

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет по курсу лабораторных работ

Тема: Разработка одностраничного веб-приложения

Дисциплина: ПО распределенных вычислительных систем

Выполнил студент гр. 23541/3

Жемелев Г. А.

(подпись)

Руководитель

Стручков И. В.

(подпись)

“21” декабря 2018 г.

Санкт-Петербург
2018

Содержание

1. Анализ задания	3
1.1. Требования к системе.....	3
1.2. Основные варианты использования.....	3
1.2.1. Подача заявки на аренду ячейки.....	3
1.2.2. Положить/изъять ценность.....	4
1.2.3. Прием заявки на аренду ячейки	5
1.3. Описание модели предметной области	5
2. Разработка серверной части приложения.....	6
2.1. Сущностные классы	6
2.2. Слой хранения.....	7
2.3. Слой бизнес-логики.....	8
2.4. Сервисный уровень и REST API	10
2.5. Конфигурационные классы	12
2.6. Описание основных сценариев функционирования	13
3. Разработка клиентской части приложения.....	15
4. Тексты программ.....	17
5. Методика и результаты тестирования	17
5.1. Вход в приложение	19
5.2. Подача заявки на аренду ячейки	24
5.3. Оплата аренды.....	28
5.4. Манипуляции с ценностями	30
6. Инструкция системного администратора.....	34
7. Инструкция пользователя.....	35
8. Выводы.....	35

1. Анализ задания

Разрабатываемая система «Банковское хранилище» (Bank Vault) предназначена для автоматизации оказания услуг в сфере хранения ценностей в хранилище некоторого банка и предоставляет пользователям возможность аренды одной или нескольких ячеек (возможно разных размеров) и хранения в них ценностей.

1.1. Требования к системе

Клиента «Банковского хранилища» интересуют возможности:

- подать заявку на получение банковской ячейки;
- осуществить выбор ячейки из предложенного размерного ряда;
- оплатить аренду ячейки при одобрении заявки;
- манипулировать ценностями в ячейках (класть/извлекать).

Менеджера банка интересуют возможности:

- рассматривать заявки от клиентов, одобрять или отклонять их;
- просматривать персональные данные клиентов, подавших заявки;
- принимать оплату аренды ячеек от клиентов;

Разрабатываемое приложение должно представлять из себя web-сервис, где клиентская часть выполнена в виде одностраничного web-приложения (single page application, SPA), которое взаимодействует с серверной частью с помощью предоставляемого им REST API по протоколу HTTP.

1.2. Основные варианты использования

Основные варианты использования имеют вид, представленный с помощью диаграмм прецедентов на рис. 1 и рис. 2 для клиента и менеджера соответственно. Текстовое описание каждого варианта использования приведено ниже.

1.2.1. Подача заявки на аренду ячейки

1. Клиент обращается к системе для получения банковской ячейки.
2. Клиент предоставляет свои паспортные данные.
3. Клиент выбирает необходимый размер ячейки.
4. Клиент оплачивает аренду: наличными или банковской картой.
5. Система подтверждает оплату аренды.

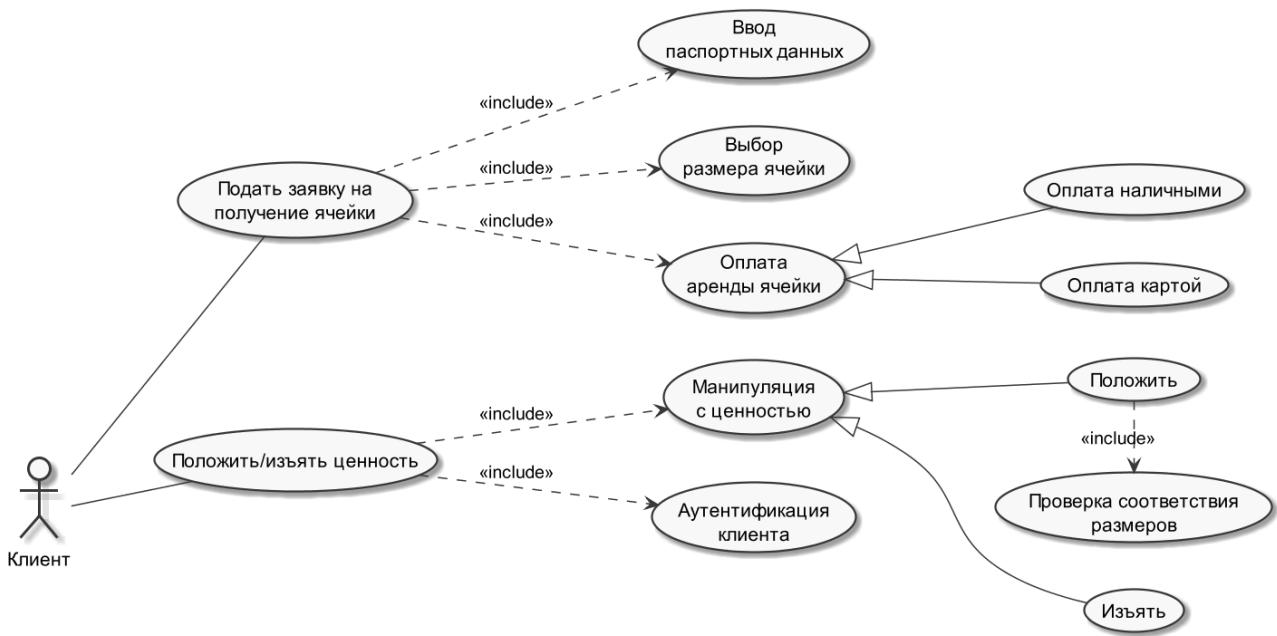


Рис. 1. Диаграмма прецедентов для роли «Клиент»

Альтернатива: *Некорректные паспортные данные:* На шаге 2 устанавливается, что введенные паспортные данные некорректны. Клиенту предоставляется возможность повторного ввода паспортных данных.

Альтернатива: *Нет свободной ячейки подходящего размера:* На шаге 3 обнаруживается, что в хранилище нет свободных ячеек запрошенного Клиентом размера.

За. Клиент выбирает другой размер ячейки – переход к шагу 3.

Зб. Клиент отказывается от услуг Банковского хранилища.

Альтернатива: *Оплата не прошла:* На шаге 4 оказывается, что Клиент не может оплатить услуги Банковского хранилища по той или иной причине. Клиенту предлагается поменять способ оплаты или изменить/исправить данные банковской карты.

1.2.2. Положить/изъять ценность

1. Клиент обращается к системе с целью положить/изъять ценность.
2. Клиенту предоставляется список его ячеек.
3. Клиент выбирает ячейку, в которую он хочет положить/изъять ценность.
4. Клиент кладет/изымает ценность в свою ячейку.

Альтернатива: Несоответствие размеров: На шаге 4 оказывается, что ценность по своим габаритам превосходит размеры ячейки и не может быть помещена в неё. Система сообщает об этом клиенту и предлагает отменить операцию или выбрать другую ячейку (переход к шагу 3).

1.2.3. Прием заявки на аренду ячейки

1. Менеджер получает от Клиента заявку на аренду банковской ячейки.
2. Менеджер просматривает паспортные данные клиента и параметры аренды, проверяет их, одобряет заявку клиента и переводит её в состояние ожидающей оплаты.
3. Менеджер отклоняет заявку, если проверки на шаге 2 не прошли.
4. Система проверяет кредитную историю Клиента, и если она хорошая (т.е. все выплаты по кредитам осуществляются в срок), то клиенту дается скидка.
5. После подтверждения системой оплаты выбранной ячейки Менеджер добавляет её в список банковских ячеек, закрепленных за Клиентом.

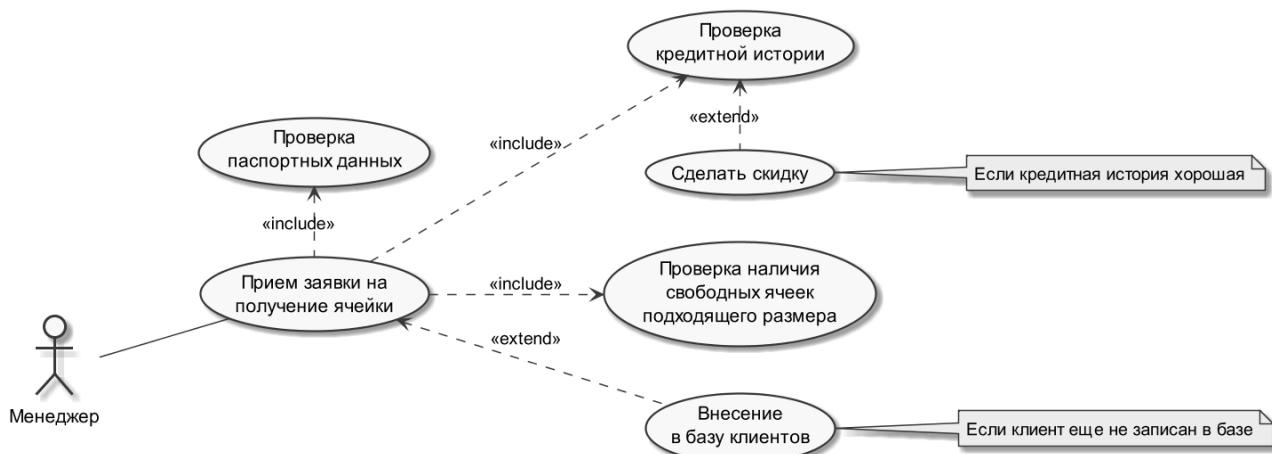


Рис. 2. Диаграмма прецедентов для роли «Менеджер»

1.3. Описание модели предметной области

Статическая модель предметной области построена при помощи диаграммы классов UML и представлена на рис. 3. Помимо сущностей, упомянутых выше, были также добавлены «Журнал манипуляций» и «Запись журнала» для возможности мониторинга действий с ячейками хранилища.

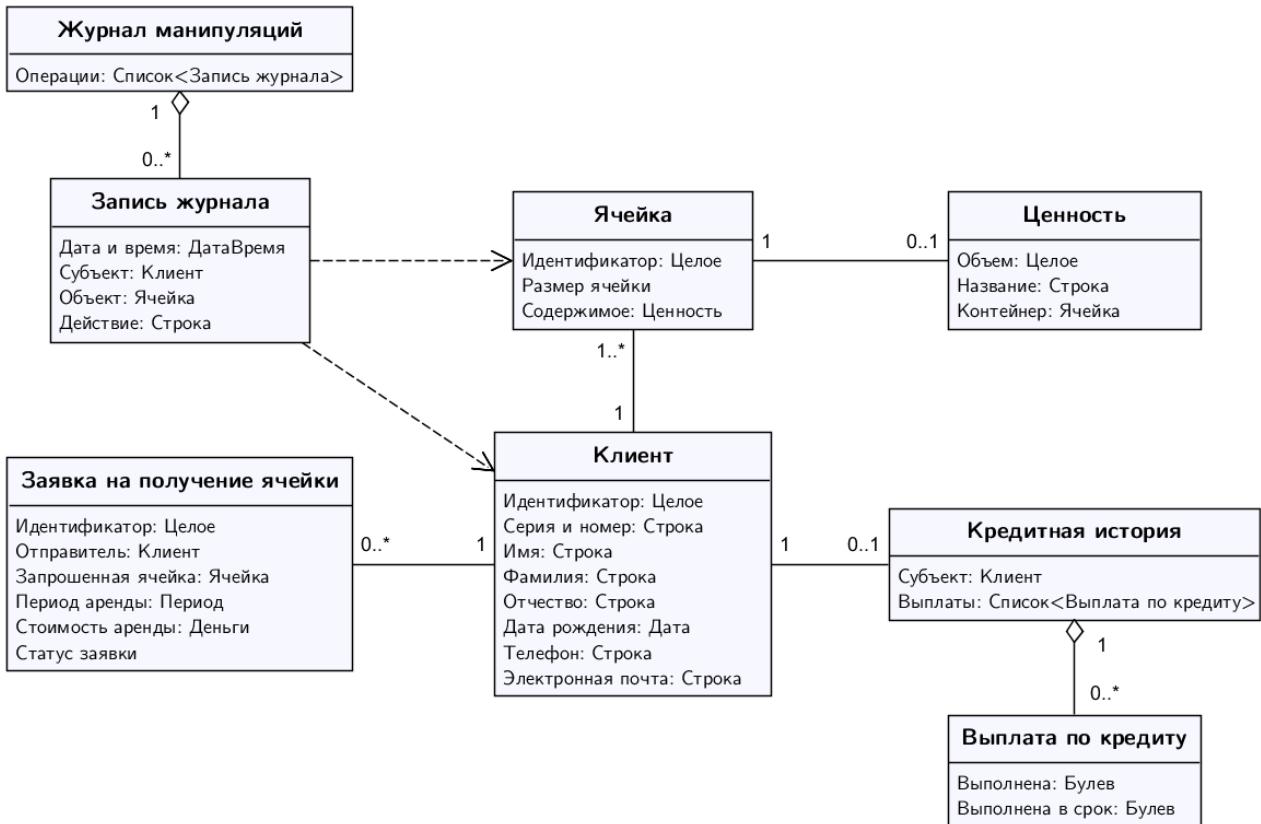


Рис. 3. Описание модели предметной области

2. Разработка серверной части приложения

Объектно-ориентированное проектирование системы выполнялось с учетом особенностей технологии «Spring Framework» (с использованием Spring Boot версии 2.0.4-RELEASE), следуя архитектурному принципу расслоения и клиент-серверной ориентированности системы.

2.1. Сущностные классы

В соответствии с описанием модели предметной области были реализованы сущностные классы, приведенные на рис. 4.

Каждый экземпляр класса Cell описывает банковскую ячейку и включает в себя экземпляр класса CellLeaseRecord, предназначенный для фиксации состояния аренды ячейки, а также может содержать экземпляр класса Precious, который описывает ценность, хранящуюся в ячейке.

Класс CellApplication соответствует заявке на аренду банковской ячейки и ссылается на подавшего эту заявку клиента, представленного классом Client.

Персональные данные клиента вынесены в отдельный класс PassportInfo, экземпляры которого содержатся в объектах типа Client.

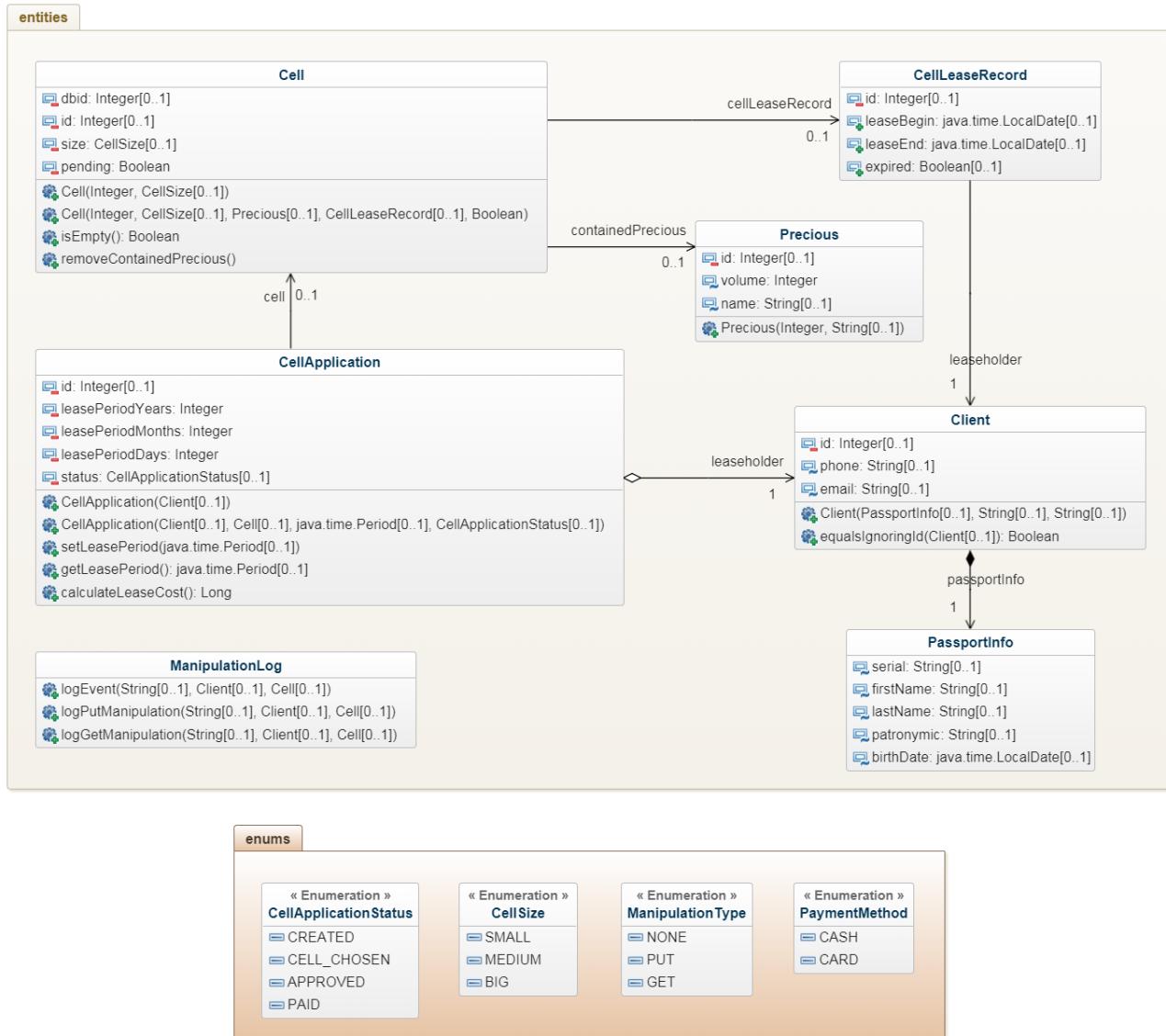


Рис. 4. UML-диаграмма сущностных классов

2.2. Слой хранения

UML-диаграмма классов для слоя хранения данных, используемого в системе, приведена на рис. 5. При реализации этого слоя использовался шаблон проектирования «Хранилище» (Repository), реализованный с помощью интерфейса **JpaRepository**, предоставляемого компонентом Spring Data, входящего в состав технологии Spring Boot.

Собственные интерфейсы JpaCellsRepository, JpaClientsRepository и JpaApplicationsRepository расширяют JpaRepository с соответствующими шаблонными

параметрами, а реализации для них генерируется Spring Data автоматически. Кроме того, эти интерфейсы наследуют CellsRepository, ClientsRepository и ApplicationsRepository, которые задают необходимое и достаточное поведение для реализации функциональности слоя хранения в разработанной системе. Наличие этих интерфейсов позволило создать реализации InMemoryCellsRepository, InMemoryClientsRepository и InMemoryApplicationsRepository, которые используются для модульного тестирования приложения.



Рис. 5. UML-диаграмма классов слоя хранения

2.3. Слой бизнес-логики

UML-диаграмма классов слоя бизнес-логики приведена на рис. 6.

Класс VaultHardware отвечает за создание экземпляров банковских ячеек

при их отсутствии и осуществляют контроль их физического состояния (открыты или закрыты).

Класс Vault предназначен для выполнения запросов на ячейки интересующего размера, а также выполняет контроль за бронированием ячеек на время осуществления операций бизнес-процесса «Подача заявки на аренду», чтобы сохранять выбранную ячейку закрепленной за клиентом в течение времени, достаточного для одобрения заявки менеджером и последующей оплаты клиентом.

Классы CellApplicationInteractor и CellManipulationInteractor реализуют конкретные сценарии использования, описанные в п. 1 и используют класс LeasingController, который отвечает за контроль аренды всех ячеек (подсчет времени, фиксация начала и окончания аренды).

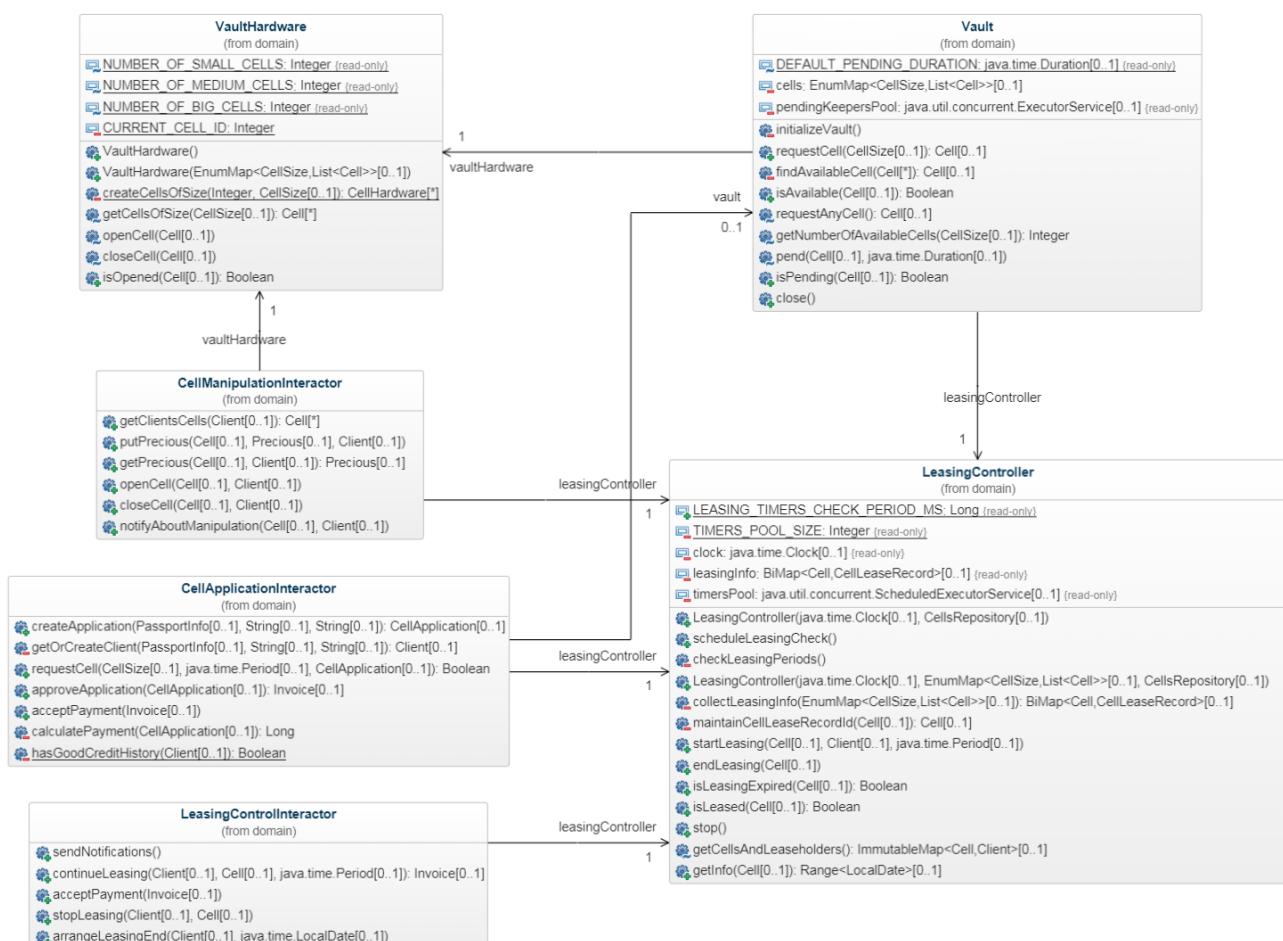


Рис. 6. UML-диаграмма классов слоя бизнес-логики

Все вышеупомянутые классы, кроме LeasingController и VaultHardware, яв-

ляются компонентами (@Component) в терминологии Spring Framework, существуют в единственном экземпляре и подставляются в поля использующих их классов с помощью механизма **внедрения зависимостей** (Dependency Injection). Единственные объекты классов LeasingController и VaultHardware конструируется как @Bean'ы, потому что для них используются разные конструкторы в зависимости от состояния CellsRepository. Сами хранилища также внедряются в объекты бизнес-логики с помощью аннотации @Autowired.

Кроме того, в пакете domain присутствуют утилитарные классы PriceCalculator, ClientPassportValidator, PutManipulationValidator (не показаны на диаграмме), статические методы которых используются в других классах для вычисления стоимости аренды, проверки корректности паспортных данных клиентов и допустимости манипуляций с ячейками соответственно.

2.4. Сервисный уровень и REST API

Над слоем бизнес-логики расположен сервисный уровень (рис. 7), включающий в себя следующие классы:

BankVaultFacade – реализует шаблон проектирования «Фасад», позволяя таким образом скрыть сложность системы путём сведения всех внешних вызовов к этому объекту, делегирующему их соответствующим объектам системы.

LoginService – сервис, необходимый для аутентификации пользователей.

PaymentService – сервис, предназначенный для обработки платежей (эмитирует внешнюю платёжную систему).

Данные классы помечены аннотацией @Service и используются в классах, реализующих REST API разработанного приложения:

CellApplicationsRestController – для подачи заявок на аренду ячеек.

CellInformationRestController – для получения информации о ячейках.

CellManagementRestController – для одобрения/отклонения заявок.

CellManipulationsRestController – для работы с ценностями и ячейками.

LoginRestController – для авторизации пользователей.

PaymentRestController – для приема оплаты ячеек.

Эти классы помечены как @RestController и им сопоставлены URL.

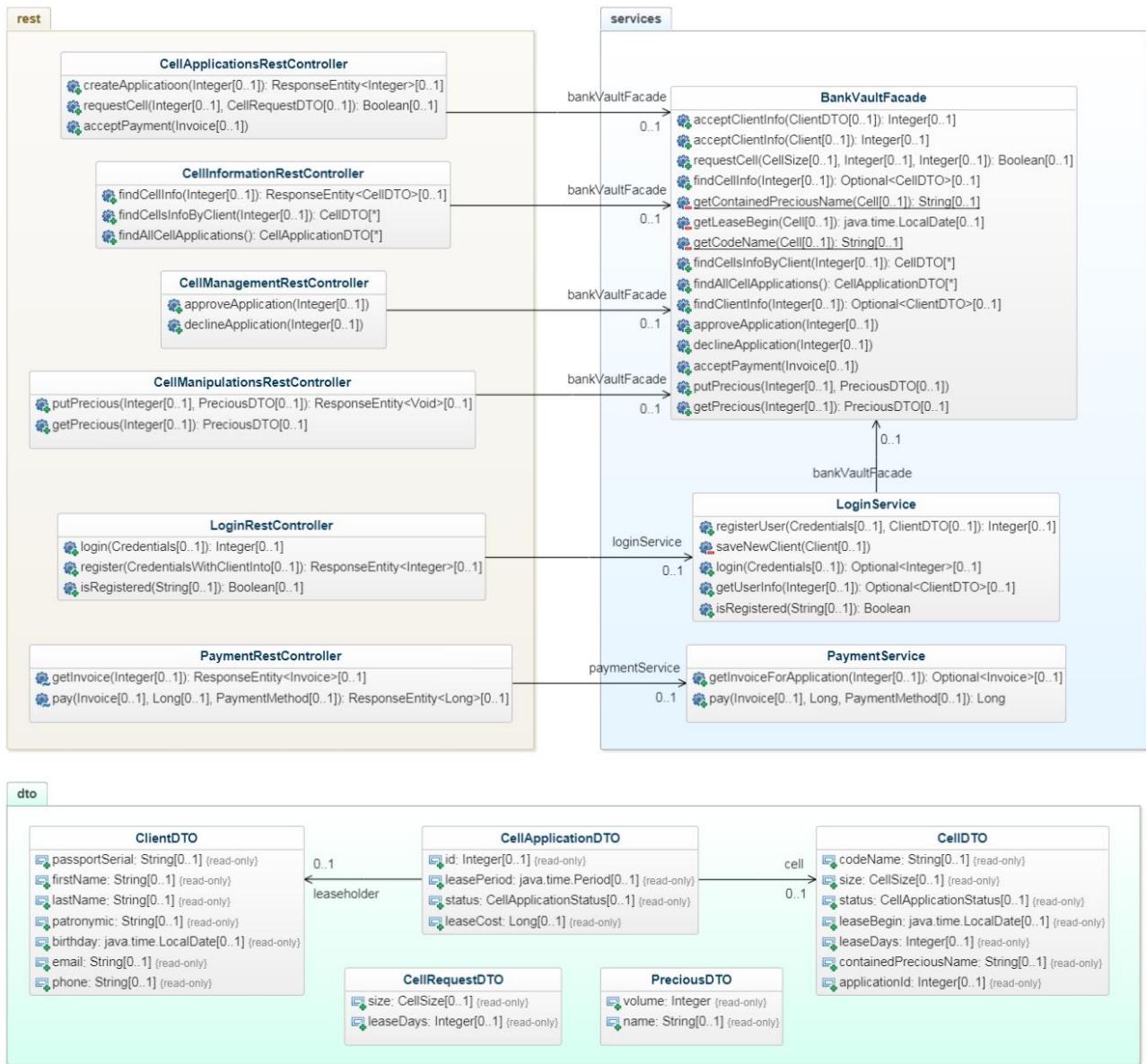


Рис. 7. UML-диаграмма классов сервисного уровня и REST API

Обмен информацией на этом уровне осуществляется с помощью классов, реализующих шаблон Data Transfer Object (DTO).

Кроме того, для реализации функциональности по авторизации пользователей и эмуляции работы внешней платежной системы были созданы классы, объединенные в пакеты external и recognition (рис. 8):

PaymentSystem – интерфейс эмулируемой платежной системы.

Invoice – счёт на оплату, выставляемый за аренду ячейки.

SimplePaymentSystem – in-memory реализация платежной системы.

FileBasedPaymentSystem – расширение SimplePaymentSystem, позволяющее сохранять информацию о счетах в виде файлов на жестком диске.

UserStorage – класс для хранения имен пользователей и хэш-кодов их паролей в файле на жестком диске.

Credentials – класс для передачи имени пользователя и пароля в UserStorage.

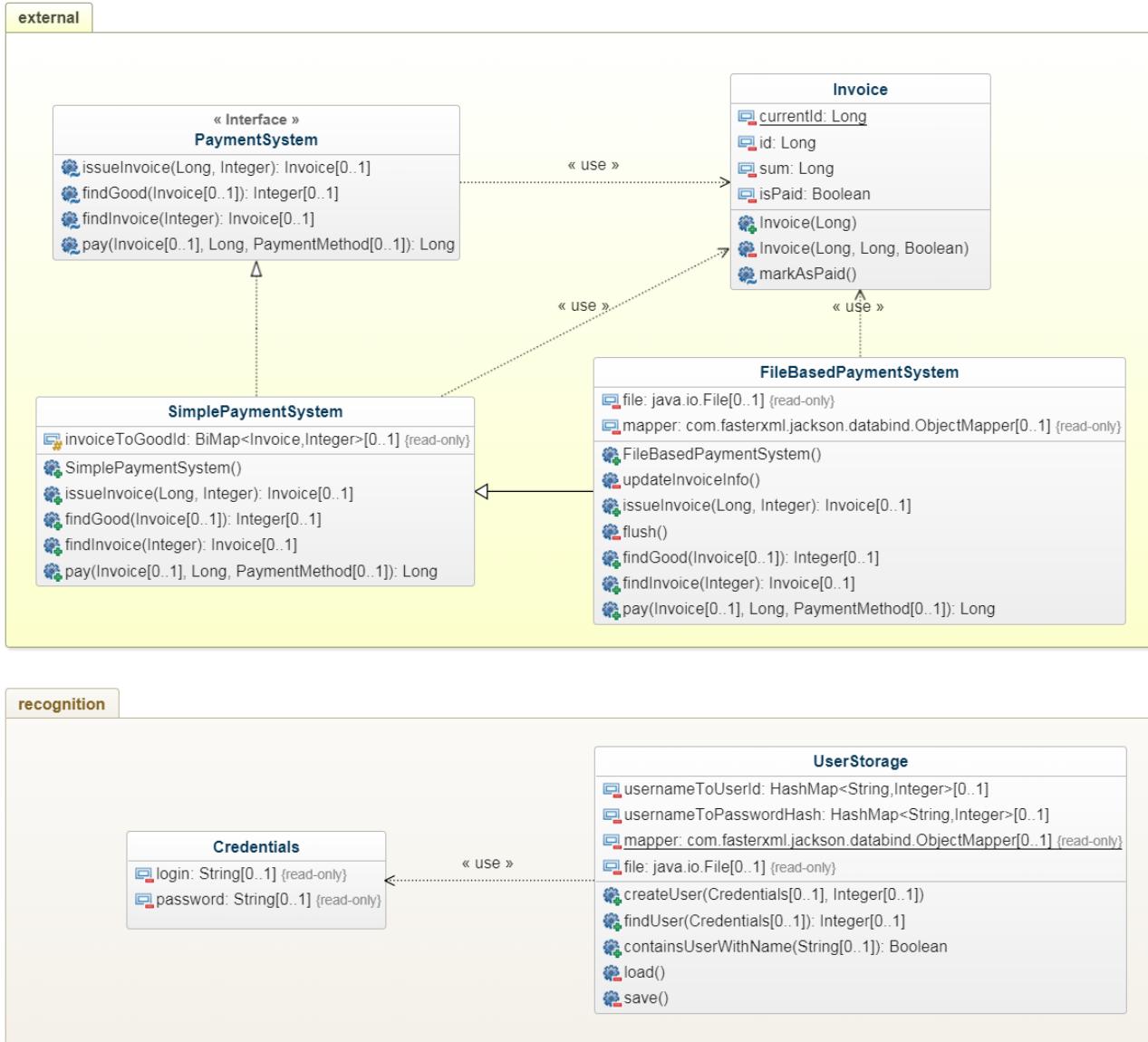


Рис. 8. UML-диаграмма вспомогательных классов

2.5. Конфигурационные классы

Для конфигурации Spring Boot также были созданы специальные классы, объединенные в пакете config (рис. 9).

Здесь классы CommonConfig и VaultConfig служат для создания объектов-бинов, SwaggerConfig – для настройки autogenerated веб-интерфейса Swagger 2 (полезного для отладки REST API), а WebConfig – для задания пути статических ресурсов, перенаправления с корневого URL и параметров CORS.

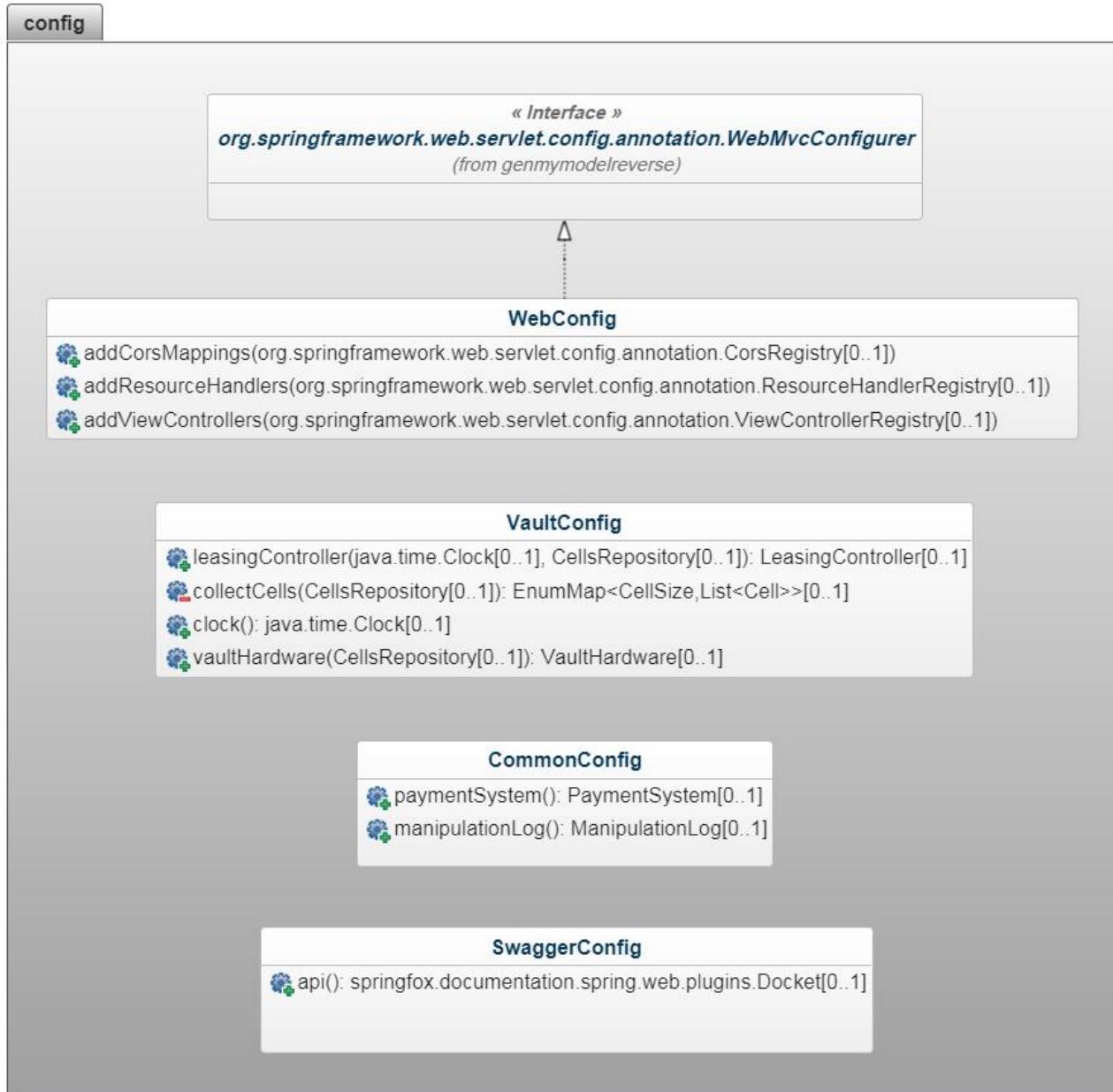


Рис. 9. UML-диаграммы конфигурационных классов

Точкой входа в программу является класс BankVaultRunner (не показан на диаграммах), вызывающий в своем методе `main()` метод `start()` класса BankVaultCoreApplication, в котором производится создание контекста приложения Spring Boot и его запуск.

2.6. Описание основных сценариев функционирования

Описание наиболее сложного сценария функционирования «Подача заявки на получение ячейки» было выполнено с помощью диаграммы последовательности UML (рис. 10):

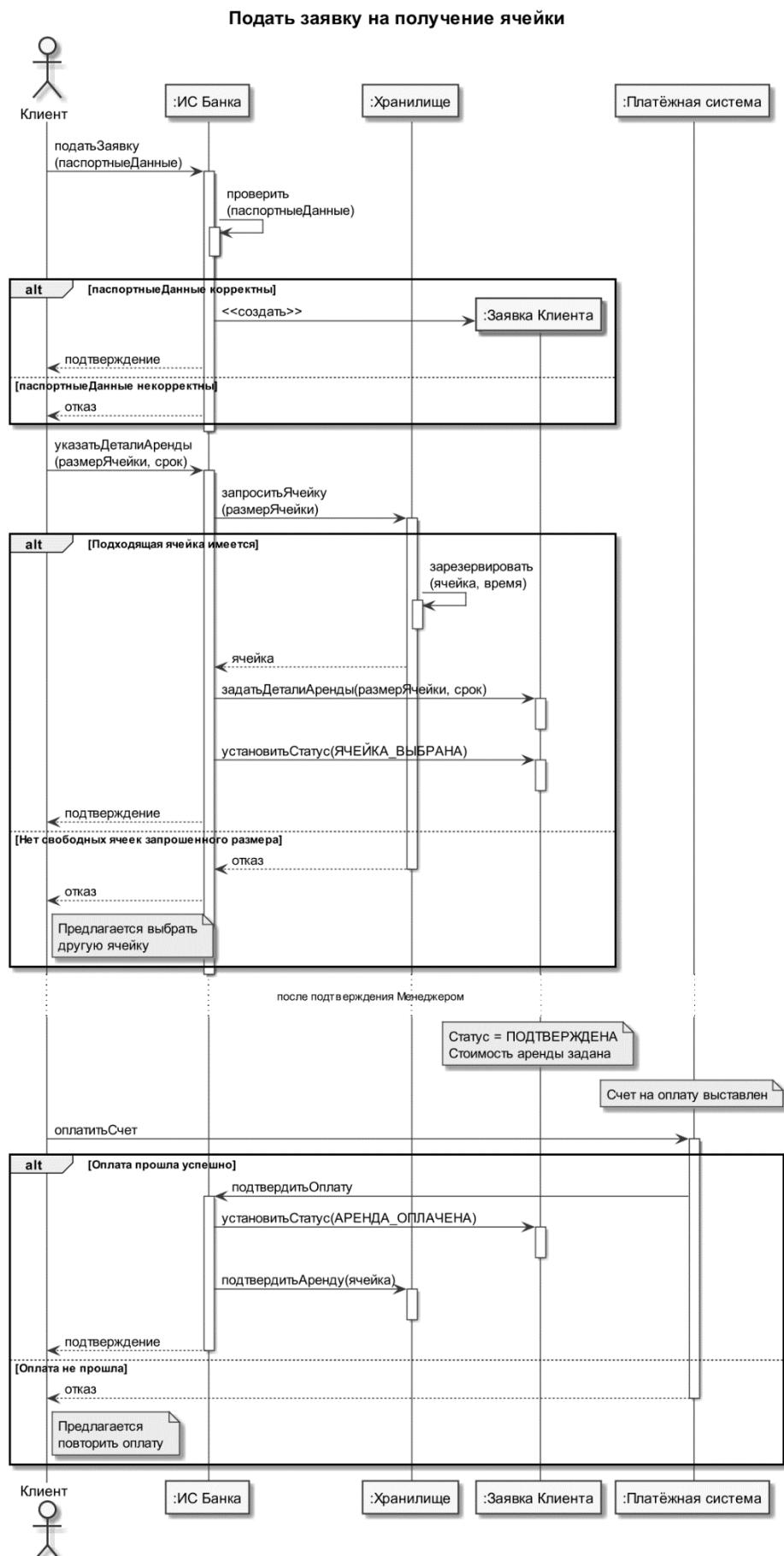


Рис. 10. UML-диаграмма последовательности основного сценария

3. Разработка клиентской части приложения

Для создания одностраничного веб-приложения, являющегося клиентской частью разрабатываемой системы, был выбран язык Dart (компилирующийся в JavaScript) и технология AngularDart версии 5.1.0.

При использовании этой технологии приложения составляется из функционально законченных компонентов, каждому из которых может сопоставляться шаблон веб-страницы в виде HTML-разметки и/или каскадная таблица стилей CSS. Шаблоны могут включать в себя директивы, позволяющие связывать наполнение веб-страниц с полями соответствующих классов компонентов на языке Dart, а также устанавливать обработчики событий, которые дают возможность реагировать на действия пользователя.

Переходы между компонентами осуществляются с помощью так называемого роутера – специального объекта Angular, перехватывающего переходы между различными URL и использующего получаемую информацию для замены одних компонентов на другие путем динамической перестройки веб-страницы. Таким образом достигается работа в режиме одностраничного приложения.

Компоненты реализуют слой представления, а также отвечают за работу с вводом-выводом данных и обеспечивают реакцию на действия пользователя. Остальную функциональность реализуют сервисы: в данном случае им делегируется обязанность по взаимодействию с REST API серверной части приложения и ряд других операций.

Полный список сервисов выглядит следующим образом:

- CellApplicationService – для подачи заявки на аренду ячейки.
- CellTableService – для вывода списка ячеек в таблицу.
- LocalUserStorage – для хранения идентификатора текущего пользователя (в т.ч. между сессиями работы с приложением).
- ManagerService – для работы с REST API менеджера.
- ModalFormsService – для взаимодействия между модальными окнами.
- PaymentService – для взаимодействия с платежной системой.

- PreciousManipulationService – для манипуляций с ячейками.
- WebTimeService – для запросов к серверу мирового времени.

Полный список компонентов:

- CellApplicationComponent
- CellTableComponent
- ClientComponent
- ManagerComponent
- PaymentComponent
- PreciousManipulationComponent
- LoginComponent
- RegisterComponent

Все компоненты, кроме последних двух полностью сопоставляются сервисам с аналогичными названиями. Компоненты LoginComponent и RegisterComponent предназначены для управления формами входа и регистрации пользователей системы и работают с UserService.

Каждый сервис, работающий с REST API серверной части приложения хранит ссылку на HTTP-клиент, предоставляемый AngularDart путем внедрения зависимостей, и использует его для асинхронного взаимодействия с сервером.

При получении ответа от сервера, в первую очередь проверяется его HTTP-статус, и, если он соответствует ошибке, выбрасывается пользовательское исключение, которое затем перехватывается вышестоящим компонентом, который, в свою очередь, отвечает за корректное отображение информации об ошибке и предоставление пользователю возможностей согласно описанию альтернатив, из п. 1.

Также стоит отметить, что на клиентской стороне реализованы классы, соответствующие тем DTO, которые отправляются с сервера. Возможности по де-серIALIZации из JSON встроены в язык Dart.

REST интерфейс северной части приложения в клиентском коде задаётся в классе ServerApi в виде строк с фиксированными URL и функций, которые возвращают параметризованные URL для запросов с идентификаторами объектов.

4. Тексты программ

Исходные коды серверной части приложения доступны в публичном репозитории по адресу: <https://github.com/wndrws/bank-vault>.

Исходные коды клиентской части приложения доступны в публичном репозитории по адресу: <https://github.com/wndrws/bank-vault-web>.

5. Методика и результаты тестирования

Тесты, написанные при разработке серверной части приложения с использованием технологии JUnit 5, можно разделить на три группы:

1. Модульные тесты слоя бизнес-логики:

- CellApplicationInteractorTest
- CellManipulationInteractorTest
- LeasingControlInteractorTest
- LeasingControllerTest
- PriceCalculatorTest
- VaultTest

2. Интеграционные тесты слоя хранения данных:

- JpaApplicationsRepositoryTest
- JpaCellsRepositoryTest
- JpaClientsRepositoryTest

3. Функциональные тесты сценариев использования:

- ApplyForCellTest
- LeasingExpiryTest
- ManipulationTest

Тесты первой группы проверяют правильность работы алгоритмов, применяемых при реализации бизнес-логики программы. Тесты второй группы – корректность схемы базы данных с точки зрения её совместимости с существующими объектами, которые используются в системе. Третья группа тестов покрывает все основные комбинации входных воздействий пользователя и реакций программы в каждом из сценариев использования приложения.

Ниже представлена часть результатов прохождения тестов (рис. 11), а также показатели покрытия для всех пакетов приложения (рис. 12). В частности, считая количество протестированных методов, слой бизнес-логики покрыт тестами на 86%, слой хранения – на 60% (часть методов слоя хранения является стандартными и не нуждается в тестировании при разработке данного приложения).

В общей сложности были успешно выполнены все 85 тестовых методов. Полный отчет о прохождении тестов доступен в репозитории проекта в файле под названием «Test Results - All_in_bank-vault-core_test.xml».

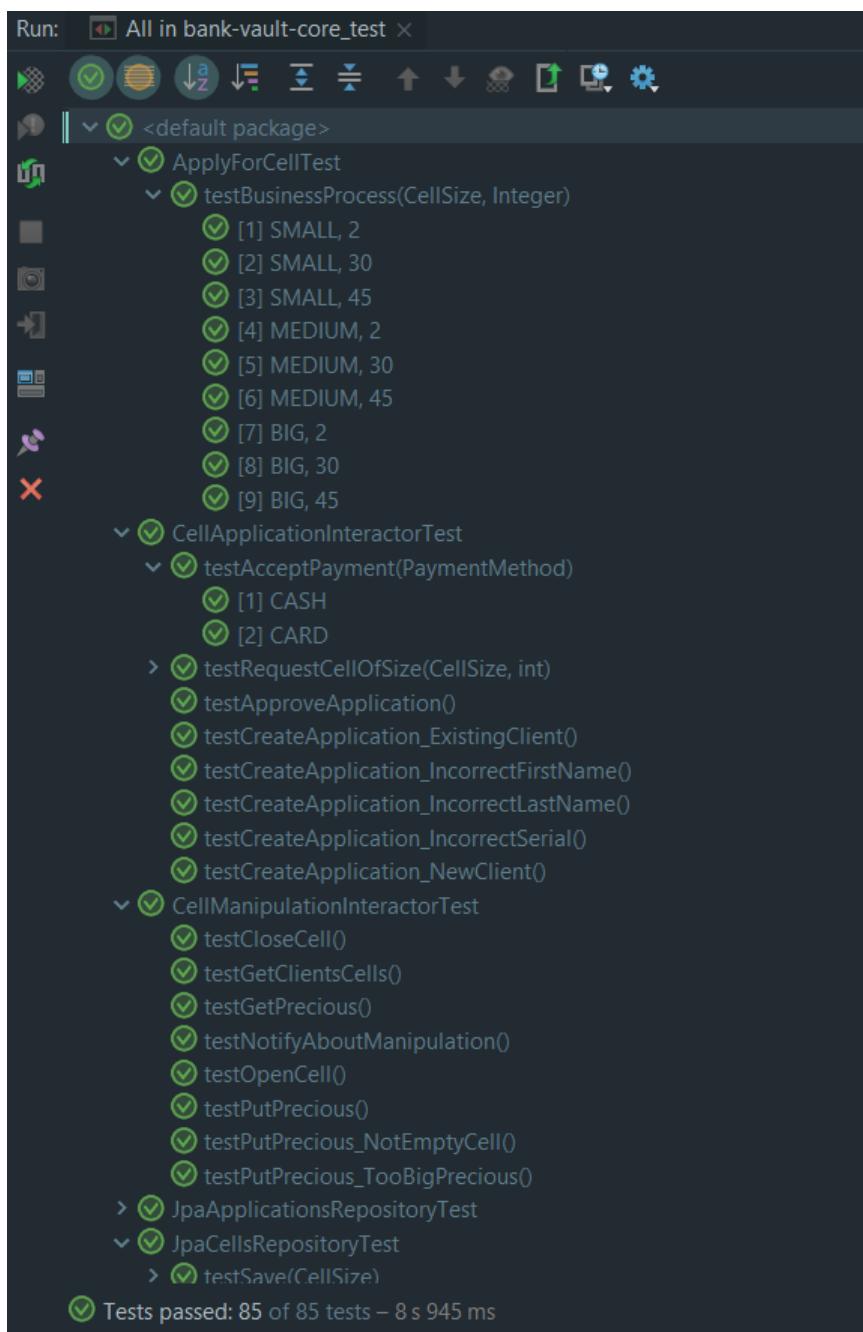


Рис. 11. Результаты тестирования серверной части приложения

Coverage: All in bank-vault-core_test ×			
50% classes, 49% lines covered in package 'bank'			
Element	Class, %	Method, %	Line, %
boundaries	100% (0/0)	100% (0/0)	100% (0/0)
config	25% (1/4)	30% (3/10)	23% (6/26)
dao	100% (6/6)	60% (17/28)	57% (48/83)
domain	90% (18/20)	86% (126/145)	82% (321/387)
dto	0% (0/5)	0% (0/29)	0% (0/29)
enums	75% (3/4)	77% (7/9)	77% (7/9)
external	33% (2/6)	40% (12/30)	29% (23/79)
recognition	0% (0/2)	0% (0/10)	0% (0/39)
rest	0% (0/7)	0% (0/25)	0% (0/46)
services	12% (1/8)	0% (0/48)	0% (1/123)
BankVaultCoreApplication	100% (1/1)	40% (2/5)	33% (4/12)
BankVaultRunner	0% (0/1)	0% (0/1)	0% (0/2)

Рис. 12. Показатели покрытия тестами для серверной части приложения

Тестируемая клиентская часть приложения осуществлялась вручную путем воспроизведения основных комбинаций входных воздействий пользователя. Результаты тестирования приведены далее на рис. 13-42.

5.1. Вход в приложение

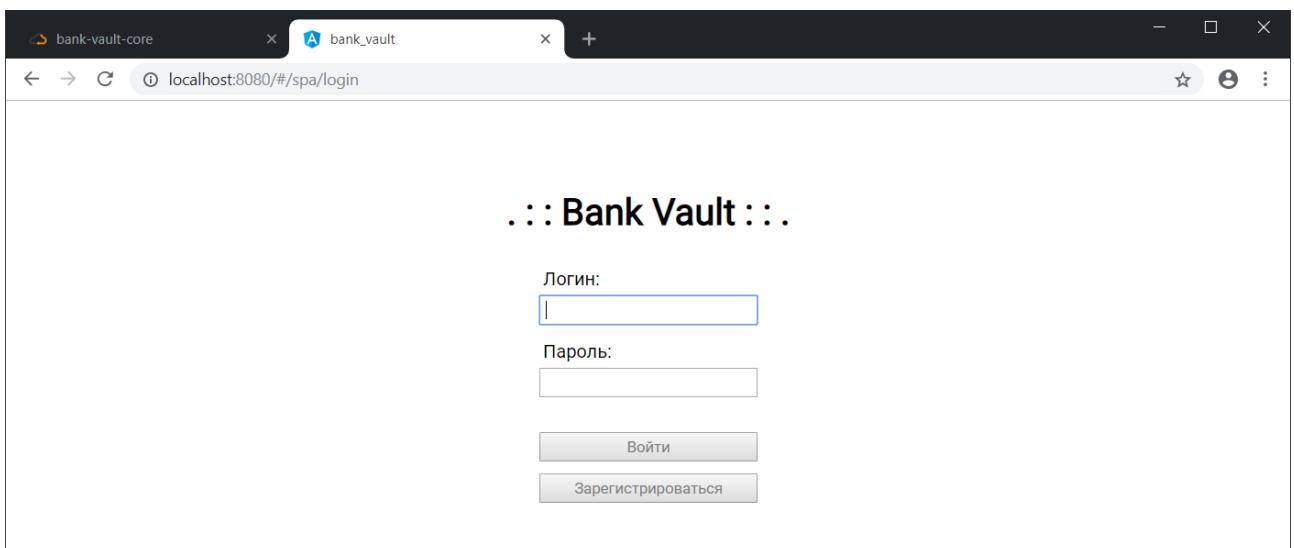


Рис. 13. Форма входа приложения:

пока имя пользователя и пароль не введены, кнопки не активны

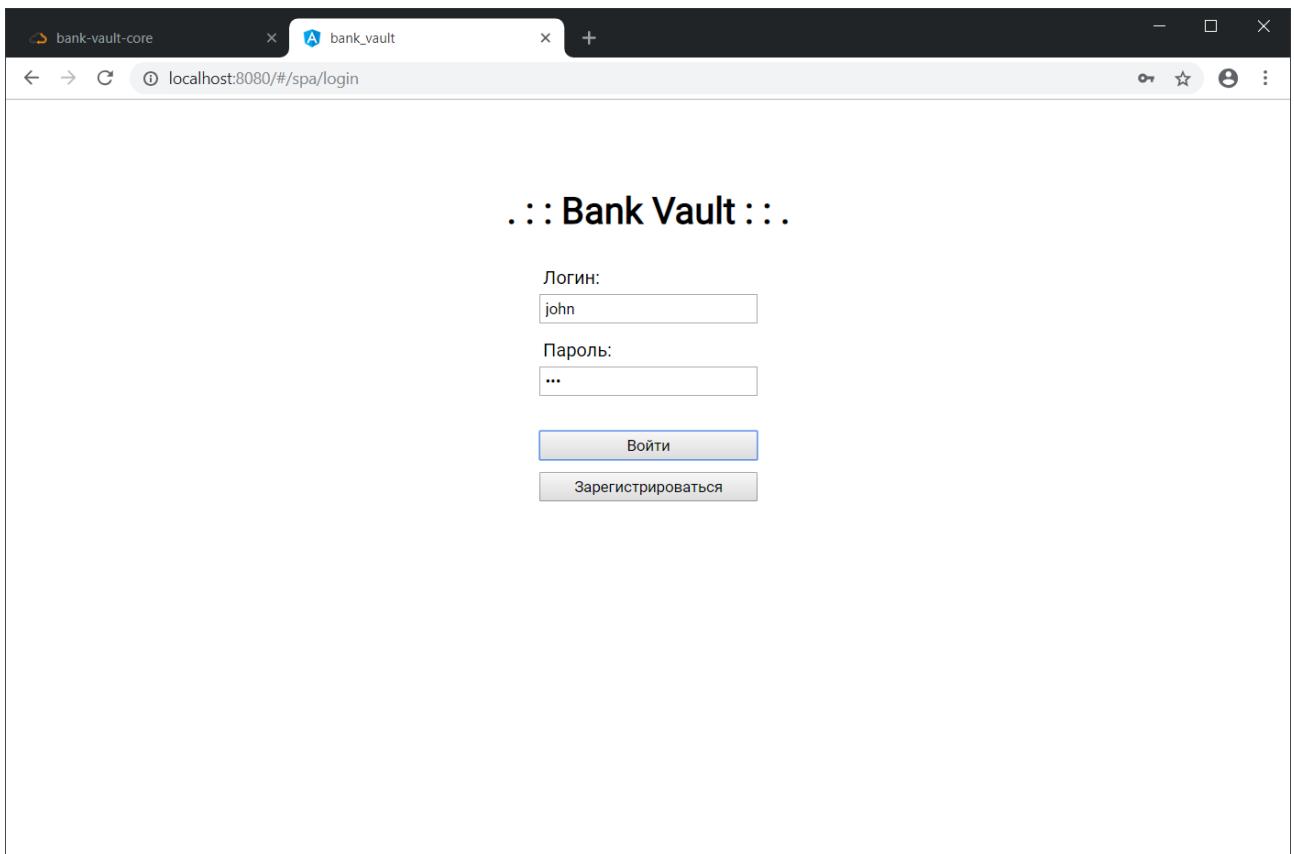


Рис. 14. После ввода учетных данных, можно войти в систему

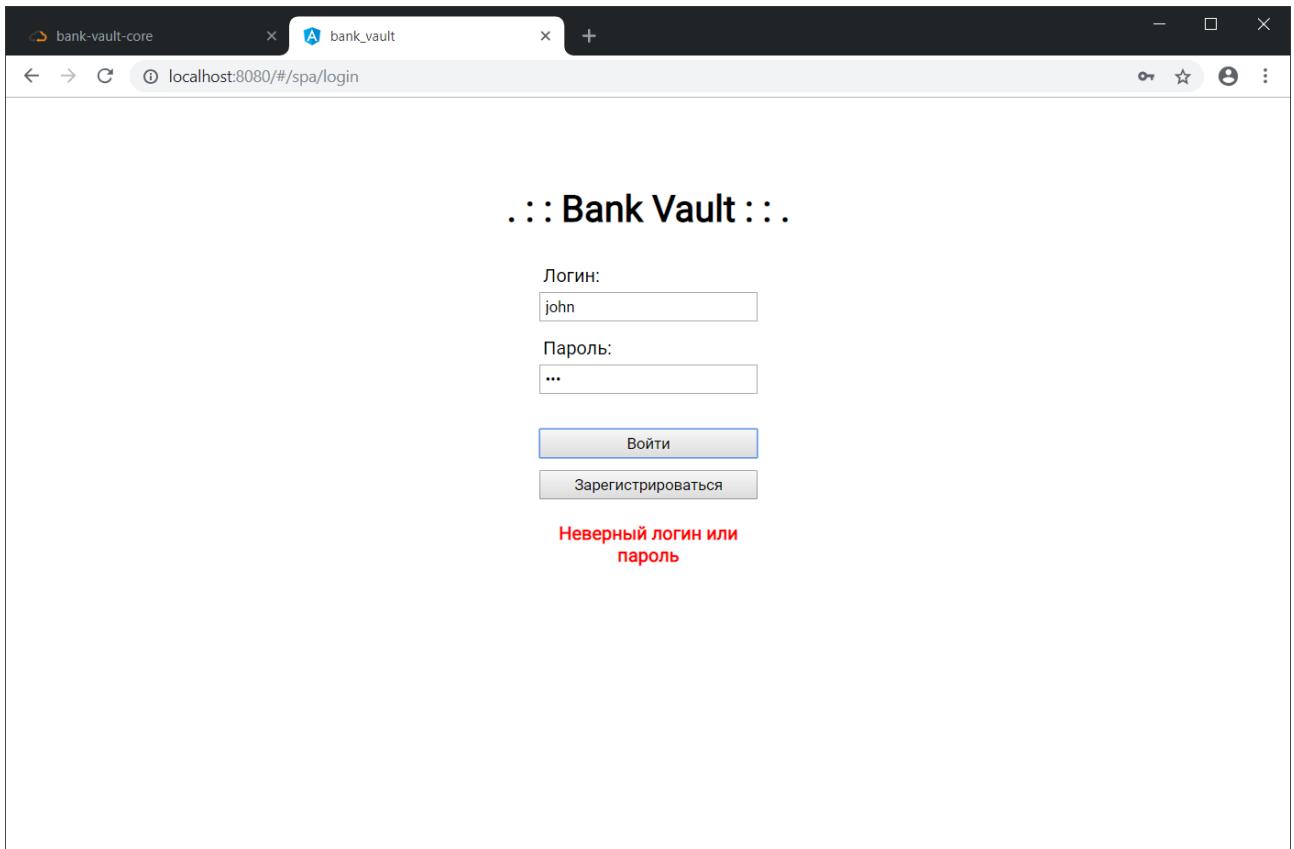


Рис. 15. В случае неправильно введенных данных, выводится сообщение

localhost:8080/#/spa/register

. . . :Bank Vault :. .

Паспортные данные

Серия и номер: | _____

Фамилия: _____

Имя: _____

Отчество: _____

Дата рождения: Select a date

Контактные данные

Телефон: _____

E-mail: _____

ОТМЕНА OK

Рис. 16. Форма регистрации нового пользователя

localhost:8080/#/spa/register

. . . :Bank Vault :. .

Паспортные данные

Серия и номер: 1234

Фамилия: Жемчугов

Имя: Георгий

Отчество: Александрович

Дата рождения:

Enter date

Enter a date

S	M	T	W	T	F	S
30	31					
AUG 1995						
1	2	3	4	5		
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Контактные данные

Телефон: +71234567890

E-mail: wws.dev@gmail.com

ОТМЕНА OK

Рис. 17. Заполнение формы регистрации (часть полей является обязательными)

localhost:8080/#/spa/register

. . . Bank Vault . . .

Паспортные данные

Серия и номер: 1

Фамилия: Жемелев

Имя: Георгий

Отчество: Алексеевич

Дата рождения: Aug 17, 1995

Контактные данные

Телефон: +71234567890

E-mail: wws.dev@gmail.com

ОТМЕНА OK

Рис. 18. Ввод некорректного серийного номера (корректный содержит 9 цифр)

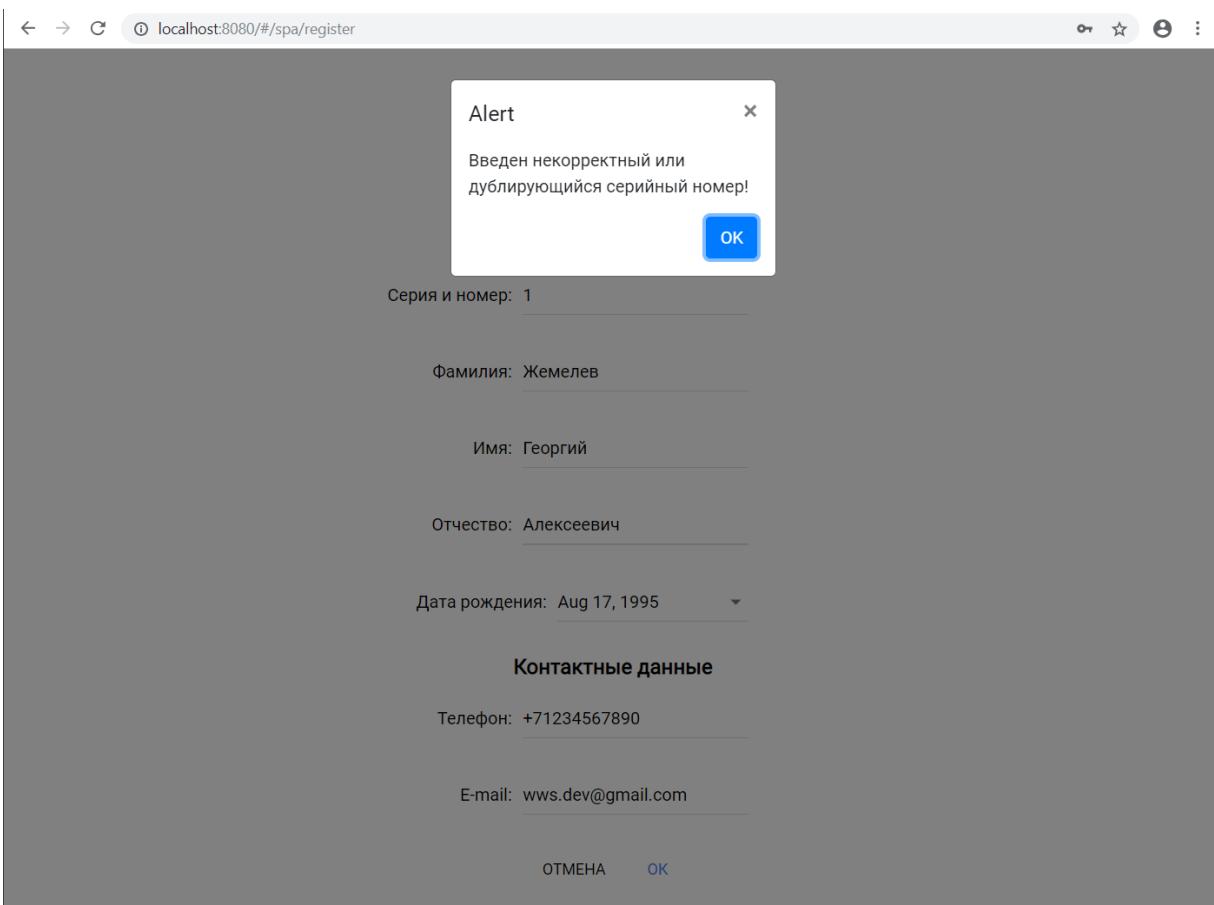


Рис. 19. Сообщение о некорректно введенном серийном номере

bank-vault-core bank_vault

localhost:8080/#/spa/register

.::: Bank Vault :::.

Паспортные данные

Серия и номер: 1234567821

Фамилия: Жемелев

Имя: Георгий

Отчество: Алексеевич

Дата рождения: Aug 17, 1995

Контактные данные

Телефон: +71234567890

E-mail: wws.dev@gmail.com

ОТМЕНА OK

This screenshot shows the 'Register' page of the Bank Vault application. It contains fields for personal information: a valid serial number '1234567821', family name 'Жемелев', first name 'Георгий', middle name 'Алексеевич', birth date 'Aug 17, 1995', phone number '+71234567890', and email 'wws.dev@gmail.com'. There are 'OK' and 'CANCEL' buttons at the bottom right.

Рис. 20. Введен корректный серийный номер

bank-vault-core bank_vault

localhost:8080/#/spa/home

.::: Bank Vault :::.

Мои банковские ячейки 21.12.2018 3:30:04

Номер	Статус	Размер	Содержимое	Начало аренды	Аренда (дней)
Загрузка списка ячеек...					

Арендовать ячейку

Оплатить аренду

Положить ценность

Изъять ценность

Выход

This screenshot shows the 'Home' page of the Bank Vault application after a successful login. It displays a header 'Мои банковские ячейки' and a timestamp '21.12.2018 3:30:04'. Below this is a table with columns: Номер, Статус, Размер, Содержимое, Начало аренды, and Аренда (дней). A message 'Загрузка списка ячеек...' is displayed above the table. At the bottom are buttons for managing deposits: 'Арендовать ячейку', 'Оплатить аренду', 'Положить ценность', 'Изъять ценность', and 'Выход'.

Рис. 21. Успешный вход в систему

5.2. Подача заявки на аренду ячейки

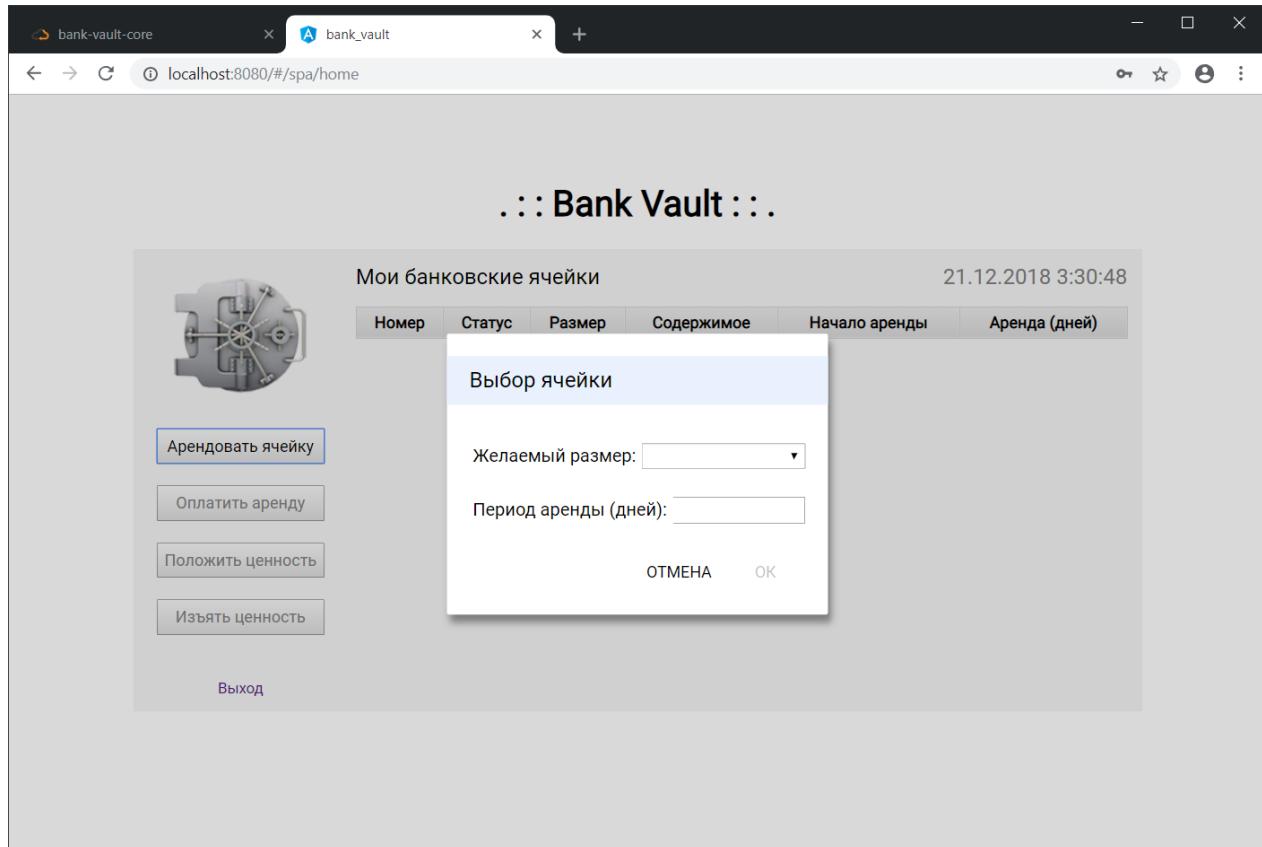


Рис. 22. Форма запроса на ячейку: кнопка OK неактивна

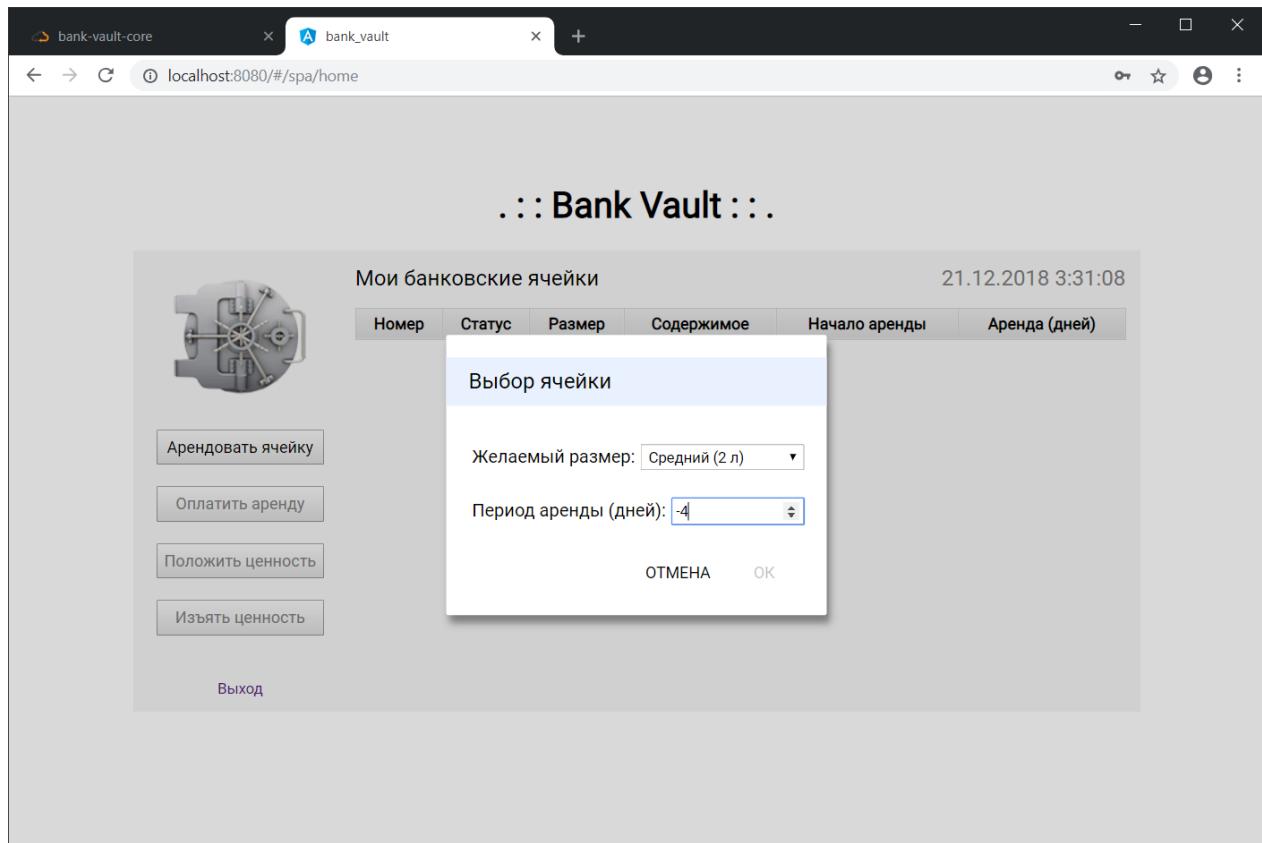


Рис. 23. Некорректно заполненная форма: кнопка OK неактивна

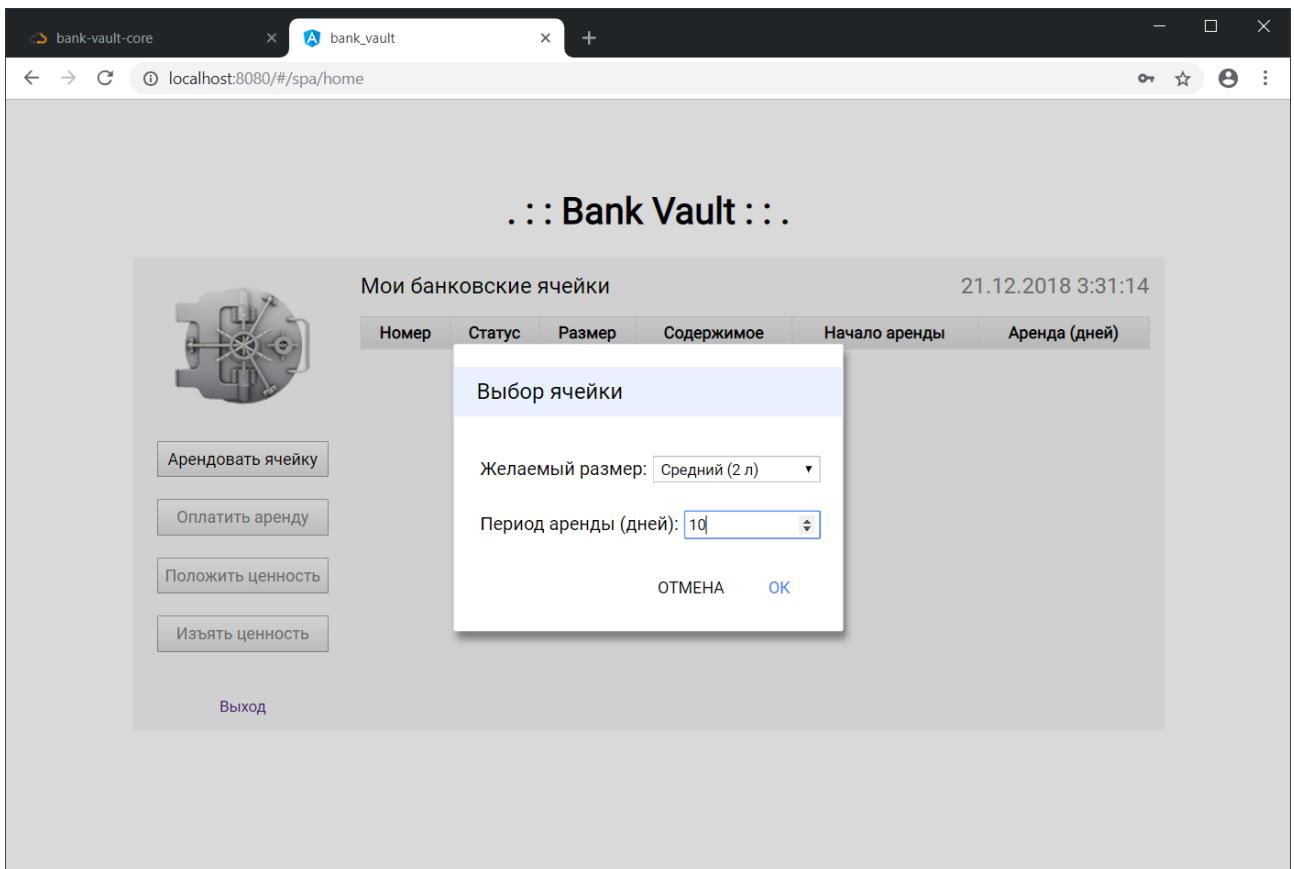


Рис. 24. Корректно заполненная форма заявки на аренду: кнопка OK активна

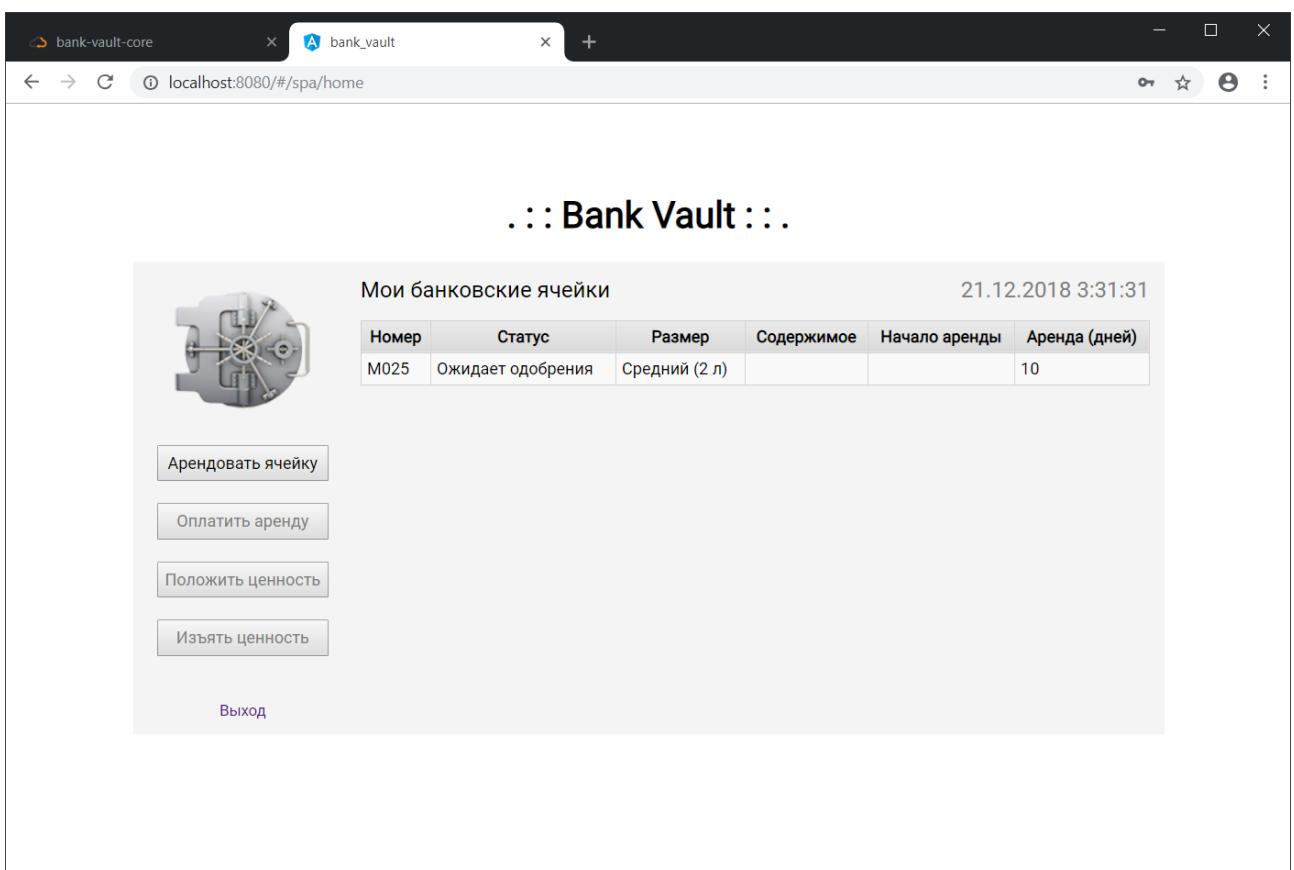


Рис. 25. Заявка на аренду ячейки успешно подана и ожидает одобрения

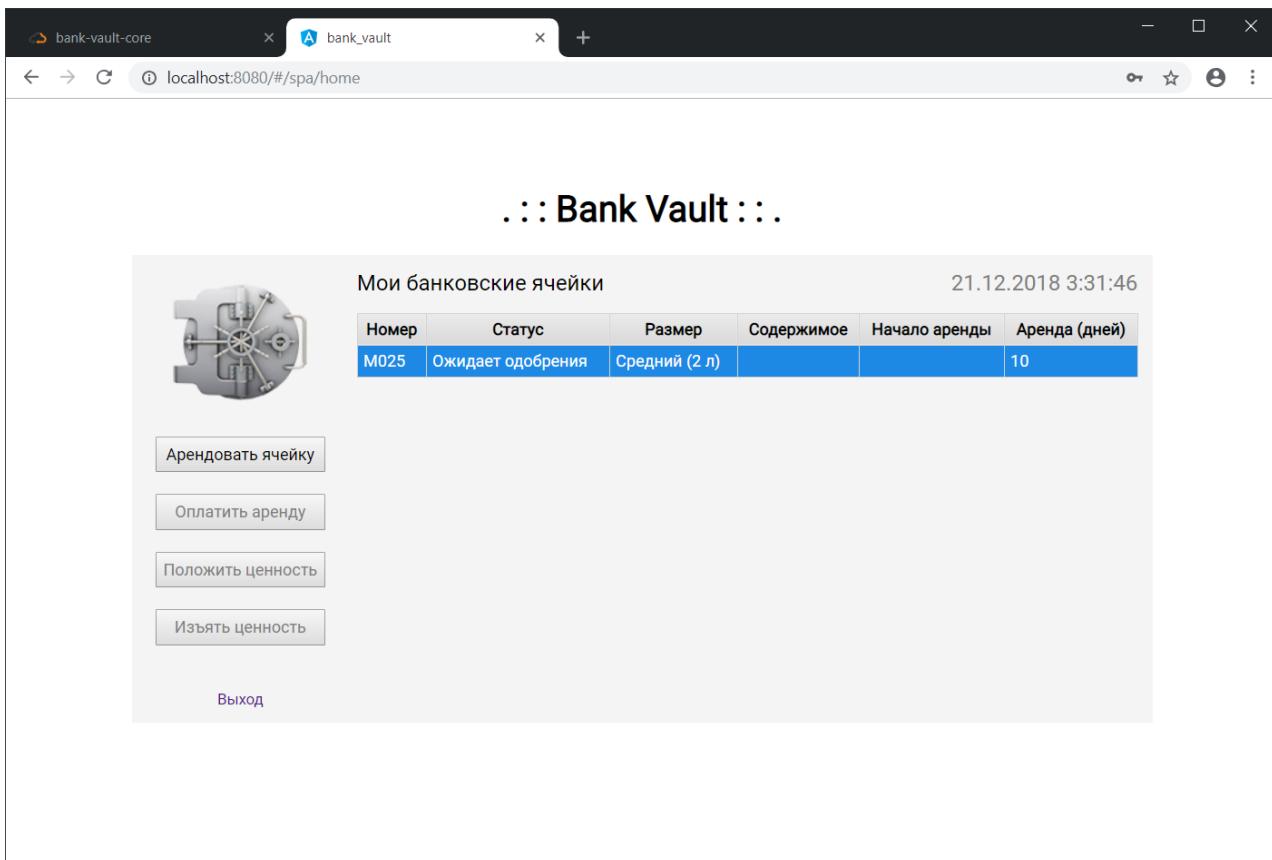


Рис. 26. Ячейка выбрана в таблице, но кнопки в левой панели неактивны, так как заявка еще не одобрена.

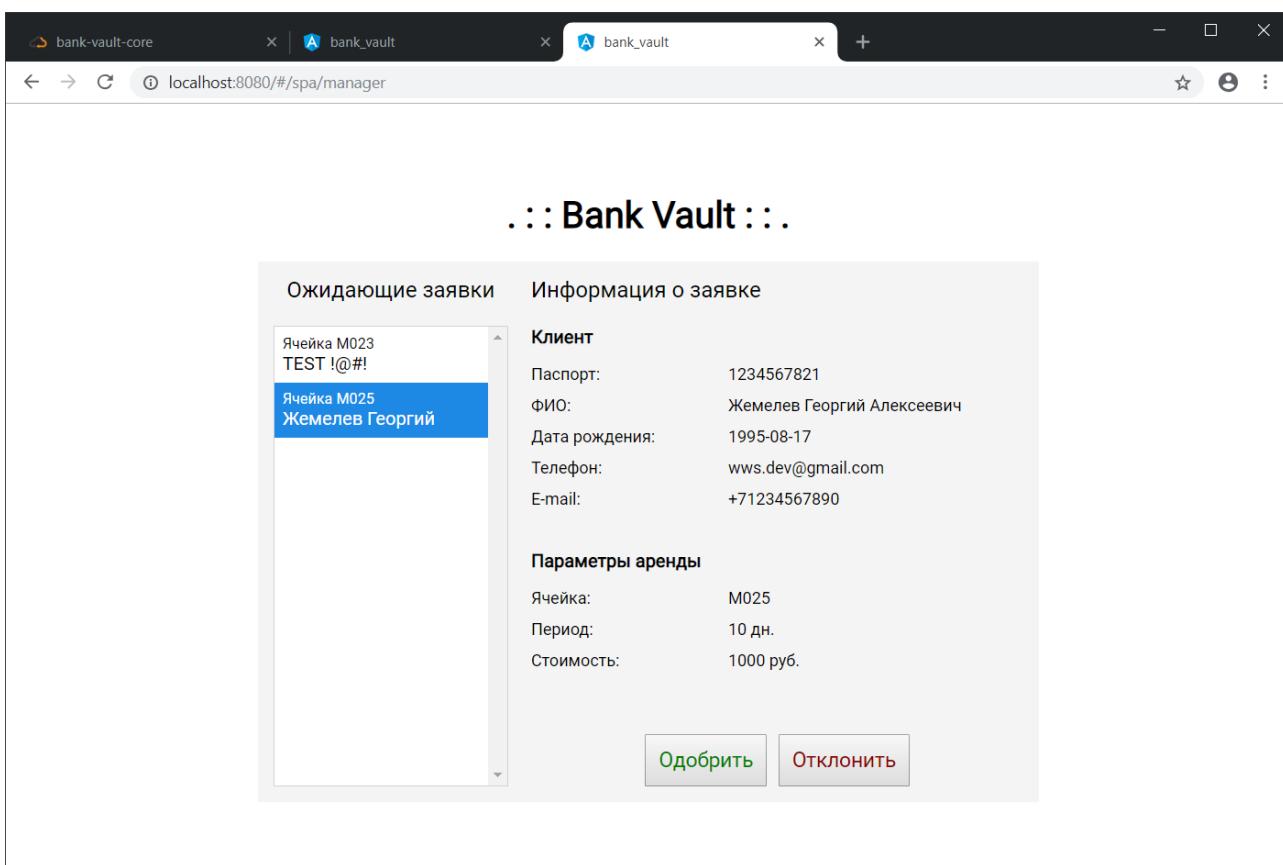


Рис. 27. Веб-интерфейс менеджера: выбрана только что поданная заявка

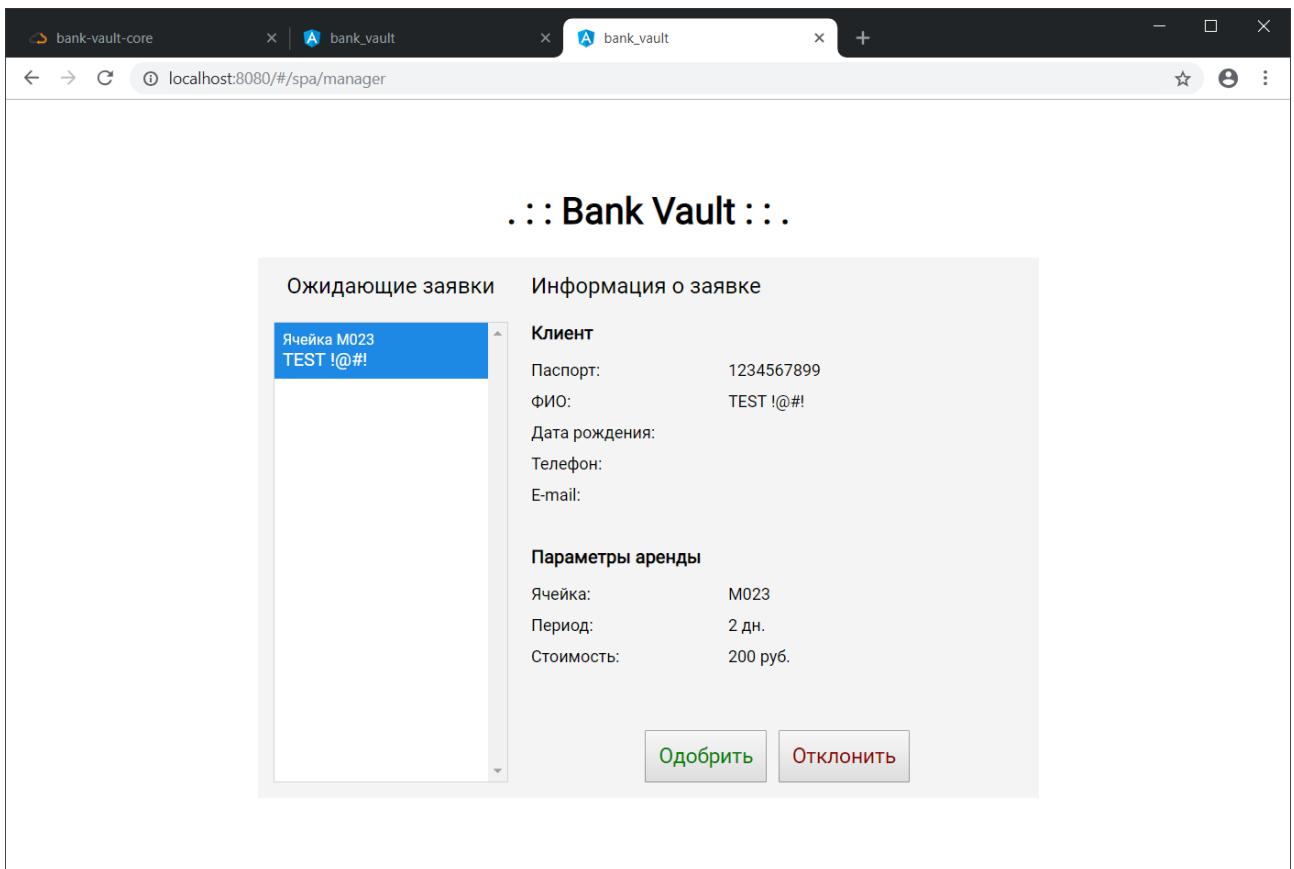


Рис. 28. После одобрения заявки на ячейку M025, она исчезает из списка

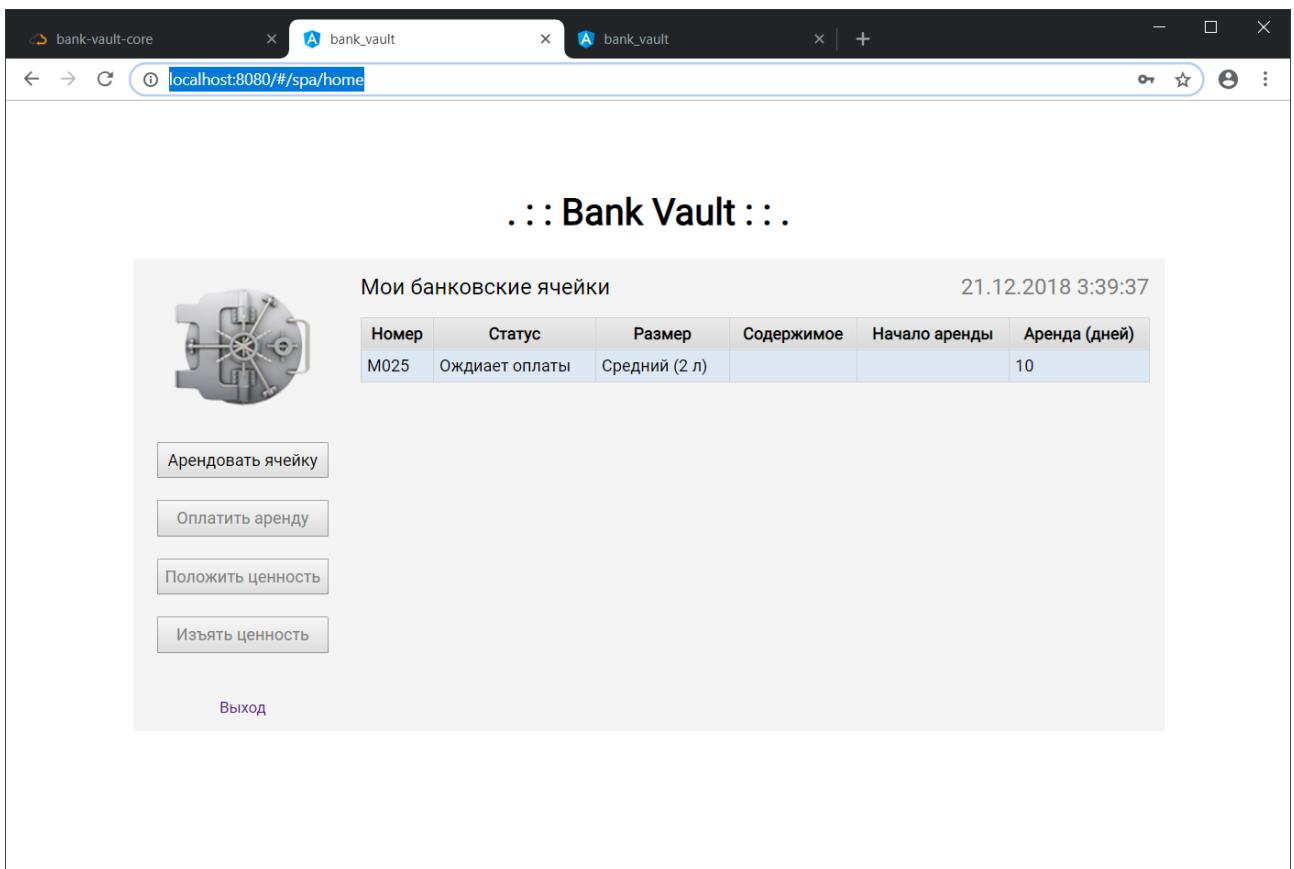


Рис. 29. Клиент видит, что заявка была одобрена и теперь ожидает оплаты

5.3. Оплата аренды

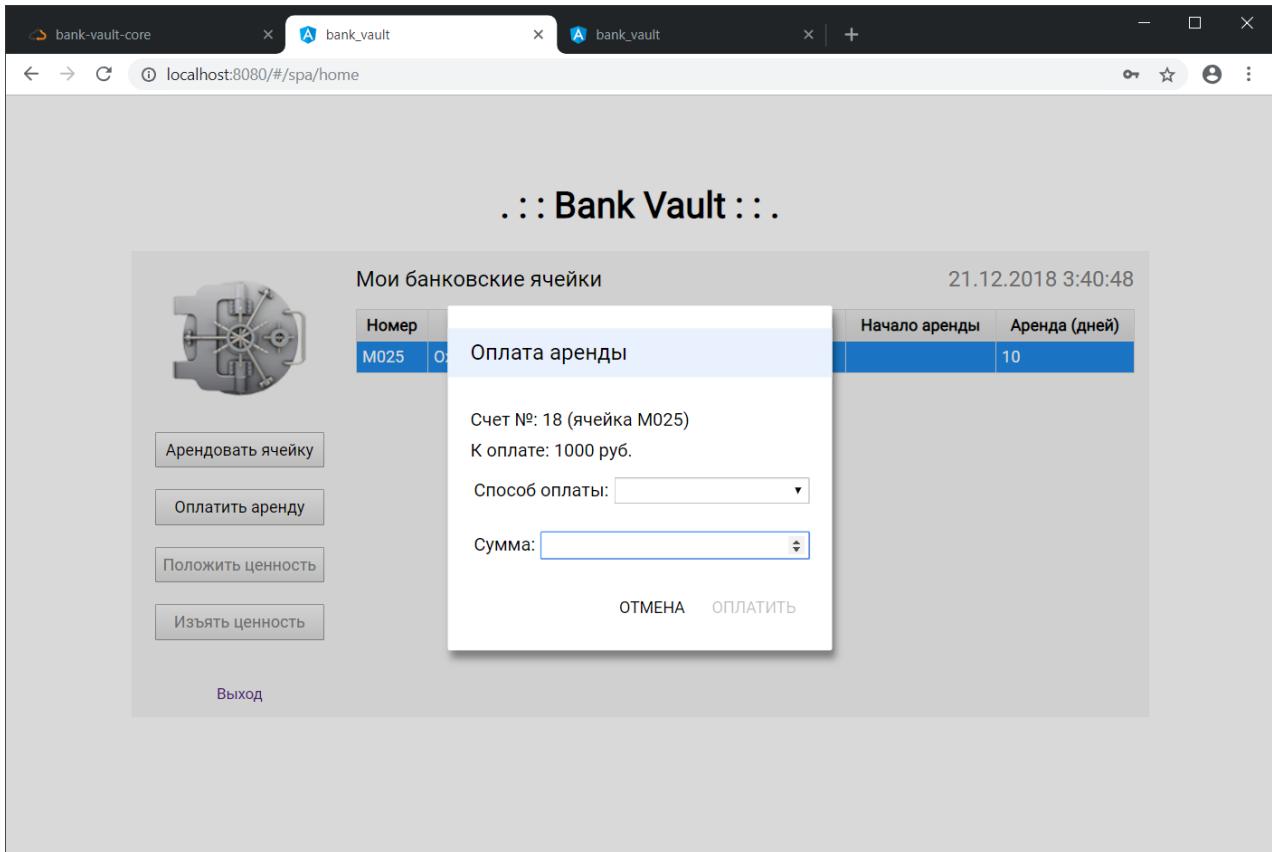


Рис. 30. Форма после выбора ячейки и нажатия на кнопку «Оплатить аренду»

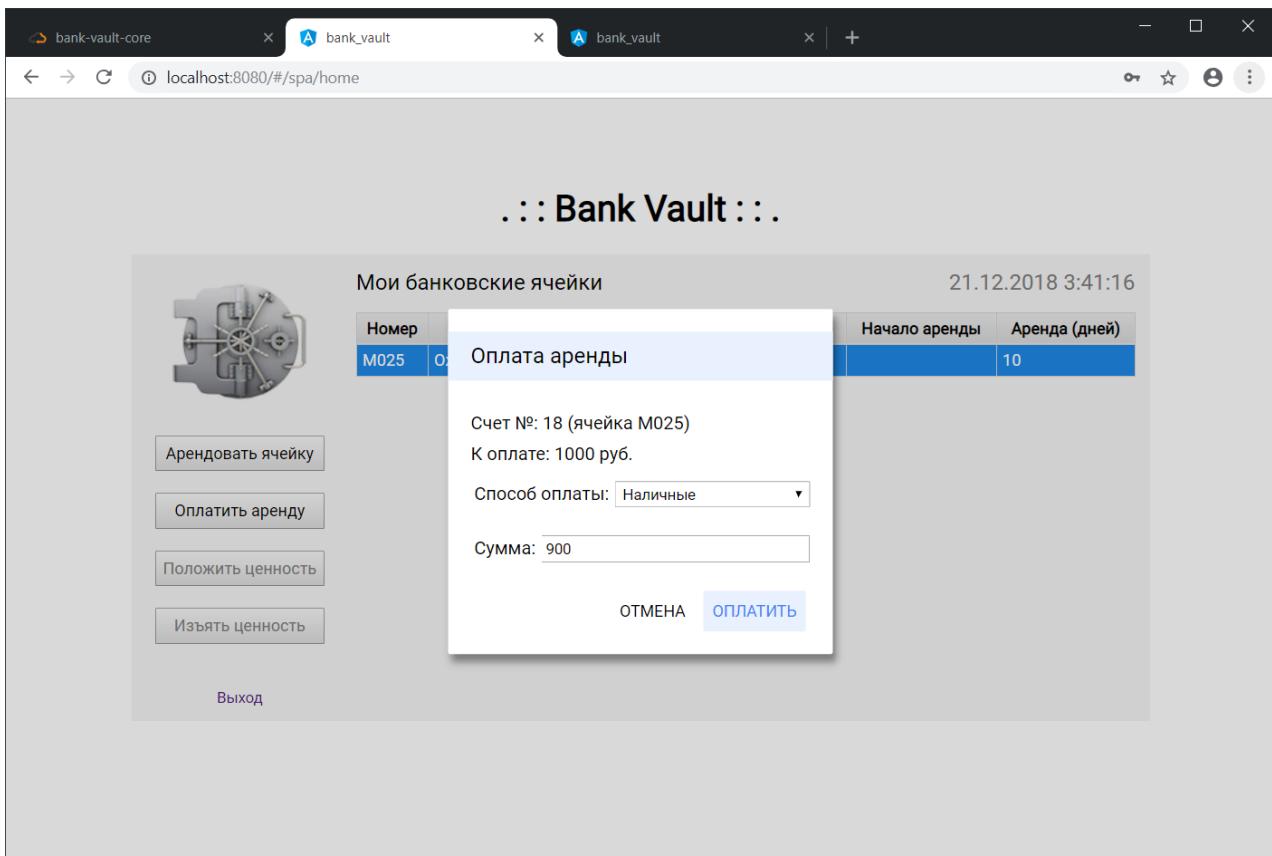


Рис. 31. Ввод заведомо недостаточной суммы для оплаты аренды

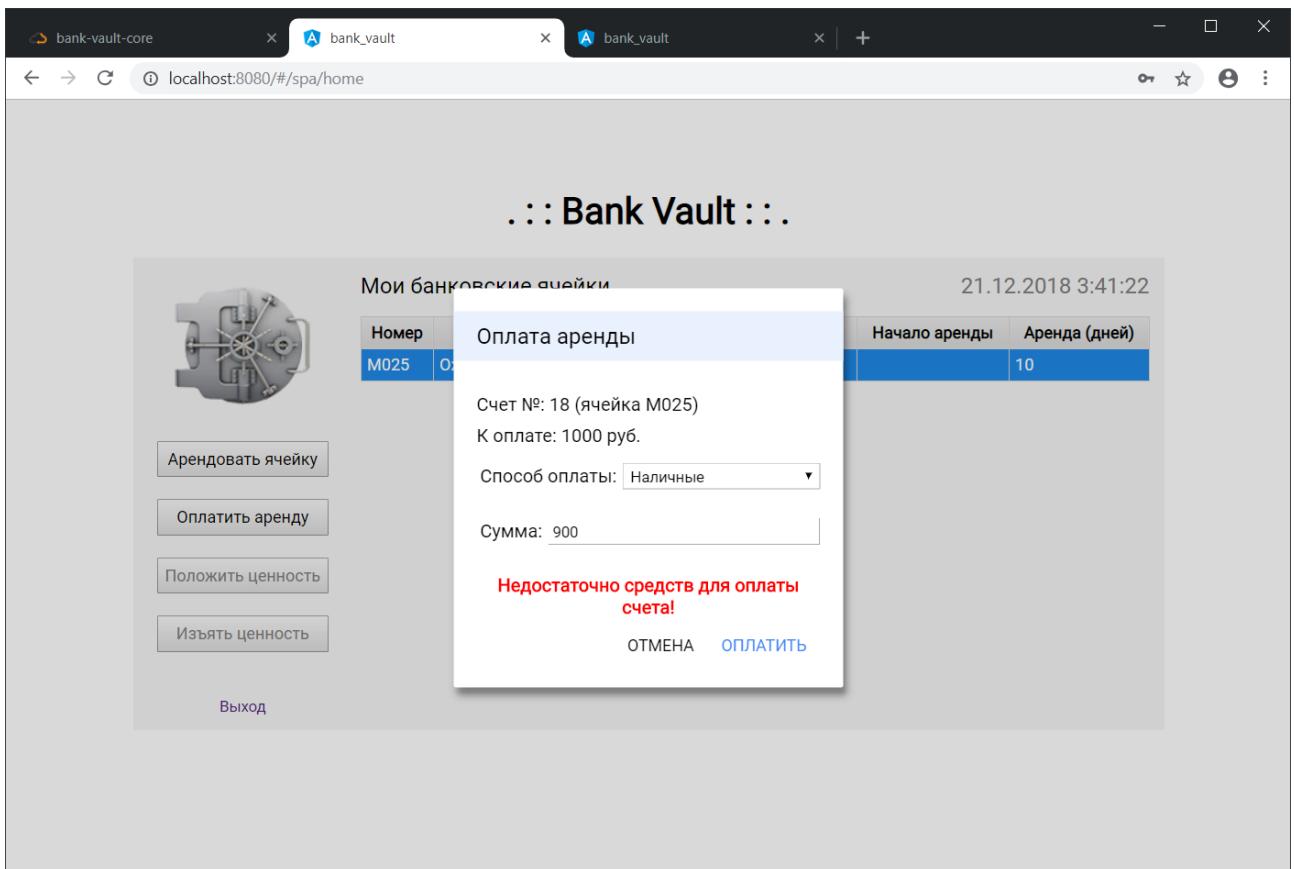


Рис. 32. Платежная система сообщает об ошибке

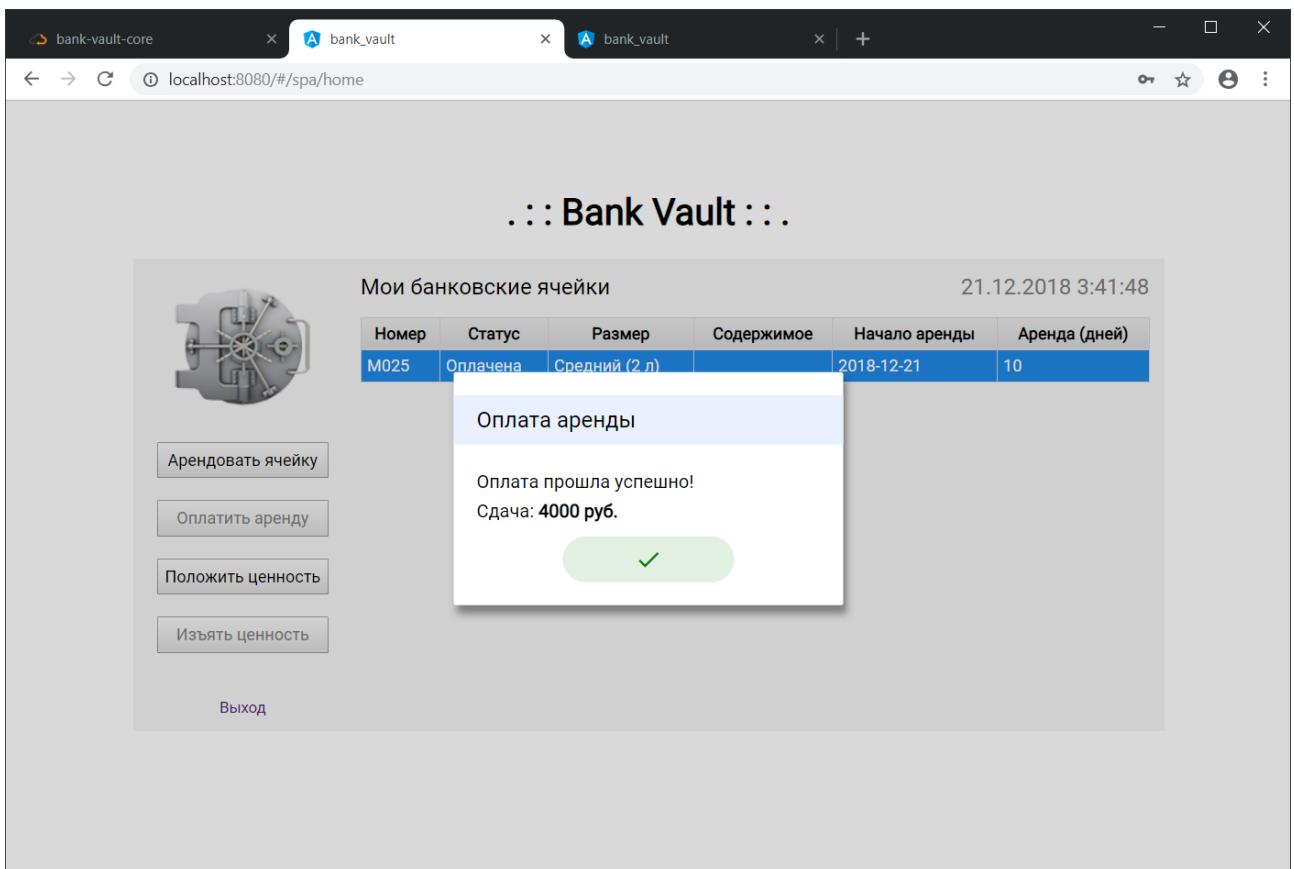


Рис. 33. После ввода 5000 руб. и нажатия «Оплатить» возвращается сдача

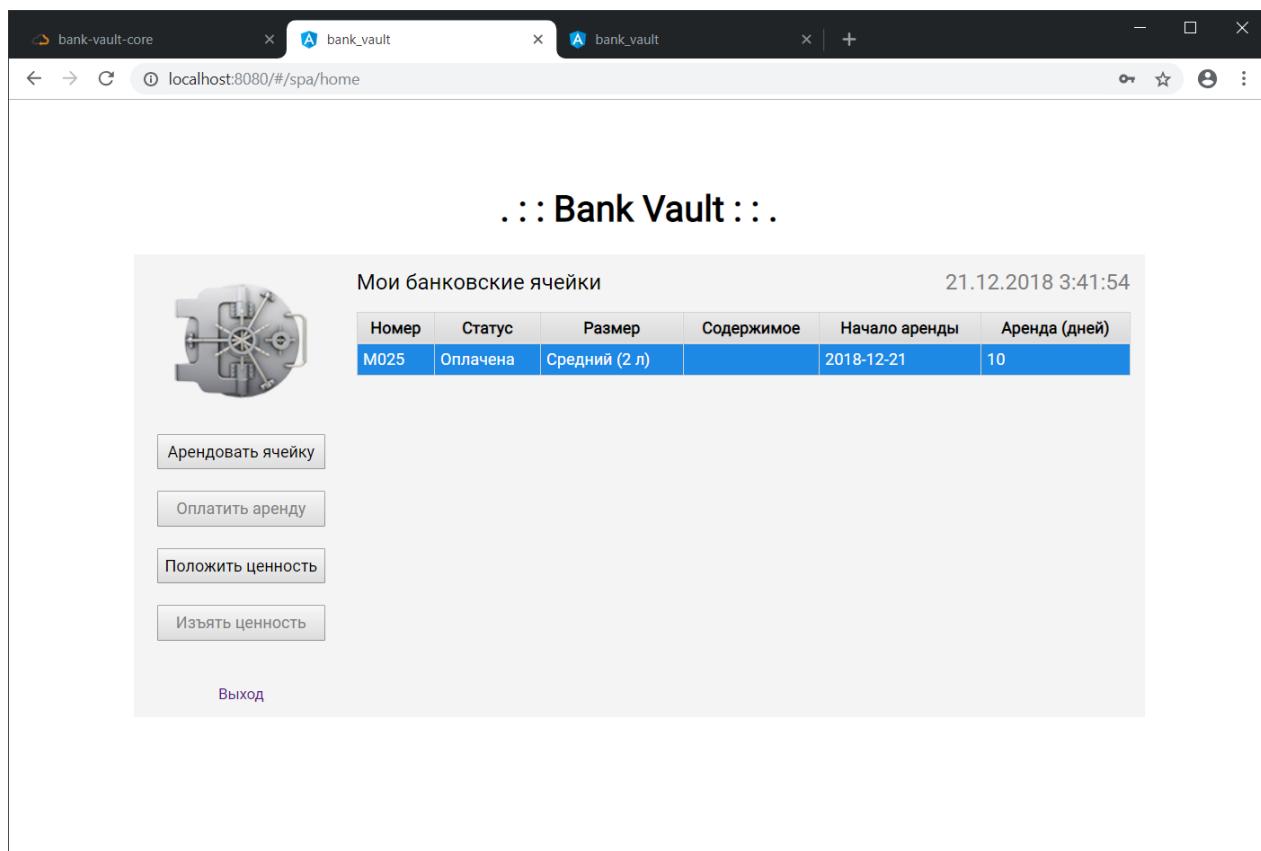


Рис. 34. После оплаты заявка меняет статус, кнопка «Положить ценность» активна

5.4. Манипуляции с ценностями

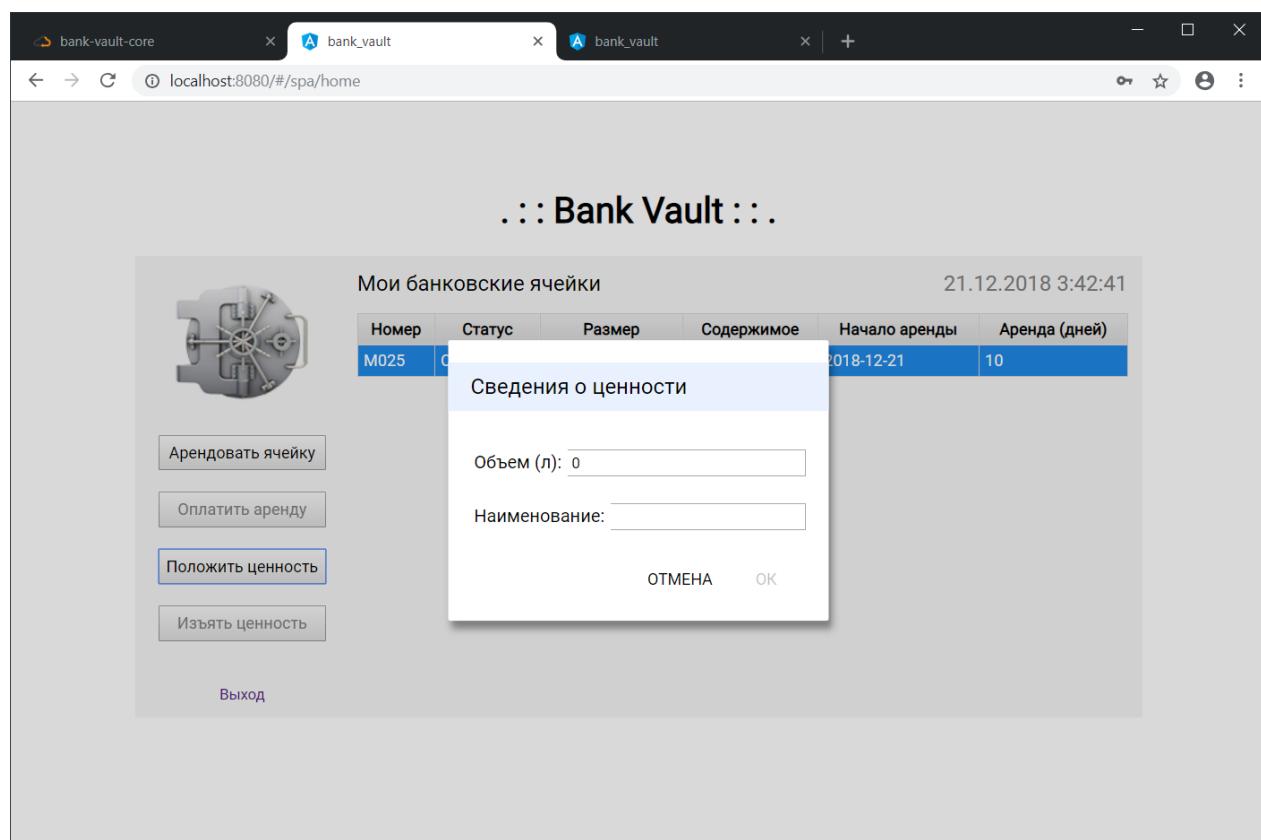


Рис. 35. Форма после нажатия кнопки «Положить ценность»

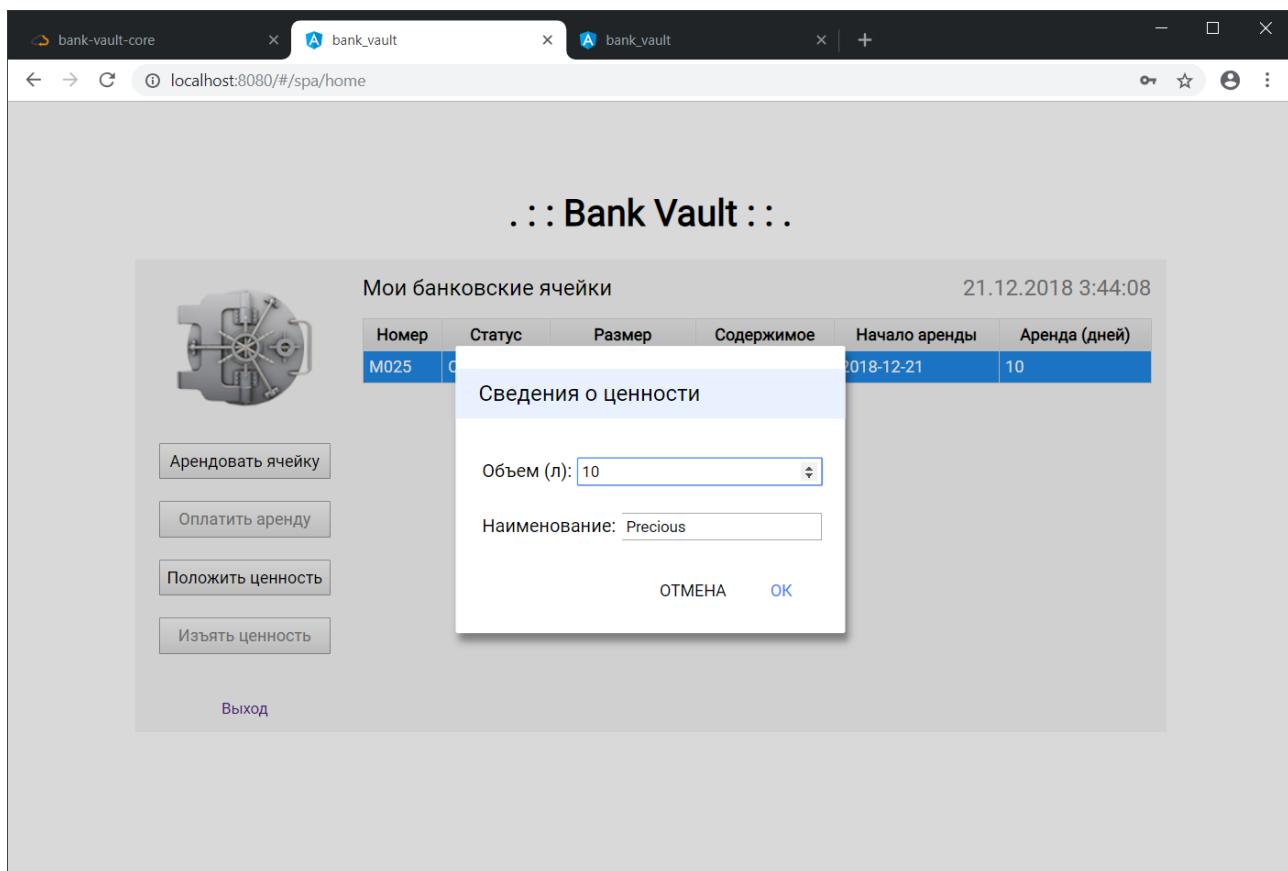


Рис. 36. Ввод сведений о ценности: объем превосходит объем ячейки

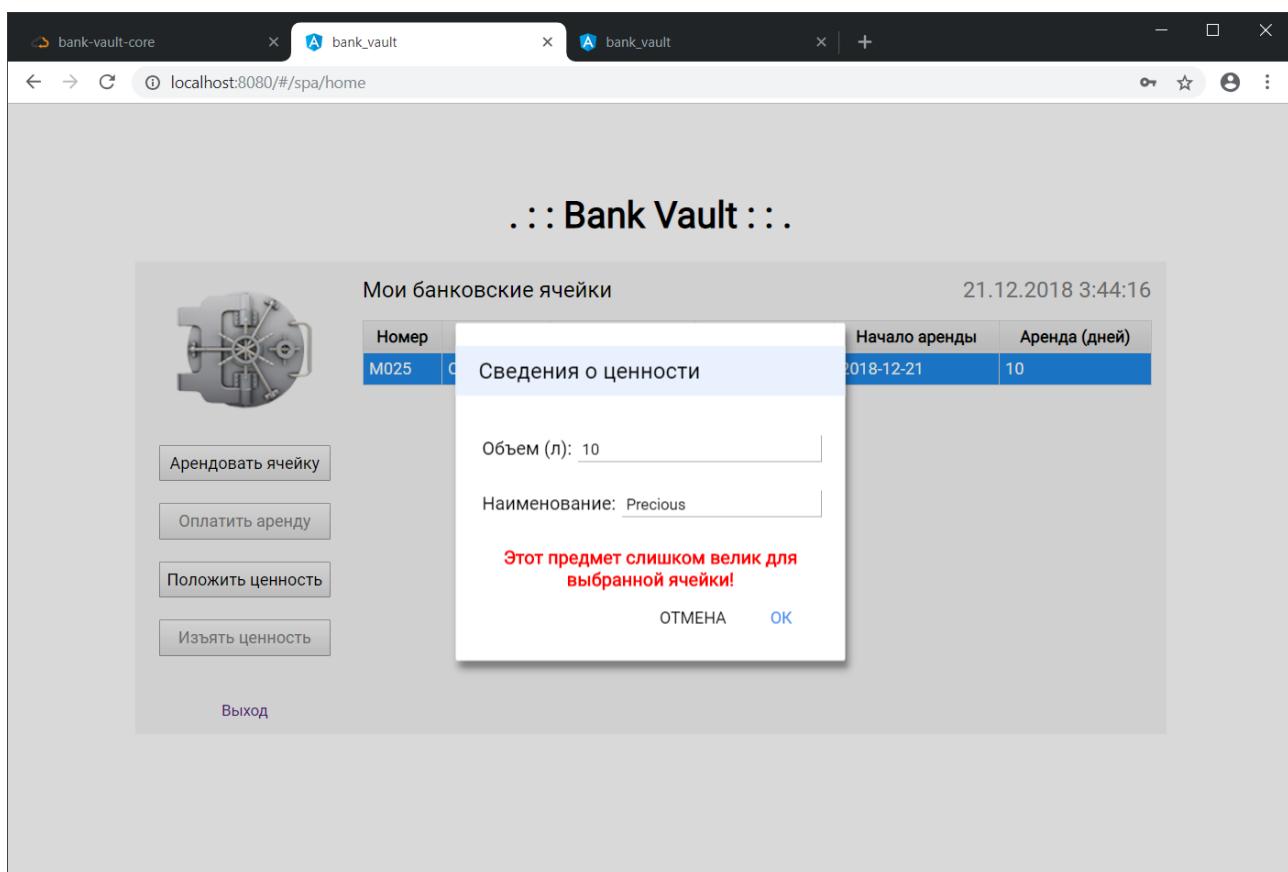


Рис. 37. Система обнаруживает несоответствие размеров и сообщает об этом

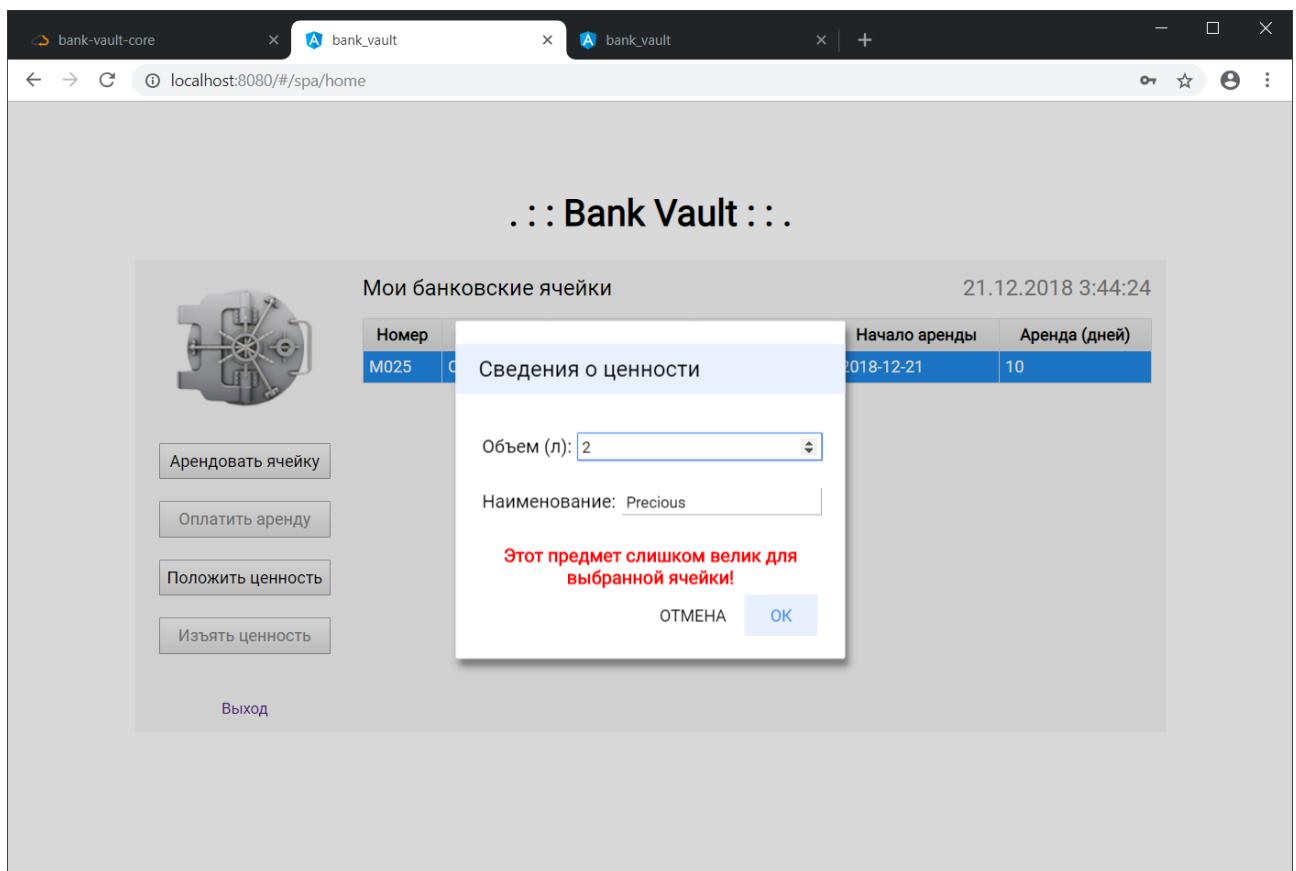


Рис. 38. Ввод корректного объема, занимаемого ценностью

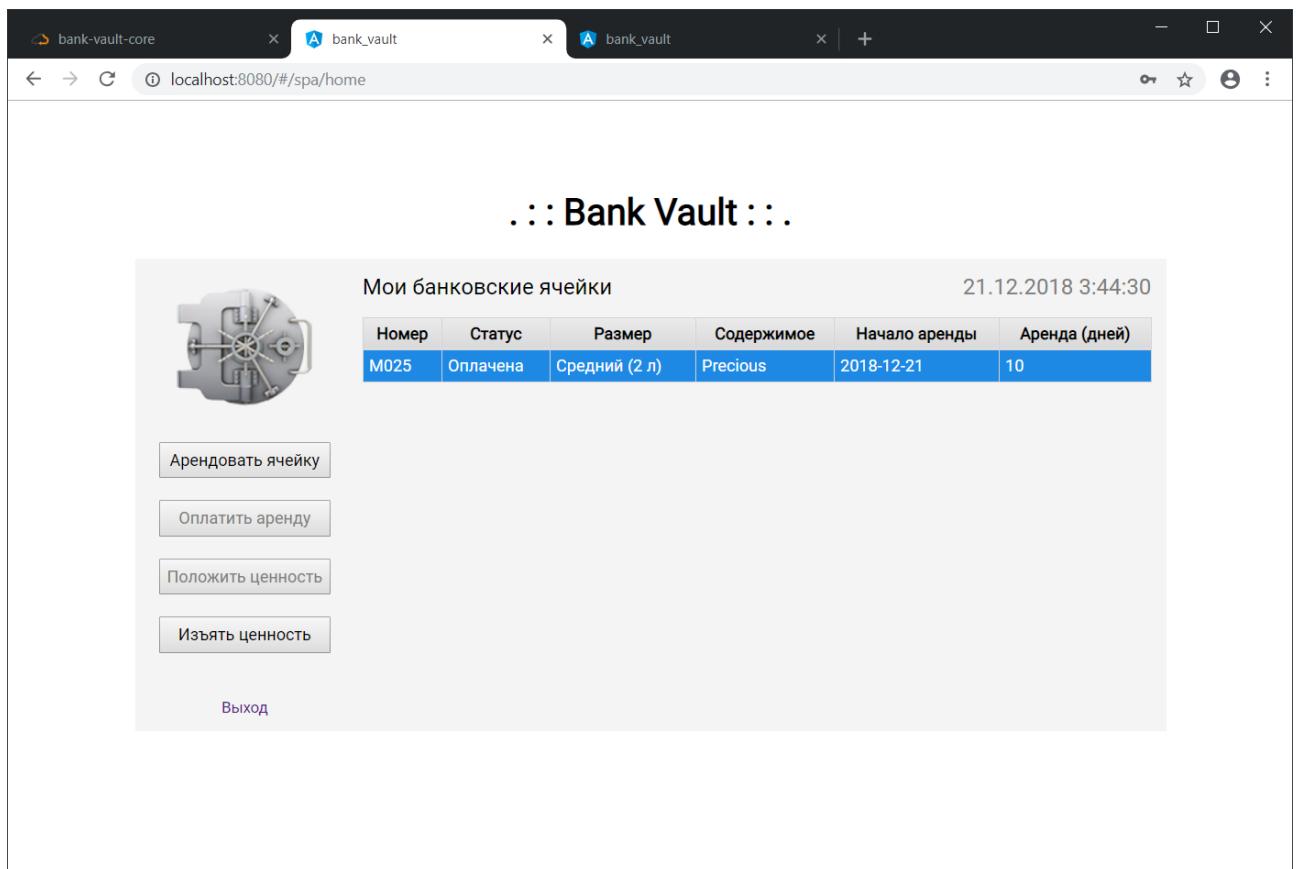


Рис. 39. Ценность помещена в ячейку, кнопка «Изъять ценность» активна

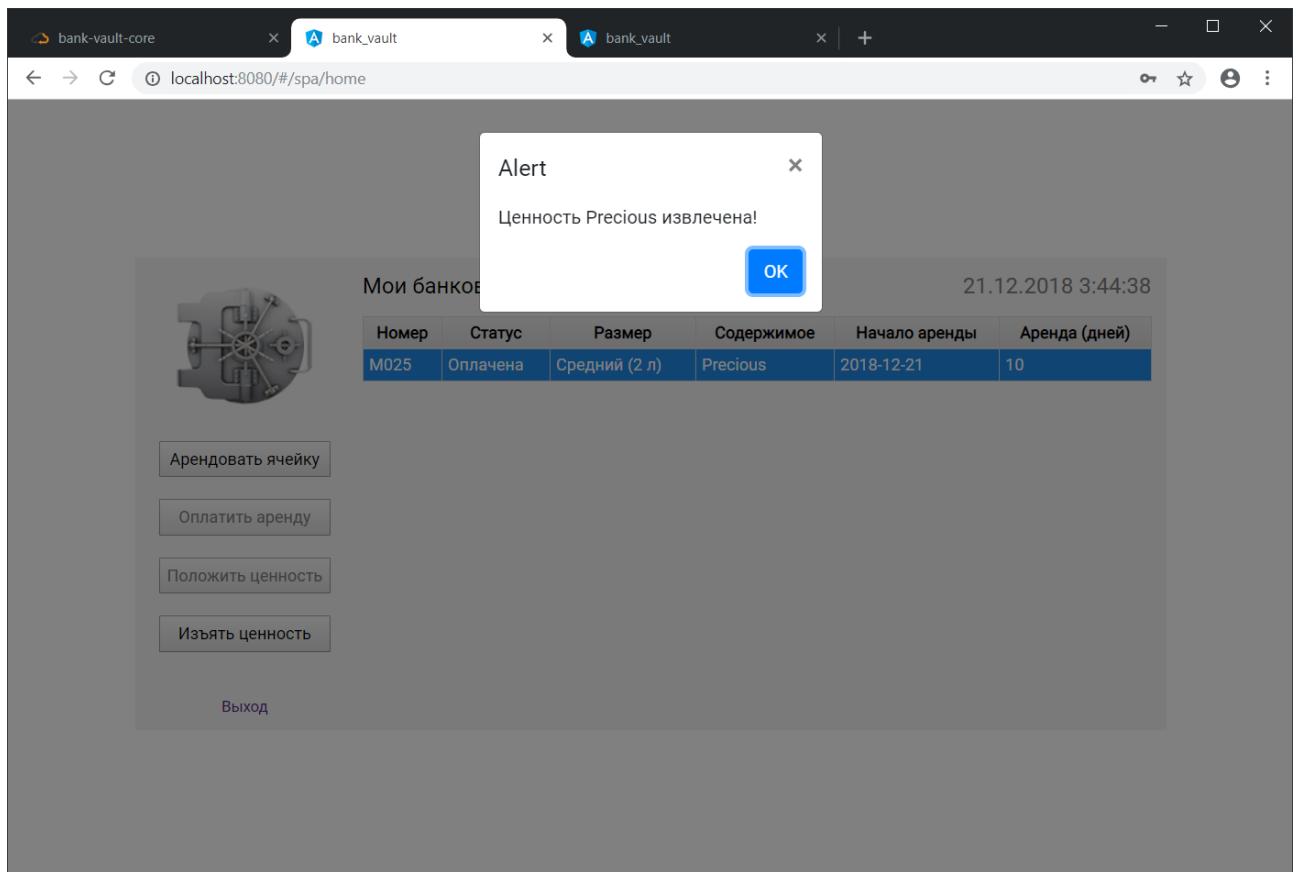


Рис. 40. Сообщение об успешно извлеченной ценности

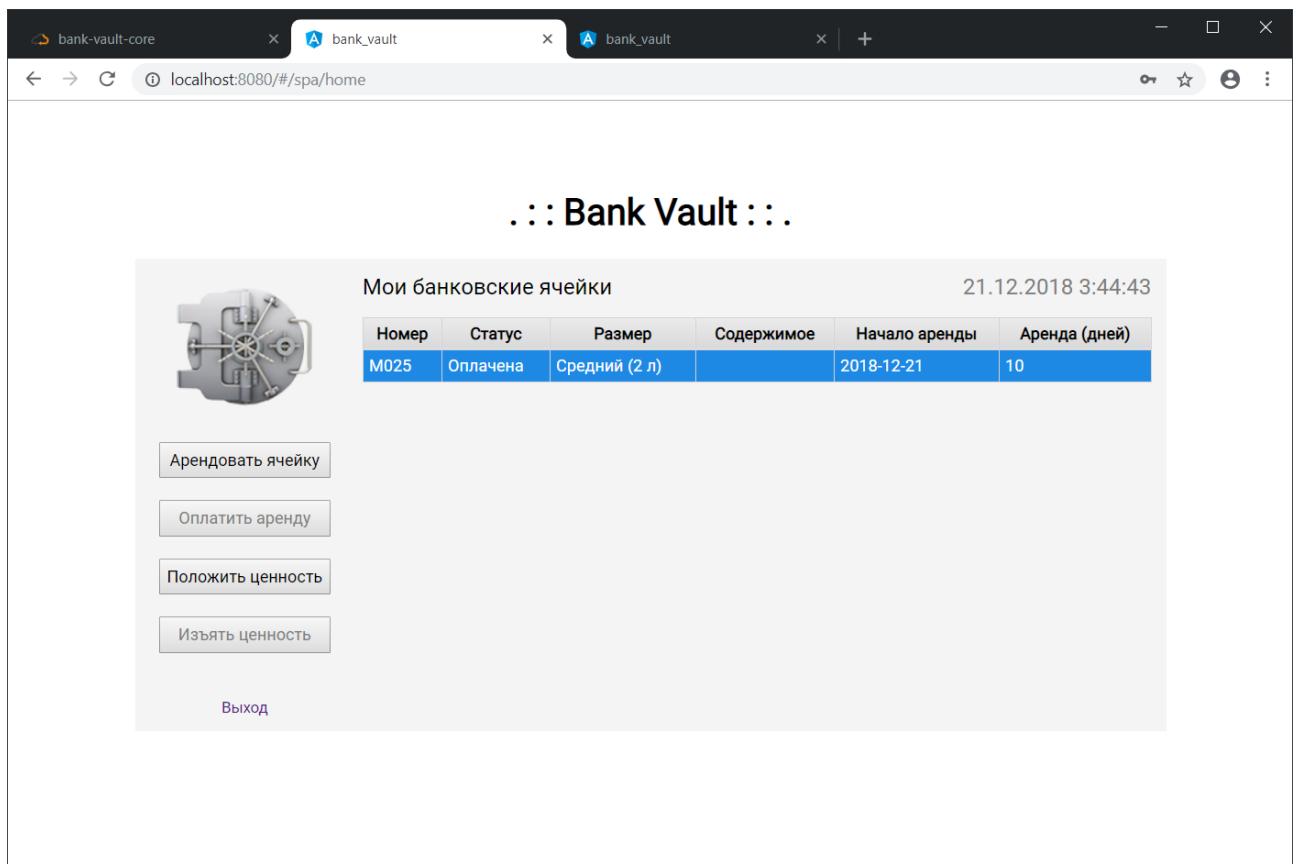


Рис. 41. После извлечения ценности ячейка пуста

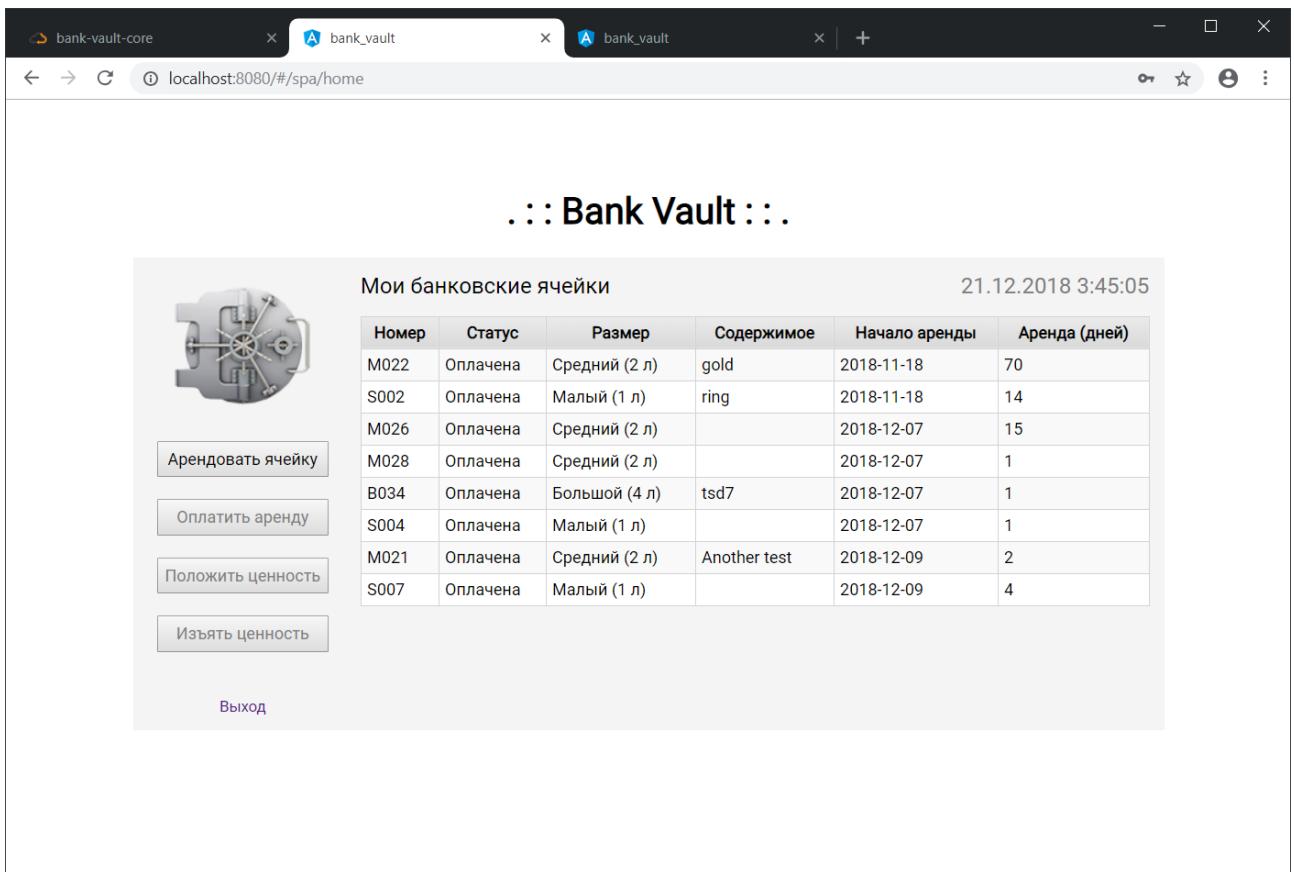


Рис. 42. Вид таблицы с банковскими ячейками другого пользователя

6. Инструкция системного администратора

Наличие в параметрах используемой системы сборки Gradle задачи bootJar позволяет собрать единый jar-архив, содержащий как исходные коды разработанной программы, так и все её внешние зависимости. Клиентская часть приложения собирается утилитой webdev и может быть упакована в zip-архив.

В предположении, что системный администратор имеет собранный jar-архив серверного приложения, доступ к файлу со схемой БД и архив с файлами клиентского приложения, для развертывания необходимо:

- 1) установить СУБД PostgreSQL версии 9.5 или выше;
- 2) создать в СУБД пользователя «jdbc» с паролем «JavaJ» (без кавычек);
- 3) создать в СУБД базу данных (БД) bank_vault и выдать пользователю jdbc полный доступ к объектам в созданной базе;
- 4) инициализировать БД схемой из файла db_init.sql, который расположен в репозитории проекта в папке bank-vault-core/src/main/resources;

- 5) распаковать архив с файлами клиентского приложения в папку /public/www (для ОС семейства Linux) или C:\public\www (для ОС семейства Windows);
- 6) запустить серверное приложение командой java –jar <имя_архива>.

В результате приложение будет запущено и доступно на TCP-порту 8080.

7. Инструкция пользователя

Чтобы воспользоваться приложением, пользователю необходимо открыть веб-браузер и ввести в его адресную строку URL, на котором размещен веб-сервер приложения. После этого пользователь может свободно работать с приложением, в том числе закрывать вкладку или окно браузера после входа – факт его аутентификации будет сохранен до нажатия кнопки «Выход».

8. Выводы

В результате выполнения поставленного задания было разработано полноценное веб-приложение, выполняющее все заявленные требования.

Возможные пути улучшения разработанного приложения включают в себя введение механизма нотификаций пользователей. В данный момент обновление статуса производится за счет периодического опроса сервера, что неэффективно с точки зрения нагрузки на серверную часть приложения и недостаточно наглядно для пользователя. Подобное улучшение может быть выполнено путем применения технологий Server Sent Events или Web Socket.

С точки зрения основных свойств распределенных систем созданное приложение можно оценить следующим образом:

+ прозрачность, т.к. веб-приложение базируется на URL и не привязано к конкретному местоположению, обеспечивает транзакционность при совместном доступе;

+ открытость, т.к. серверная часть предоставляет REST API и документацию к нему (в виде страницы Swagger 2), а DTO сериализует в JSON;

+/- масштабируемость, т.к. приложение успешно работает со множеством

пользователей и может быть защищено административно, но не расширяемо географически, потому что не поддерживает кластерный режим работы.

Приложение не испытывалось при больших нагрузках, поэтому нельзя сделать выводы о производительности.

Разработанный веб-интерфейс с точки зрения пользователя можно охарактеризовать как достаточно удобный, интуитивно понятный и эстетичный.