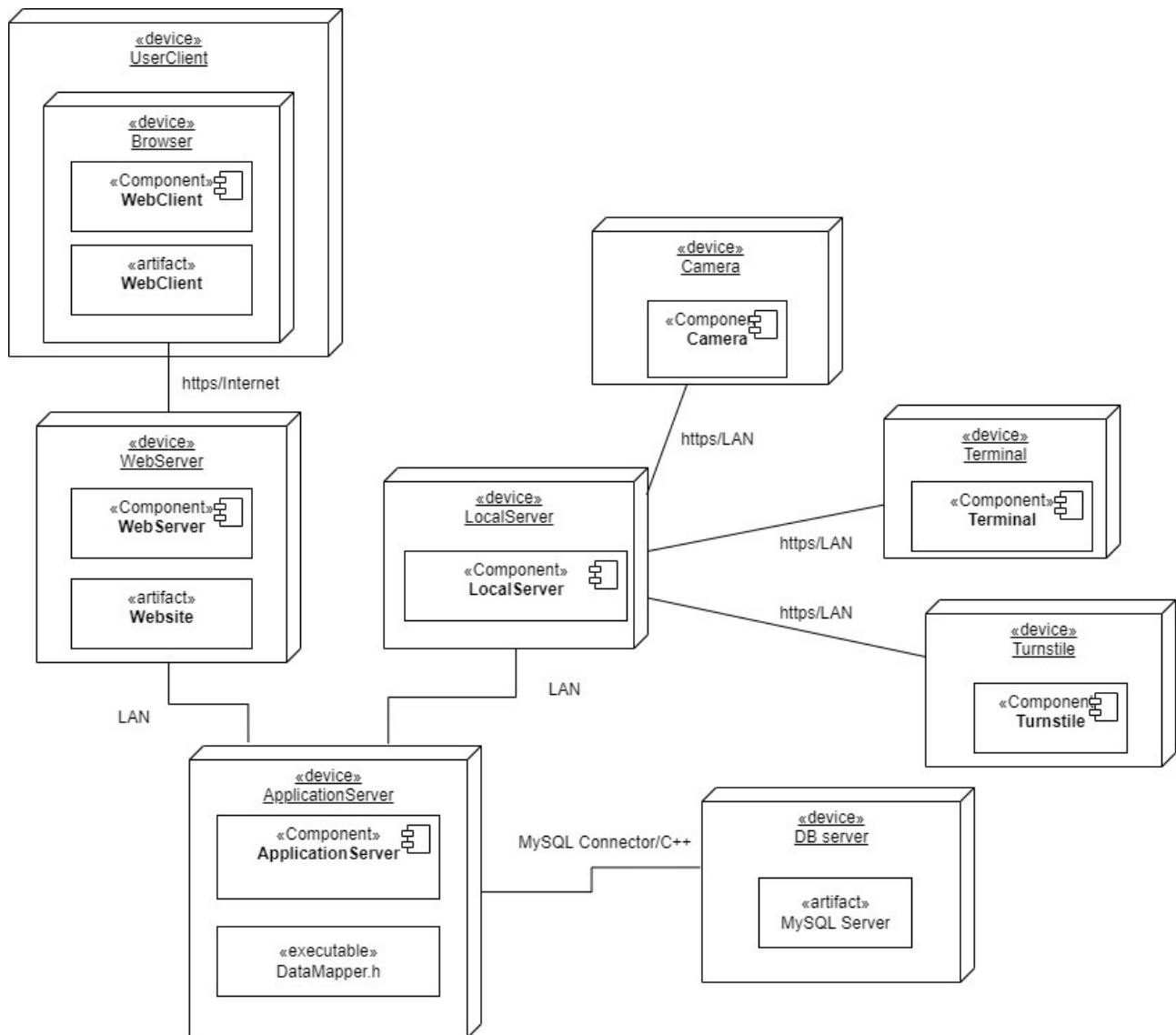


Рисунок 11: Диаграмма развертывания

Описание:

На данной диаграмме (см. рис. 11) показаны как компоненты будут размещаться в физической структуре. UserClient по Internet соединяется с WebServer. WebServer, ApplicationServer, LocalServer, Camera, Terminal, Turnstile образуют единую закрытую локальную сеть. Обращение к базе данных MySQL происходит с помощью MySQL Connector/C++.

Диаграмма пакетов

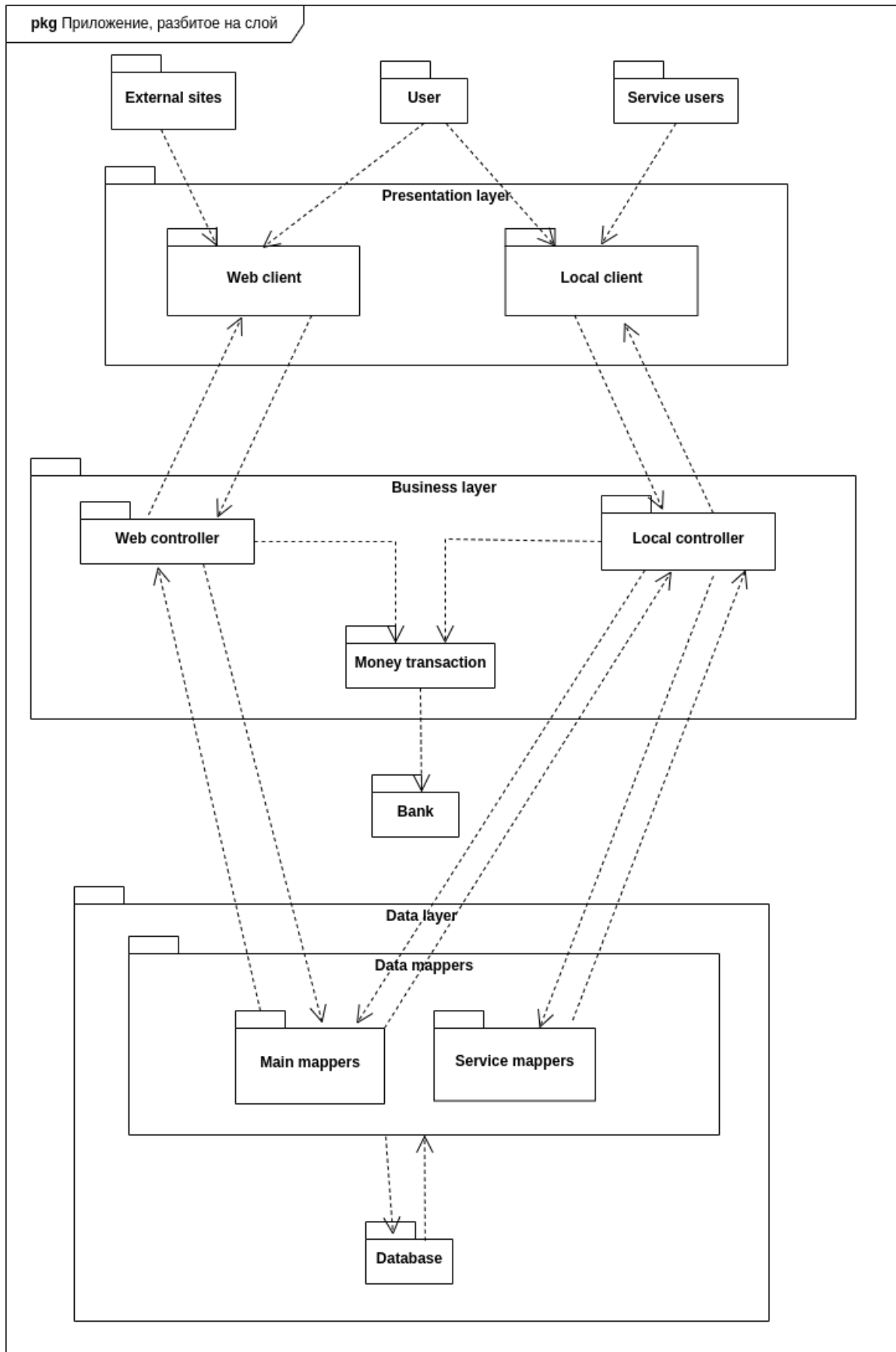


Рисунок 12: Диаграмма пакетов

Описание:

Вся система разбита на слой: представление (Presentation), бизнес-логика (Business logic) и данные (Data).

External sites — сторонние сайты для покупки билетов, которые взаимодействуют с разрабатываемой системой с помощью пакета Web client. User — пользователь, который зашел на сайт стадиона для покупки билета. Service users — служебные пользователи: терминалы, камеры и турникеты, которые взаимодействуют с контроллером с помощью локального клиента.

Web controller и Local controller содержат контроллеры для взаимодействия по Интернету и по локальной сети соответственно. Money transaction содержит модули для выполнения банковских транзакции. Bank — банк, с которым происходит взаимодействие во время транзакции.

Data mappers — пакет, содержащий модули для «маппинга» записей из БД в экземпляры соответствующих классов. Main mappers - «мапперы», которыми пользуются обычные пользователи, Service mappers — служебные «мапперы», с которыми в основном работают служебные устройства: турникеты, камеры и терминалы. Database — пакет, содержащий модули для работы с БД.