

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
Национальный исследовательский ядерный университет «МИФИ» (НИЯУ МИФИ)

ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

ОТЧЕТ ПО ИТОГОВОМУ ПРОЕКТУ
СИСТЕМА УПРАВЛЕНИЯ ЛИЧНЫМИ ФИНАНСАМИ
по дисциплине «Принципы объектно-ориентированного программирования»

Выполнил студент группы: М24-535:

Почернин В. С.

Преподаватели:

Калин А.
Хлебников А.

3 января 2025 г.

Санкт-Петербург
2025

Содержание

1	Описание проекта	2
1.1	Используемые технологии	2
1.2	Описание модулей	2
1.3	Схема БД	2
2	Инструкции по запуску кода	5
3	Демонстрация работы	7
3.1	Вывод сообщения помощи (help)	7
3.2	Регистрация (register)	7
3.3	Аутентификация (login)	8
3.4	Создание категории (category)	8
3.5	Установка бюджета (budget)	9
3.6	Добавление транзакции (transaction)	10
3.7	Вывод подробной информации по доходам/расходам/бюджетам (info) . .	12
3.8	Вывод подробной информации по доходам/расходам/бюджетам в файл (info-file)	12
3.9	Вывод подробной информации по конкретным категориям (info-certain) .	13
3.10	Выход из программы (exit)	13
4	Соответствие техническим требованиям	14
4.1	Хранение данных	14
4.2	Авторизация пользователей	14
4.3	Функционал управления финансами	14
4.4	Работа с кошельком пользователя	14
4.5	Вывод информации	14
4.6	Подсчет доходов и расходов	14
4.7	Проверка вводимых данных	15
4.8	Оповещения	15
4.9	Сохранение данных	15
4.10	Чтение команд пользователя в цикле	15
5	Соответствие критериям оценивания	16
5.1	Реализация авторизации пользователей	16
5.2	Взаимодействие с пользователем	16
5.3	Управление доходами и расходами	16
5.4	Работа с кошельком пользователя	16
5.5	Вывод информации	16
5.6	Оповещения пользователя	16
5.7	Сохранение и загрузка данных	16
5.8	Чтение команд в цикле	16
5.9	Валидация данных	16
5.10	Дополнительные возможности	16
5.11	Разделение функционала по классам	17

1 Описание проекта

Система управления личными финансами представляет собой приложение, предоставляющее возможность пользователям добавлять доходы и расходы, просматривать статистику по финансам, а также устанавливать бюджеты и категории.

Приложение написано в строгом соответствии с критериями, описанными в `README.md` (соответствие критериям будет показано ниже).

1.1 Используемые технологии

- Java - язык программирования.
- Spring - основной фреймворк.
- PostgreSQL - используемая СУБД.
- Maven - система сборки проекта.
- Flyway - инструмент для применения файлов миграций БД.
- Lombok - библиотека для сокращения шаблонного кода.
- Bcrypt - инструмент для хеширования паролей.

1.2 Описание модулей

Вкратце рассмотрим кодовую базу проекта:.

- Пакет `command`: содержит в себе классы, отвечающие за работу интерфейса командной строки, такие как сама команда, типы команд, а также сам класс интерфейса.
- Пакет `context`: содержит класс, отвечающий за переменные контекста приложения.
- Пакет `entity`: содержит классы моделей данных (связанных с базой при помощи ORM).
- Пакет `exception`: содержит в себе класс-исключение, связанное с системой управления личными финансами.
- Пакет `handler`: содержит в себе классы, реализующие паттерн Стратегия и используемые для обработки команд.
- Пакет `repository`: содержит в себе классы - JPA репозитории.
- Пакет `utils`: содержит в себе различные утильные классы.
- Папка `migration`: содержит в себе файлы миграции БД, которые устанавливают схему и наполняют базу тестовыми данными.

1.3 Схема БД

База данных проекта имеет следующую схему:

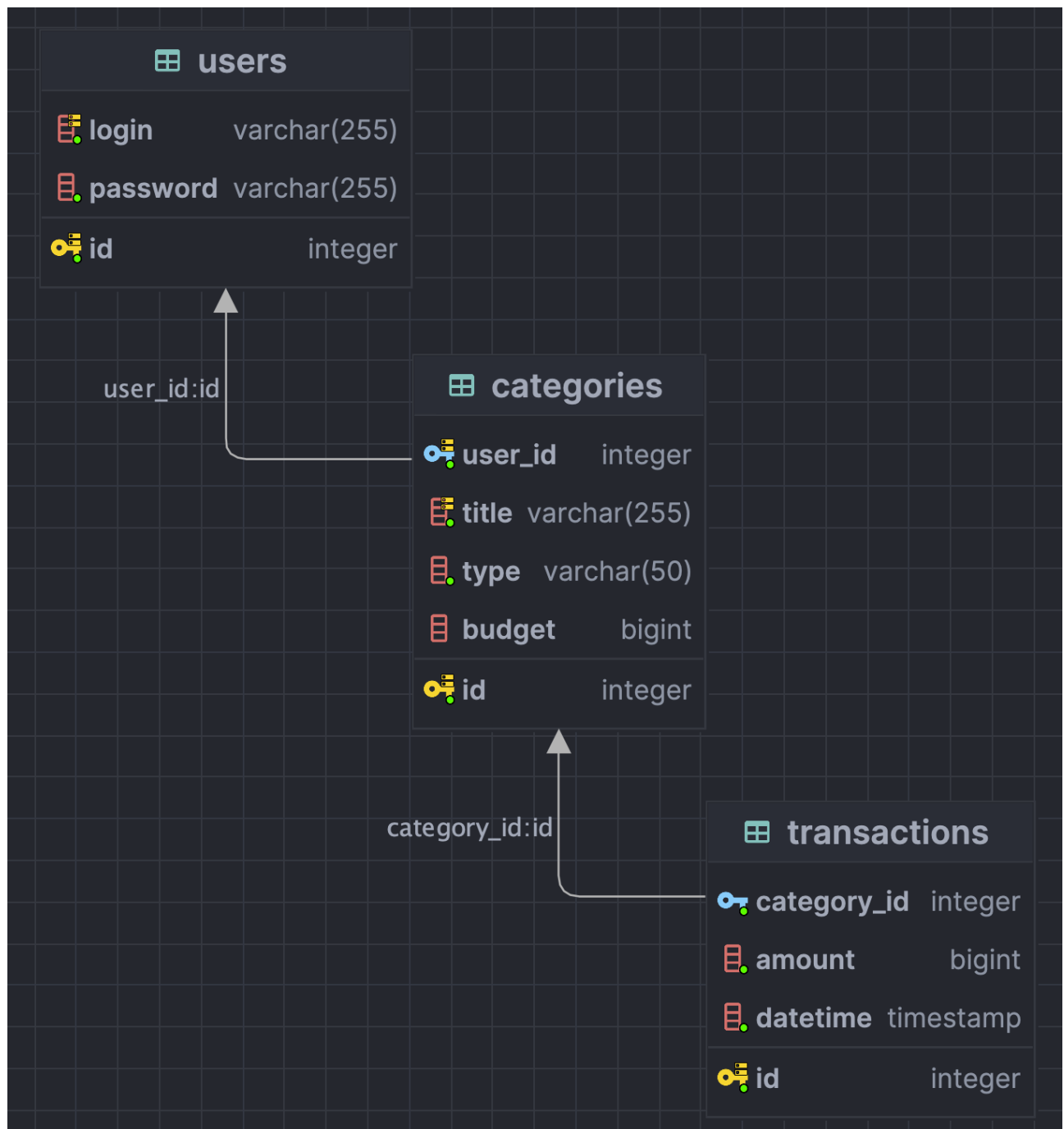


Рис. 1: Схема БД

- Таблица **users** содержит информацию о пользователях.
 - **id** выступает ключом записи.
 - **login** - логин, присутствует требование уникальности.
 - **password** - пароль, будет храниться в зашифрованном виде.
- Таблица **categories** содержит информацию о категориях пользователей.
 - **id** выступает ключом записи.
 - **user_id** - ссылка на таблицу **users**, у каждого пользователя свои категории.
 - **title** - название категории.

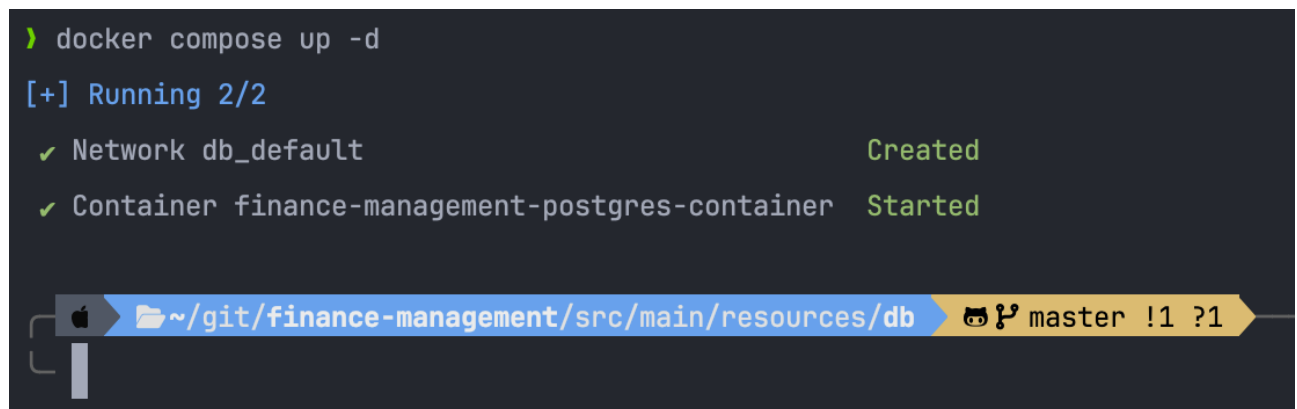
- **type** - тип категории, возможные значения **INCOME** (доход) или **EXPENSE** (расходы).
- **budget** - бюджет на категорию в копейках, может отсутствовать.
- Присутствует требование уникальности по паре (**user_id**, **title**).
- Таблица **transactions** содержит информацию о конкретных транзакциях (доходах, расходах).
 - **id** выступает ключом записи.
 - **category_id** - ссылка на таблицу **categories**. Каждая транзакция связана с какой-либо категорией.
 - **amount** - сумма транзакции в копейках.
 - **datetime** - метка времени совершения транзакции (по умолчанию выставляется **current_timestamp**).

2 Инструкции по запуску кода

Для того, чтобы запустить код, необходимо иметь

- Java 17 (полагаю, что версии выше также подойдут).
- Docker (необходим для быстрого поднятия БД, но можно поднимать руками).
- Maven (поддерживается в IntelliJ IDEA).

Запустим наш проект. Для начала, перейдем в директорию, содержащую конфигурационный файл докера - `/src/main/resources/db`, затем запустим базу данных в `detach` режиме командой `docker compose up -d`



```
> docker compose up -d
[+] Running 2/2
 ✓ Network db_default          Created
 ✓ Container finance-management-postgres-container Started

~/git/finance-management/src/main/resources/db master !1 ?1
```

Рис. 2: Запуск БД

Чтобы запустить проект, необходимо запустить `main()` метод из класса `FinanceManagementApplication` с помощью IntelliJ IDEA (идея при запуске проекта должна автоматически скачать все зависимости).

После запуска кода мы увидим приветственное сообщение с описанием всех команд, а также требованиями к некоторым аргументам.

PS. Вероятно, это относится только к моему компьютеру, но я получал ошибку при запуске, пока не добавил в конфигурацию запуска в VM Options

```
-Djava.rmi.server.hostname=localhost
```

```
Run FinanceManagementApplication x
>> Console Actuator
2025-01-03T12:28:28.251+03:00 INFO 65826 --- [finance-management] [main] o.h.e.t.j.p.i.JtaPlatformInitiator
2025-01-03T12:28:28.252+03:00 INFO 65826 --- [finance-management] [main] j.LocalContainerEntityManagerFactoryBean
2025-01-03T12:28:28.384+03:00 INFO 65826 --- [finance-management] [main] r.v.f.FinanceManagementApplication
Система управления личными финансами поддерживает следующие команды:
-----
'help' - вывод сообщения помощи,
'register [логин] [пароль]' - зарегистрироваться,
'login [логин] [пароль]' - пройти аутентификацию,
'category [тип категории] [название категории]' - создать категорию,
'budget [название категории расходов] [денежное значение]' - установить бюджет на категорию,
'transaction [название категории] [денежное значение]' - добавить транзакцию (доход/расход),
'info' - получить подробную информацию по доходам/расходам/бюджетам,
'info-file [абсолютный путь до несуществующего файла]' - вывести подробную информацию по доходам/расходам/бюджетам в файл,
'info-certain [...названия категорий...]' - вывести подробную информацию по конкретным категориям,
'exit' - выйти из программы,
-----
- Логин должен содержать от 3 до 32 символов
- Пароль должен содержать от 6 до 32 символов
- Название категории не должно превышать 50 символов
- Если название категории состоит из нескольких слов, слова должны быть разделены символом "-"
- Тип категории должен быть одним из двух значений: income (доход), либо expense (расход)
- Любое денежное значение должна быть введена либо целым числом (рубли), либо числом вида xxx.yy (рубли.копейки),
где xxx (многозначное) - количество рублей, yy - количество копеек (обязательно двузначное), при этом
строка не должна быть длиннее 15 символов
- Денежное значение должно быть положительным числом
-----
-----
Введите очередную команду [неидентифицирован]:
|
```

Рис. 3: Запуск кода

3 Демонстрация работы

Проведем демонстрацию работы всех команд приложения.

3.1 Вывод сообщения помощи (help)

Команда `help` выводит сообщение с описанием всех команд, а также с требованиями к некоторым аргументам.

```
-----
Введите очередную команду [неидентифицирован]:
help
Система управления личными финансами поддерживает следующие команды:
-----
'help' - вывод сообщения помощи,
'register [логин] [пароль]' - зарегистрироваться,
'login [логин] [пароль]' - пройти аутентификацию,
'category [тип категории] [название категории]' - создать категорию,
'budget [название категории расходов] [денежное значение]' - установить бюджет на категорию,
'transaction [название категории] [денежное значение]' - добавить транзакцию (доход/расход),
'info' - получить подробную информацию по доходам/расходам/бюджетам,
'info-file [абсолютный путь до несуществующего файла]' - вывести подробную информацию по доходам/расходам/бюджетам в файл,
'info-certain [...названия категорий...]' - вывести подробную информацию по конкретным категориям,
'exit' - выйти из программы,
-----
- Логин должен содержать от 3 до 32 символов
- Пароль должен содержать от 6 до 32 символов
- Название категории не должно превышать 50 символов
- Если название категории состоит из нескольких слов, слова должны быть разделены символом "-"
- Тип категории должен быть одним из двух значений: income (доход), либо expense (расход)
- Любое денежное значение должна быть введено либо целым числом (рубли), либо числом вида xxx.yy (рубли.копейки),
где xxx (многозначное) - количество рублей, yy - количество копеек (обязательно двузначное), при этом
строка не должна быть длиннее 15 символов
- Денежное значение должно быть положительным числом
-----
Введите очередную команду [неидентифицирован]:
|
```

Рис. 4: Вывод сообщения помощи

3.2 Регистрация (register)

Команда `register` регистрирует нового пользователя в системе. Пароль при этом хэшируется алгоритмом `Bcrypt`.


```
-----  
Введите очередную команду [неидентифицирован]:  
register vspochernin password  
Регистрация прошла успешно  
-----  
Введите очередную команду [неидентифицирован]:
```

Рис. 5: Проведение регистрации

3.3 Аутентификация (login)

Команда `login` проводит аутентификацию пользователя. При этом сравниваются хеши переданного и сохраненного в базе паролей.

```
-----  
Введите очередную команду [неидентифицирован]:  
login vspochernin password  
Аутентификация прошла успешно  
-----  
Введите очередную команду [vspochernin]:
```

Рис. 6: Проведение аутентификации

3.4 Создание категории (category)

Команда `category` создает категорию дохода или расхода. Создадим категории согласно примеру из `README.md`:

```
-----  
Введите очередную команду [vspochernin]:  
category expense Еда  
Категория успешно создана  
-----  
Введите очередную команду [vspochernin]:  
category expense Развлечения  
Категория успешно создана  
-----  
Введите очередную команду [vspochernin]:  
category expense Коммунальные-услуги  
Категория успешно создана  
-----  
Введите очередную команду [vspochernin]:  
category expense Такси  
Категория успешно создана  
-----  
Введите очередную команду [vspochernin]:  
category income Зарплата  
Категория успешно создана  
-----  
Введите очередную команду [vspochernin]:  
category income Бонус  
Категория успешно создана  
-----  
Введите очередную команду [vspochernin]:
```

Рис. 7: Создание категорий

3.5 Установка бюджета (budget)

Команда `budget` устанавливает бюджет на определенную категорию расхода. Установим бюджеты согласно примеру из `README.md`.

```
-----  
Введите очередную команду [vspochernin]:  
budget Еда 4000  
Бюджет успешно установлен  
-----  
Введите очередную команду [vspochernin]:  
budget Развлечения 3000  
Бюджет успешно установлен  
-----  
Введите очередную команду [vspochernin]:  
budget Коммунальные-услуги 2500  
Бюджет успешно установлен  
-----  
Введите очередную команду [vspochernin]:
```

Рис. 8: Установка бюджетов

3.6 Добавление транзакции (transaction)

Команда `transaction` добавляет транзакцию по одной из категории дохода или расхода. Добавим транзакции согласно примеру из `README.md`.

Также здесь можно будет заметить уведомления о превышении расходов над доходами, а также о превышении конкретного установленного бюджета.

```

-----
Введите очередную команду [vsPOCHernin]:
transaction Еда 300
Транзакция успешно записана
Внимание! Расходы превысили доходы: (300.00/0.00)
-----
Введите очередную команду [vsPOCHernin]:
transaction Еда 500
Транзакция успешно записана
Внимание! Расходы превысили доходы: (800.00/0.00)
-----
Введите очередную команду [vsPOCHernin]:
transaction Развлечения 3000
Транзакция успешно записана
Внимание! Расходы превысили доходы: (3800.00/0.00)
-----
Введите очередную команду [vsPOCHernin]:
transaction Коммунальные-услуги 3000
Транзакция успешно записана
Внимание! Превышен бюджет по категории: Коммунальные услуги (3000.00/2500.00)
Внимание! Расходы превысили доходы: (6800.00/0.00)
-----
Введите очередную команду [vsPOCHernin]:
transaction Такси 1500
Транзакция успешно записана
Внимание! Превышен бюджет по категории: Коммунальные услуги (3000.00/2500.00)
Внимание! Расходы превысили доходы: (8300.00/0.00)
-----
Введите очередную команду [vsPOCHernin]:
transaction Зарплата 20000
Транзакция успешно записана
Внимание! Превышен бюджет по категории: Коммунальные услуги (3000.00/2500.00)
-----
Введите очередную команду [vsPOCHernin]:
transaction Зарплата 40000
Транзакция успешно записана
Внимание! Превышен бюджет по категории: Коммунальные услуги (3000.00/2500.00)
-----
Введите очередную команду [vsPOCHernin]:
transaction Бонус 3000
Транзакция успешно записана
Внимание! Превышен бюджет по категории: Коммунальные услуги (3000.00/2500.00)
-----
Введите очередную команду [vsPOCHernin]:
|

```

Рис. 9: Добавление транзакций

3.7 Вывод подробной информации по доходам/расходам/бюджетам (info)

Команда `info` выводит подробную информацию по доходам/расходам/бюджетам пользователя. Убедимся, что ее вывод будет таким же, какой ожидается в `README.md`:

```
-----
Введите очередную команду [vspochernin]:
info
Общий доход: 63000.00
Доходы по категориям:
  - Бонус: 3000.00
  - Зарплата: 60000.00
Общие расходы: 8300.00
Расходы по категориям:
  - Коммунальные услуги: 3000.00
  - Такси: 1500.00
  - Развлечения: 3000.00
  - Еда: 800.00
Бюджет по категориям:
  - Коммунальные услуги: 2500.00, оставшийся бюджет: -500.00
  - Развлечения: 3000.00, оставшийся бюджет: 0.00
  - Еда: 4000.00, оставшийся бюджет: 3200.00

Внимание! Превышен бюджет по категории: Коммунальные услуги (3000.00/2500.00)
-----
Введите очередную команду [vspochernin]:
```

Рис. 10: Вывод подробной информации по доходам/расходам/бюджетам

3.8 Вывод подробной информации по доходам/расходам/бюджетам в файл (info-file)

Команда `info-file` повторяет работу команды `info`, однако выводит результат своей работы в файл.

```
-----
Введите очередную команду [vspochernin]:
info-file /Users/pochernin-vla/finance-info.txt
Информация успешно записана в файл
Внимание! Превышен бюджет по категории: Коммунальные услуги (3000.00/2500.00)
-----
Введите очередную команду [vspochernin]:
```

Рис. 11: Вывод подробной информации по доходам/расходам/бюджетам в файл

Проверим содержимое файла:

```

> cat /Users/pochernin-vla/finance-info.txt
Общий доход: 63000.00
Доходы по категориям:
  - Бонус: 3000.00
  - Зарплата: 60000.00
Общие расходы: 8300.00
Расходы по категориям:
  - Коммунальные услуги: 3000.00
  - Такси: 1500.00
  - Развлечения: 3000.00
  - Еда: 800.00
Бюджет по категориям:
  - Коммунальные услуги: 2500.00, оставшийся бюджет: -500.00
  - Развлечения: 3000.00, оставшийся бюджет: 0.00
  - Еда: 4000.00, оставшийся бюджет: 3200.00

```

Рис. 12: Проверка содержимого файла

3.9 Вывод подробной информации по конкретным категориям (info-certain)

Команда `info-certain` выводит подробную информацию о транзакциях по конкретным категориям, которые передаются аргументами. Также она сигнализирует, если категория отсутствует.

Посмотрим ее вывод для трех категорий, две из которых существуют, а одна - нет:

```

-----
Введите очередную команду [vsPOCHERNIN]:
info-certain Зарплата Еда Отпуск
- Категория: Зарплата, тип: доход, общая сумма: 60000.00
  - Транзакция на сумму: 20000.00, произведена: 03.01.2025 в 13:23
  - Транзакция на сумму: 40000.00, произведена: 03.01.2025 в 13:23
- Категория: Еда, тип: расход, общая сумма: 800.00
  - Транзакция на сумму: 300.00, произведена: 03.01.2025 в 13:22
  - Транзакция на сумму: 500.00, произведена: 03.01.2025 в 13:22
- Категории Отпуск не найдено
Внимание! Превышен бюджет по категории: Коммунальные услуги (3000.00/2500.00)
-----
Введите очередную команду [vsPOCHERNIN]:

```

Рис. 13: Вывод подробной информации по конкретным категориям

3.10 Выход из программы (exit)

Наконец, команда `exit` осуществляет выход из программы:

```

-----
Введите очередную команду [vsPOCHERNIN]:
exit
Будет выполнен выход из программы
2025-01-03T13:41:02.833+03:00 INFO 25483 --- [finance-management] [onShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2025-01-03T13:41:02.839+03:00 INFO 25483 --- [finance-management] [onShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2025-01-03T13:41:02.840+03:00 INFO 25483 --- [finance-management] [onShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
Process finished with exit code 0

```

Рис. 14: Выход из программы

4 Соответствие техническим требованиям

Ниже будут даны комментарии относительно соответствия программы техническим требованиям.

4.1 Хранение данных

- Все данные должны храниться в памяти приложения: данное требования реализован с помощью использования БД PostgreSQL.

4.2 Авторизация пользователей

- Реализовать функциональность для авторизации пользователей по логину и паролю. Приложение должно поддерживать несколько пользователей: соответствие данному требованию можно увидеть в работе команд `login` и `register`.

4.3 Функционал управления финансами

- Разработать логику для добавления доходов и расходов. Пользователь должен иметь возможность создавать категории для планирования бюджета: соответствие данному требованию можно увидеть в работе команды `category`.
- Предусмотреть функциональность для установления бюджета на каждую категорию расходов: соответствие данному требованию можно увидеть в работе команды `budget`.

4.4 Работа с кошельком пользователя

- Привязать кошелек к авторизованному пользователю. Кошелек должен хранить информацию о текущих финансах и всех операциях (доходах и расходах): кошельком пользователя является его учетная запись, в ней пользователь создает различные категории, для которых добавляет транзакции. Категории привязаны непосредственно к пользователю, транзакции к конкретной категории.

4.5 Вывод информации

- Реализовать возможность отображения общей суммы доходов и расходов, а также данных по каждой категории: соответствие данному требованию можно увидеть в работе команды `info`.
- Выводить информацию о текущем состоянии бюджета для каждой категории, а также оставшийся лимит: соответствие данному требованию можно увидеть в работе команды `info`.
- Поддерживать вывод информации в терминал или в файл: соответствие данному требованию можно увидеть в работе команды `info-file`.

4.6 Подсчет доходов и расходов

- Разработать методы, подсчитывающие общие расходы и доходы, а также по категориям соответствие данному требованию можно увидеть в работе команды `info`.
- Поддерживать возможность подсчета по нескольким выбранным категориям. Если категория не найдена, уведомлять пользователя: соответствие данному требованию можно увидеть в работе команды `info-certain`.

4.7 Проверка вводимых данных

- Валидация пользовательского ввода и уведомление о некорректных данных: Для каждой команды проводится тщательная валидация ввода. Для этого используются методы класса `ValidationUtils`, а также специфические валидации (например, в методе `parse` класса `MoneyUtils`).

4.8 Оповещения

- Оповещать пользователя, если превышен лимит бюджета по категории или расходы превысили доходы: в случае, если превышен лимит бюджета по категории или расходы превысили доходы, пользователь будет уведомлен об этом перед вводом очередной команды. Соответствие данному требованию можно, например, увидеть в проверке команды `transaction` (пункт 3.6 отчета).

4.9 Сохранение данных

- При выходе из приложения сохранять данные кошелька пользователя в файл: данное требование реализовано с помощью использования БД PostgreSQL.
- При авторизации загружать данные кошелька из файла: данное требование реализовано с помощью использования БД PostgreSQL.

4.10 Чтение команд пользователя в цикле

- Реализовать цикл для постоянного чтения команд пользователя. Поддержать возможность выхода из приложения: данное требование реализовано с помощью классов команд (пакет `command`), а также использования паттерна Стратегия (пакет `handler`).

5 Соответствие критериям оценивания

Ниже будут даны комментарии относительно соответствия программы критериям оценивания.

5.1 Реализация авторизации пользователей

Пользователь может зарегистрироваться и войти в систему. Только аутентифицированный пользователь может полноценно работать с программой.

5.2 Взаимодействие с пользователем

Взаимодействие с пользователем осуществляется через консоль.

5.3 Управление доходами и расходами

В программе реализовано управление доходами и расходами: их добавление, установка бюджета на конкретные категории, вывод отчетов.

5.4 Работа с кошельком пользователя

Пользователь может добавлять в свой кошелек категории доходов или расходов, устанавливать бюджет и добавлять транзакции.

5.5 Вывод информации

Реализован подробный вывод информации как в консоль, так и в файл.

5.6 Оповещения пользователя

Перед вводом команды пользователь оповещается о возможном превышении расходов над доходами, а также о превышении конкретных бюджетов.

5.7 Сохранение и загрузка данных

Для сохранения и загрузки данных реализовано взаимодействие с БД PostgreSQL.

5.8 Чтение команд в цикле

Команды пользователя считываются в бесконечном цикле, выход из которого происходит после ввода команды `exit`.

5.9 Валидация данных

Вводимые пользователем данные тщательно валидируются как базовыми методами (например, проверка количества аргументов для команды), так и специфическими (например, проверка корректности ввода денежного значения).

5.10 Дополнительные возможности

Креативным решением данного приложения предлагаю считать использование паттерна **Стратегия**, что подчеркивает использование практик чистого кода и принципов ООП и позволяет быстро и удобно добавлять новые команды.

5.11 Разделение функционала по классам

Код приложения тщательно разделен на классы и пакеты для удобочитаемости и облегчения поддержки.