### 1 Планирование системы

**Требования заказчика:**

* Пользователь должен иметь возможность увидеть свой баланс
* Пользователь должен иметь возможность снять деньги со своего банковского счета
* Пользователь должен иметь возможность пополнить банковский счет
* Пользователь должен иметь возможность посмотреть список совершенных операций
* Система должна предоставлять админку для банкомата ([HTTP-Basic Auth](https://api.rubyonrails.org/classes/ActionController/HttpAuthentication/Basic.html))
* В админке можно указать количество наличных денег в банкомате (имитируя поведение инкасатора)
* Изменение наличия в банкомате наличности должно влиять на проведение операций
  + Снятие наличности - уменьшает количество доступной наличности в банкомате
  + Пополнение счета - увеличивает количество наличности в банкомате

анализ:

**банкомат:**

от банкомата требуется прием/выдача наличных при проведении через него операций над счетом клиента банка. так как банкомату самому по себе ничего про клиентов и их счета не должно быть известно, то он будет обладать только методами для изменения хранимой в нем наличности, а сами операции проводимые над банкоматом следует вынести в опциональный уровень.

так как само приложение + банкоматы представляют собой клиент-серверную систему, то это позволит одному серверу поддерживать множество банкоматов

**пользователи:**

Во всех пунктах упоминается пользователь, но не упоминается он должен быть один на все приложение, или их должно быть множество. Однако учитывая специфику приложения - делать его однопользовательским нет никакого смысла.

сущность пользователей необходима для того, чтоб система могла отличить одного клиента от другого, и определить какие ресурсы доступны каждому из них.

**счета и деньги:**

Во всех операциях присутствуют операции с деньгами, но не указано явно в каком месте и формате они должны храниться.

В некоторых пунктах упоминается термин “счет”, предположим что он и будет местом хранения денег, тогда встает вопрос связи счета с деньгами и с пользователем:

* так как нам от денег нужно только их количество, то сделаем их как численное значение и поместим его в счет связью ОТО
* возможность завести несколько счетов под разные нужды звучит как рациональный функционал банковской системы, следовательно связь пользователей со счетами должны быть ОТМ

В последнем пункте задания упоминается что деньги должны быть не только у счетов пользователя, но и у банкомата, привязку денег к банкомату будем производить аналогично привязке денег к счетам.

**администратор/инкассатор:**

по факту администратор/инкассатор является тоже пользователем, но с другой функцией. и так как у нас есть более чем один тип юзеров(термин введен чтоб не путать с “пользователями), то для разграничения функционала создадим сущность “юзер” которая будет уровнем абстракции, и от неё будет по наследнику для каждого типа пользовательских аккаунтов, для каждого из которых будет своя таблица в БД. Доступ к попыткам вызова функционала будем ограничивать с помощью авторизации.

**операции:**

в приложении операции будут вызываться через опциональный уровень, где они будут проводиться по алгоритму:

1. из базы достаются сущности счета и банкомата с проверками на существование и доступ.
2. у банкомата и счета вызываются соответствующие операции методы. с проверкой на достаточность наличных, для их проведения.
3. если хоть одна из проверок в п1 и п2 не прошла, то операция отменяется. если все прошло без проблем, то состояния банкомата и счета перезаписываются в БД

полученная стартовая схема предметной области:

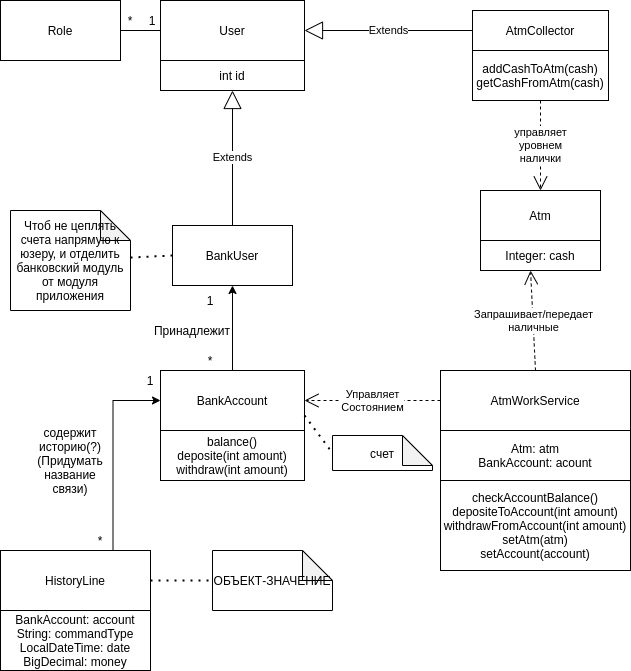


Рисунок 1 - Стартовая схема предметной области

### 2. Анализ системы

**Видение проекта:**

продукт представляет собой систему взаимодействия с банкоматами, подключенным к централизованной системой управления как тонкий клиент к серверу. каждому банкомату присваивается id, при установке обеспечения на банкомат этот id вбивается в файл конфигурации. для проведения каких-либо действий банкомат обращается по HTTP протоколу к Rest API серверу централизованного управления для обменом данных, визуальный интерфейс банкомата построен на основе HTML+CSS+JS.

**Функциональность продукта:**

продукт позволяет пользователю посредством визуального интерфейса идентифицироваться в банковской системе и получить доступ к своему счету для проведения операций над ним, а также для получения возможности узнать состояния своих счетов или истории операций над ними

**Классы и характеристики пользователей:**

Пользователи приложения деляться на два класса - непосредственно пользователи банковской системы и обслуживающий персонал. Пользователи банковской системы имеют возможность производить операции над своими счетами, пользователь техобслуживания следят за уровнем наличности в банкоматах.

**Среда функционирования продукта:**

Продукт планируется писать на Java для серверной части и HTML+CSS+JS для тонкого клиента, база данных на PostgreSQL, все технологии являются кроссплатформенными, так что ОС как сервера так и банкомата не имеет значения. Главным условиям функционирования является наличие сетевого/интернет соединения между банкоматом и сервером.

**Документация для пользователей:**

Так как данная система очень похожа на те, которые уже имеют крупный обиход в сфере обслуживания банкоматов, да и в целом имеет интуитивный интерфейс, то наличие документации для пользователей является лишним элементом продукта

### 3. Дизайн системы

**Видение дизайна:**

Так как банкомат представляет собой устройство широкого пользования, то у него должен быть максимально простой интерфейс, чтоб им могли с легкостью пользоваться люди плохо знакомые с компьютерной техникой. Потому было принято решения за основу дизайна взять уже существующие банкоматы.

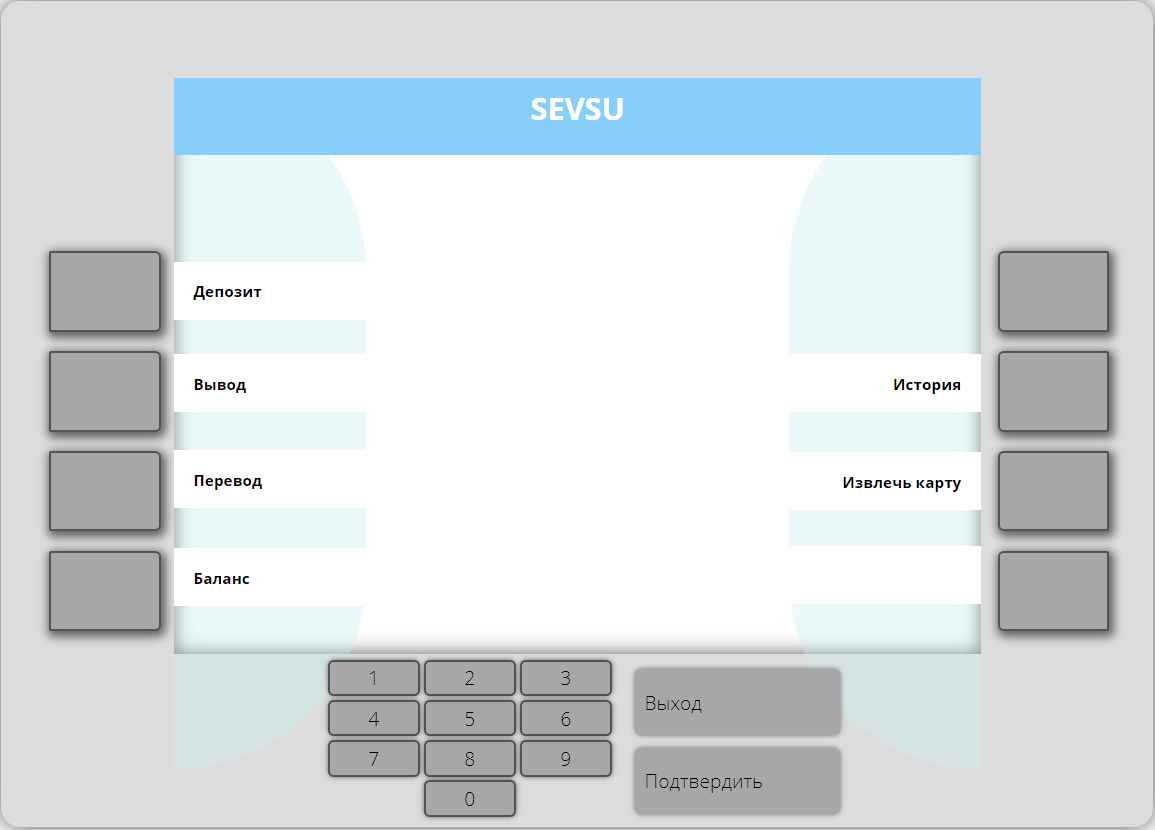


Рисунок 2 - Главная страница банкомата.

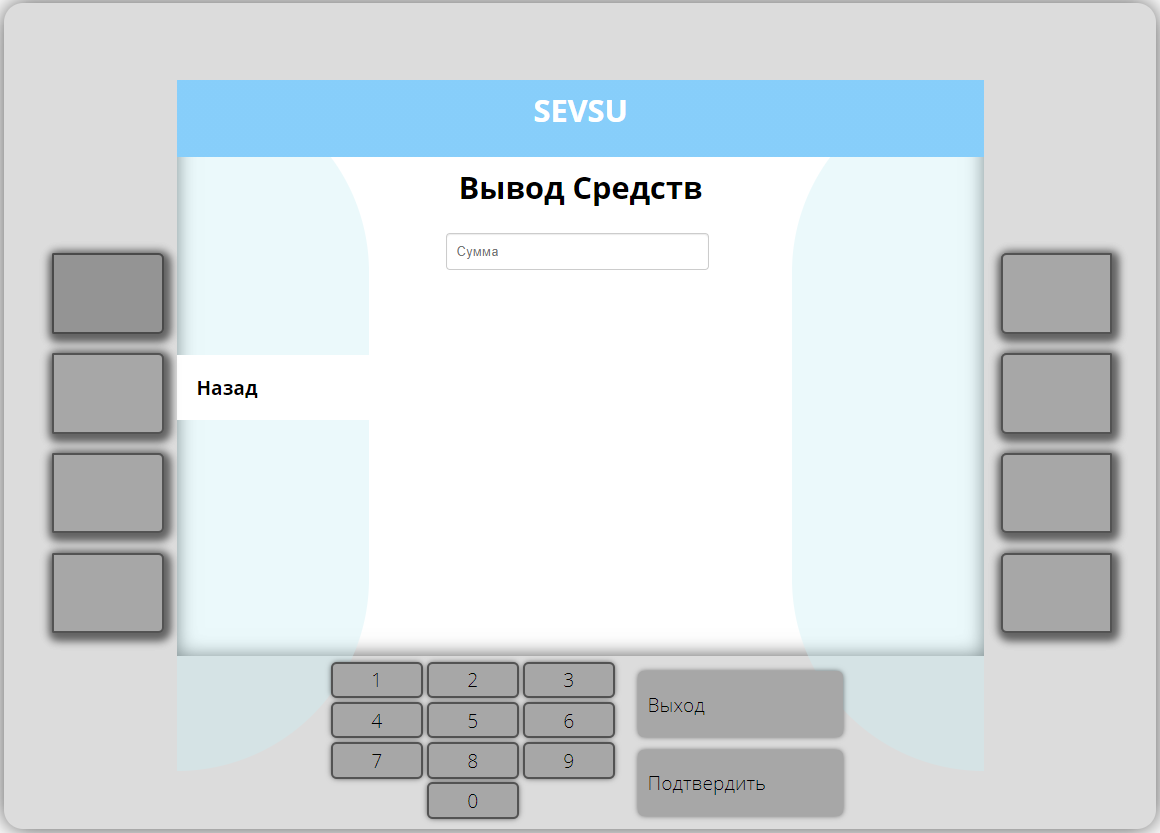


Рисунок 3 - Пример функциональной страницы банкомата.

Как можно видеть из приведенных выше макетов, банкомат представляет собой устройства с кнопками, на главной странице которого находится меню выбора проводимой операции, а при выборе операции появляется окно в котором надо указать сумму для проводимой операции.

**Дизайн аутентификации:**

Было принято решение аутентификацию проводить таким же образом, как она производится в существующих системах. В банковском отделении при регистрации счета будет выдаваться ключ-карта, которая будет представлять собой физический носитель логина пользователя. При вводе карты в банкомат будет запрашиваться ПИН-код, который, по факту, будет являться паролем.

### 4. Создание и разработка продукта.

**Подготовка к разработке:**

Для начало был определен фронт работ и создан бэклог на JIRA, подготовлен репозиторий на Gitlab, настроен CI на проверку сборки серверной части приложения и прохождение ею тестов. Далее был создан проект на Java с использованием Gradle для подключения фреймворков и библиотек, таких как Mockito, Spring, Swagger и так далее.

**Непосредственно разработка:**

Разработка представляла собой последовательность действий:

1. Участниками проекта в ходе совещаний определяется дальнейшее развитие системы.
2. Менеджер заводит в столбце “TO DO” в JIRA задачи.
3. Разработчик берет задачу из столбца JIRA “TO DO” на выполнение, помечает себя в ней как исполнителя, и переносит её в столбец “IN PROGRESS”.
4. Разработчик выполняет взятую задачу и переносит её в столбец “SELF TESTING”.
5. Теперь разработчик должен проверить эту задачу на выполнение инвариантов и очевидные баги, после чего он переносит задачу в столбец “CODE REVIEW”.
6. Когда задача находится в столбце “CODE REVIEW”, её осматривают другие участники проекта на наличие ошибок в бизнес-логике или плохо оптимизированного кода.
7. После проверки задача переноситься в столбец “DONE” чем помечается как выполненная.

**Документация:**

По мере разработки приложения с помощью Swagger создавалась документация по использованию контроллеров API сервера.

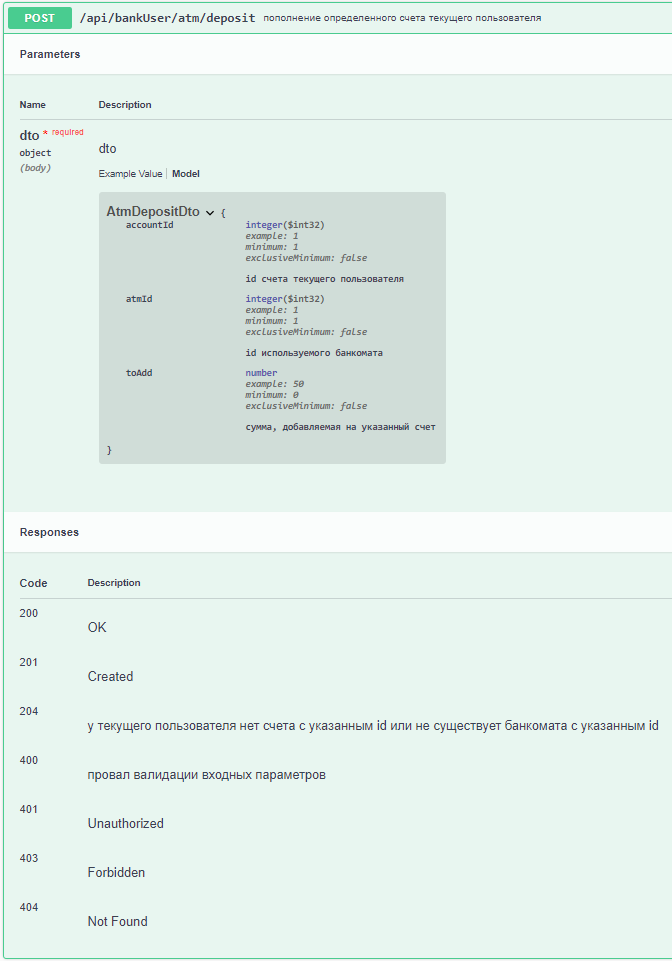


Рисунок 4 - Пример документации метода контроллера.

Как можно видеть, фронт-разработчик, или разработчик мобильного/десктоптного UI-приложения имея такую документацию будет знать, к какому функционалу у него есть доступ, какие требуются параметры, и какие инварианты у этого функционала.

### 5. Тестирование продукта.

**Автотесты:**

Автотесты написаны с помощью библиотек JUnit, Mockito, PowerMock и SpringTest. Эти автотесты представляют собой тестирование методов на корректность выполнения. Для этого все объекты, не являющиеся POJO заменяются на заглушки (моки), а POJO создаются в контексте теста, после чего вызывается метод, ему передаются параметры, и проверяются возвращаемые значения и значения на которые метод должен был повлиять. Покрытие тестами проверялось с помощью утилиты “jacoco”

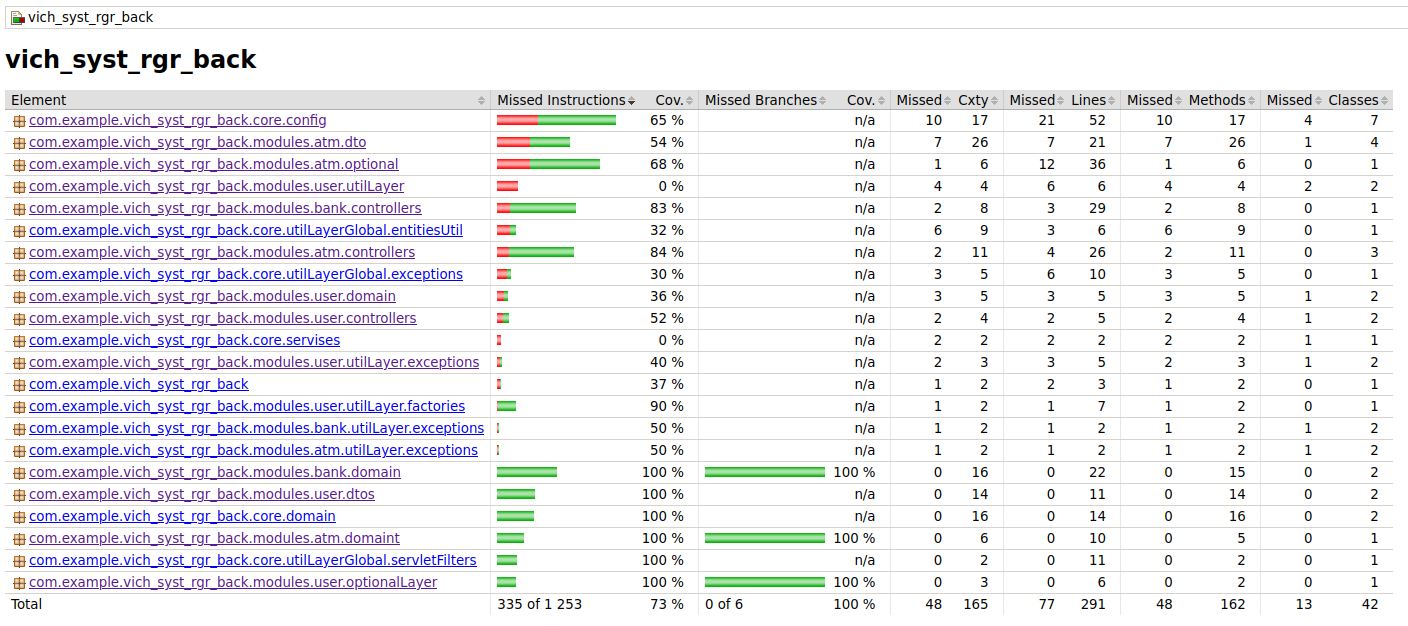


Рисунок 5 - html страница-отчет jacoco об покрытии сервера тестами.

# 5.1 Описание процессов тестирования

Будет произведена разработка автоматических тестов. Какие ситуации возникают при работе с сервисом:

1. Авторизация;
2. Получение информации;
3. Запись информации;
4. Перемещение между страницами сервиса.

В зависимости от результатов первого пункта меняются остальные. Поэтому тестировать необходимо:

1. Чтение/запись данных;
2. Перемещение между страницами.

Соответствующие тесты будут произведены от неавторизованного и авторизованного пользователей (которые, в свою очередь, делятся на обычного пользователя и администратора).

Чтение и запись относится к работе с БД в MVC – модели это больше подходит модели, поэтому и тестировать этот функционал будем в модели.

Перемещение отнесем к контроллерам и рендерингу View, поэтому тестировать будем в контроллерах (а точнее в request в контексте контроллера).

*Таблица 1 - Возможности пользователя в зависимости от авторизации.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | View | | | Возможности | | |
| Guest  View | User  View | Admin  View | Изменение денег в банкомате | Снятие наличных | Пополнение счета |
| Guest | + |  |  |  |  |  |
| User |  | + |  |  | + | + |
| Admin |  |  | + | + |  |  |

Тестироваться будет веб-приложение «CuteBank». Определим объекты тестирования:

* Модели
  + «CashMachine»;
  + «User»;
  + «UserHistoryRecord».
* Контроллеры
* «AdminController»;
* «UserController»;
* «MainController».

# 

# 

# 5.2 Тесты

Показать всех пользователей (успешно)

/api/bankUser/accounts/allId

args = ('postgres3', '12345678')

expected :

users and status code 200

response :

[{'username': 'postgres3', 'id': '1'}, {'username': 'postgres4', 'id': '2'}, {'username': 'username1', 'id': '3'}]

show showAllBankUsers status code 200

Показать баланс (успешно)

/api/bankUser/atm/balance{}

args = ('postgres3', '12345678') accID= 1

expected :

balance and status code 200

response :

balance 0.00

show balance status code 200

Депозит (успешно)

/api/bankUser/atm/deposit

args = ('postgres3', '12345678')accID= 1{'accountId': 1, 'atmId': 1, 'toAdd': 20000}

expected : status code 200

response :

deposit status code 200

Показать баланс (успешно)

/api/bankUser/atm/balance{}

args = ('postgres3', '12345678') accID= 1

expected :

balance and status code 200

response :

balance 20000.00

show balance status code 200

Совершить перевод (успешно)

/api/bankUser/atm/transfer

args = ('postgres3', '12345678')accID= 1{'amount': 20000, 'fromAccountId': 1, 'toAccountId': 2}

expected : status code 200

transfer status code 200

Показать баланс (успешно)

/api/bankUser/atm/balance{}

args = ('postgres3', '12345678') accID= 1

expected :

balance and status code 200

response :

balance 0.00

show balance status code 200

(Отрицательный)

/api/bankUser/atm/withdraw

args = ('postgres3', '12345678')accID= 1{'accountId': 1, 'atmId': 1, 'toSub': 10000}

expected : status code 200

withdrw status code 400

Показать баланс (успешно)

/api/bankUser/atm/balance{}

args = ('postgres3', '12345678') accID= 1

expected :

balance and status code 200

response :

balance 0.00

show balance status code 200

Добавить (успешно)

/api/collector/atmCash/add

args = ('collector', '12345678') {'atmId': 1, 'toAdd': 20000}

expected : status code 200

add status code 200

Показать статус (успешно)

/api/collector/atm/allId

args = ('collector', '12345678')

expected : show atms and status code 200

show all status code 200

atms [1,2,3,4]

Удалить статус (успешно)

/api/collector/atm/{}

args = ('collector', '12345678')id- 2

expected : status code 200

delete status code 200

Регистрация (Отрицательный)

/api/reg/bankUser

args = {'password': 'qwerty123', 'username': 'username2'}

expected : status code 200

registration status code 201

Показать банковские счета текущего пользователя (успешно)

/api/bankUser/accounts/allId

args = ('postgres3', '12345678')

expected : ids and status code 200

response :

show bankAccountsOfCurrentUser status code 200

ids [1,2,3,4,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23]

Показать всех пользователей банка (Отрицательный)

/api/bankUser/accounts/allId

args = ('postgres3', '1234567')

expected :

users and status code 200

response :

{'timestamp': '2020-12-02T11:36:25.072+00:00', 'status': 401, 'error': 'Unauthorized', 'message': '', 'path': '/api/user/util/allBankUsers'}

show showAllBankUsers status code 401

Показать баланс (Отрицательный)

/api/bankUser/atm/balance{}

args = ('postgres3', '1234567') accID= None

expected :

balance and status code 200

response :

balance {"timestamp":"2020-12-02T11:36:26.799+00:00","status":401,"error":"Unauthorized","message":"","path":"/api/bankUser/atm/balanceNone"}

show balance status code 401

Перевод статус (Отрицательный)

/api/bankUser/atm/deposit

args = ('postgres3', '1234567')accID= None{'accountId': None, 'atmId': 1, 'toAdd': 20000}

expected : status code 200

response :

deposit status code 401

Показать баланс (Отрицательный)

/api/bankUser/atm/balance{}

args = ('postgres3', '1234567') accID= None

expected :

balance and status code 200

response :

balance {"timestamp":"2020-12-02T11:36:29.314+00:00","status":401,"error":"Unauthorized","message":"","path":"/api/bankUser/atm/balanceNone"}

show balance status code 401

Показать статус перевода (Отрицательный)

/api/bankUser/atm/transfer

args = ('postgres3', '1234567')accID= None{'amount': 20000, 'fromAccountId': None, 'toAccountId': 2}

expected : status code 200

transfer status code 401

Показать баланс (Отрицательный)

/api/bankUser/atm/balance{}

args = ('postgres3', '1234567') accID= None

expected :

balance and status code 200

response :

balance {"timestamp":"2020-12-02T11:36:32.008+00:00","status":401,"error":"Unauthorized","message":"","path":"/api/bankUser/atm/balanceNone"}

show balance status code 401

(Отрицательный)

/api/bankUser/atm/withdraw

args = ('postgres3', '1234567')accID= None{'accountId': None, 'atmId': 1, 'toSub': 10000}

expected : status code 200

withdrw status code 401

Показать баланс (Отрицательный)

/api/bankUser/atm/balance{}

args = ('postgres3', '1234567') accID= None

expected :

balance and status code 200

response :

balance {"timestamp":"2020-12-02T11:36:34.535+00:00","status":401,"error":"Unauthorized","message":"","path":"/api/bankUser/atm/balanceNone"}

show balance status code 401

Добавить (Отрицательный)

/api/collector/atmCash/add

args = ('collector', '1234567') {'atmId': 1, 'toAdd': 20000}

expected : status code 200

add status code 401

Показать атм (Отрицательный)

/api/collector/atm/allId

args = ('collector', '1234567')

expected : show atms and status code 200

show all status code 401

atms {"timestamp":"2020-12-02T11:36:36.225+00:00","status":401,"error":"Unauthorized","message":"","path":"/api/collector/atm/allId"}

Удалить (Отрицательный)

/api/collector/atm/{}

args = ('collector', '1234567')id- 1

expected : status code 200

delete status code 401

Регестрация (Отрицательный)

/api/reg/bankUser

args = {'password': 'qwerty123', 'username': 'username1'}

expected : status code 200

registration status code 409

Показать банковские счета текущего пользователя (Отрицательный)

/api/bankUser/accounts/allId

args = ('postgres3', '1234567')

expected : ids and status code 200

response :

show bankAccountsOfCurrentUser status code 401

ids {"timestamp":"2020-12-02T11:36:38.801+00:00","status":401,"error":"Unauthorized","message":"","path":"/api/bankUser/accounts/allId"}

Показать всех пользователей банка (Успешно)

/api/bankUser/accounts/allId

args = ('postgres3', '12345678')

expected :

users and status code 200

response :

[{'username': 'postgres3', 'id': '1'}, {'username': 'postgres4', 'id': '2'}, {'username': 'username1', 'id': '3'}, {'username': 'username2', 'id': '4'}]

show showAllBankUsers status code 200

Показать баланс (успешно)

/api/bankUser/atm/balance{}

args = ('postgres3', '12345678') accID= 1

expected :

balance and status code 200

response :

balance 0.00

show balance status code 200

Вклад (успешно)

/api/bankUser/atm/deposit

args = ('postgres3', '12345678')accID= 1{'accountId': 1, 'atmId': 1, 'toAdd': -20000}

expected : status code 200

response :

deposit status code 200

Показать баланс (успешно)

/api/bankUser/atm/balance{}

args = ('postgres3', '12345678') accID= 1

expected :

balance and status code 200

response :

balance -20000.00

show balance status code 200

Перевод (успешно)

/api/bankUser/atm/transfer

args = ('postgres3', '12345678')accID= 1{'amount': -20000, 'fromAccountId': 1, 'toAccountId': 2}

expected : status code 200

transfer status code 200

Показать баланс (успешно)

/api/bankUser/atm/balance{}

args = ('postgres3', '12345678') accID= 1

expected :

balance and status code 200

response :

balance 0.00

show balance status code 200

(Успешно)

/api/bankUser/atm/withdraw

args = ('postgres3', '12345678')accID= 1{'accountId': 1, 'atmId': 1, 'toSub': -10000}

expected : status code 200

withdrw status code 200

Показать баланс (Успешно)

/api/bankUser/atm/balance{}

args = ('postgres3', '12345678') accID= 1

expected :

balance and status code 200

response :

balance 10000.00

show balance status code 200

Добавить (Успешно)

/api/collector/atmCash/add

args = ('collector', '12345678') {'atmId': 1, 'toAdd': 20000}

expected : status code 200

add status code 200

/api/collector/atm/allId

Показать все (Успешно)

args = ('collector', '12345678')

expected : show atms and status code 200

show all status code 200

atms [1,2,3,4]

/api/collector/atm/{}

Удалить (Успешно)

args = ('collector', '12345678')id- 3

expected : status code 200

delete status code 200

/api/reg/bankUser

Регистрация (Отрицательный)

args = {'password': 'qwerty123', 'username': 'username1'}

expected : status code 200

registration status code 409

/api/bankUser/accounts/allId

Показать банковские счета текущего пользователя (успешно)

args = ('postgres3', '12345678')

expected : ids and status code 200

response :

show bankAccountsOfCurrentUser status code 200

ids [1,2,3,4,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23]

В ходе проведенного тестирования обнаружена и исправлена небольшая неточность. Конечный разработанный продукт, по результатам тестирования, пригоден к эксплуатации. Проблем в логике работы не обнаружено.

### 6. Развертывание.

**Платформа развертывания:**

Для развертывания сервера был выбран Heroku, так как он предоставляет бесплатный хостинг и поддерживает java11. БД тоже была создана на хероку, так как он также предоставляет бесплатный хостинг БД на основе PostgreSQL. Под фронт, так как он будет доступен только с терминалов, было решено пока сервер не выделять, а запускать его локально на каждом из терминалов-банкоматов, но в случае необходимости можно создать отдельный сервер для фронт на основе apache или njinx.

**Использование развертывания в разработке:**

Так как приложение основано на принципе API, то фронт-разработчикам нужно иметь доступ к серверу для создания и тестирования AJAX запросов. Для этого можно запускать сервер локально и обращаться к нему как к localhost, но этот подход имеет ряд неудобств:

* разработчик должен сам следить за актуальностью версии сервера на своей машине
* разработчик должен знать как запустить сервер и иметь для этого все необходимые утилиты, которыми ему не нужно уметь пользоваться для прямого выполнения своих задач.

Потому было принято решение деплоить новую версию сервера каждый раз, когда меняется интерфейс обращения к контроллеру, или контроллер создается/удадяется. Это позволяет фронт-энд рабработчику рабоать исключительно с актуальной версией сервера.