**Федеральное государственное образовательное бюджетное учреждение высшего образования**

**«ФинансовЫЙ УНИВЕРСИТЕТ при Правительстве Российской Федерации»**

Факультет информационных технологий и анализа больших данных

**КУРСОВАЯ работа**

на тему:

«Разработка программного обеспечения для учета денежных средств организации»

Руководитель:

Профессор

Чистов Дмитрий Владимирович

Выполнил:

Прудников Матвей Денисович

ПИ21-1

Москва 2023

Оглавление

[ВВЕДЕНИЕ 3](#_Toc134778484)

[ОСНОВНАЯ ЧАСТЬ 4](#_Toc134778485)

[Глава 2. Описание предметной области 4](#_Toc134778486)

[Глава 3. Актуальность автоматизации 5](#_Toc134778487)

[Глава 4. Описание программы 5](#_Toc134778488)

[4.1. Алгоритмические решения 5](#_Toc134778489)

[4.2. Описание интерфейса программы 8](#_Toc134778490)

[4.3. Состав приложения 15](#_Toc134778491)

[4.3. Деплой на сервер 18](#_Toc134778492)

[ЗАКЛЮЧЕНИЕ 20](#_Toc134778493)

[СПИСОК ЛИТЕРАТУРЫ 21](#_Toc134778494)

[ПРИЛОЖЕНИЕ 22](#_Toc134778495)

# ВВЕДЕНИЕ

Ведение учета денежных средств – неотъемлемая часть любой бизнес-стратегии. Отношения денежных потоков должны регулироваться в соответствии с правилами и законами, принятыми в организации. Это позволяет не только вести надлежащий учет средств, но и способствует максимальной эффективности финансового управления. Правильно проведенный учет денежных средств помогает организации правильно планировать свои размеры прибыли, создавать реалистичные цели и более точно оценивать производительность своего бизнеса.

В настоящее время во многих организациях есть потребность в системе учета денежных средств. Для эффективной работы такой системы необходимо разработать программное обеспечение. Целью данного курсового проекта является разработка программного обеспечения для решения задач учета денежных средств.

Основные функции, которые должна обеспечивать программа для учета денежных средств, включают: ввод и обработку данных, суммирование, а также любые другие необходимые функции. Программа должна быть удобной и легкой для использования. Она должна поддерживать несколько форматов данных.

Многие программы такого типа уже устарели и необходимо создать что-то новое. Поэтому мной была выбрана данная тема для курсовой работы.

Задачи, по которым велась работа в ходе разработки данного приложения:

* Погрузиться в предметную область и изучить принципы учёта средств
* Проанализировать существующее программное обеспечение
* Определить язык программирования и среду разработки
* Определить функционал приложения
* Реализовать алгоритм работы и интерфейс приложения на выбранном языке программирования
* Тестирование приложения

# ОСНОВНАЯ ЧАСТЬ

Глава 1. Постано­вка задачи

В рамках курсовой работы требуется разработать кроссплатформенное приложение (Клиент) и приложение, которое занимается хранением и обработкой информации, (Сервер), выполняющее учет и отображение объектов предметной области, хранящихся в базе данных (БД), и предоставляющее справочную и статистическую информацию. Клиент должен быть с графическим интерфейсом.

Для реализации данного приложения был выбран фреймворки Spring и Vaadin на основе Java, которые позволяют создать Web приложение с интерфейсом для большинства браузеров. Был выбран архитектурный паттерн MVC – схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер)

Далее было необходимо определить функционал будущего приложения. После изучения предметной области мной были выявлены следующие важные функции:

* Авторизация пользователей;
* Создание, редактирование и удаление данных;
* Фильтрация задач проекта по колонкам;
* Поиск по данным;
* Суммарные затраты;

## Глава 2. Описание предметной области

Предметная область предполагает использование программного обеспечения для учета денежных средств организации. Программное обеспечение должно предоставлять возможность отслеживания движения денег внутри организации и быть интегрированным с другими системами учета денежных средств. Также программное обеспечение должно поддерживать, контроль доходов и расходов, контроль налогов и так далее. Оно должно быть доступно на нескольких платформах и быть управляемым через интерфейс пользователя.

## Глава 3. Актуальность автоматизации

Автоматизация работы с денежными средствами является необходимым условием для успешной работы организации. При автоматизации документооборота, отслеживания движения денежных средств и проведения анализа финансовых показателей можно ускорить процессы и увеличить объем обрабатываемых данных. Автоматизация дает возможность создавать динамические отчеты, упрощая работу с подсчётом денег. Это очень полезно для организации, поскольку позволяет делать более оперативные и оптимальные решения.

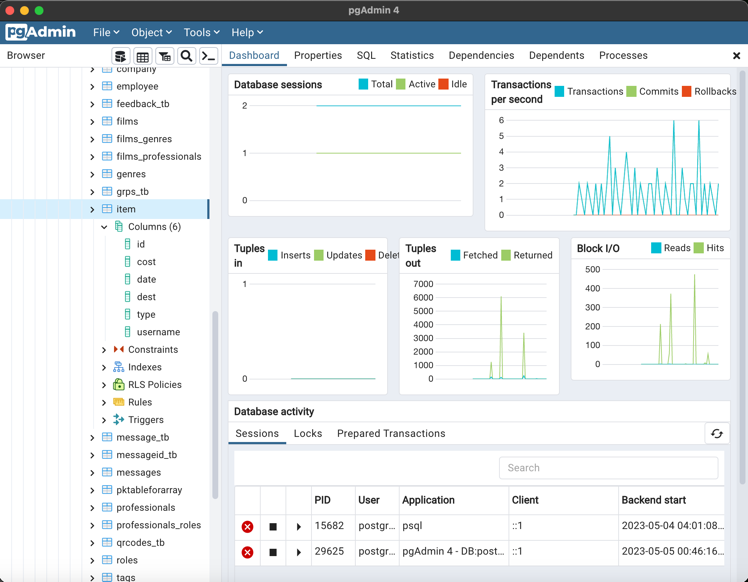
## Глава 4. Описание программы

### 4.1. Алгоритмические решения

Основная задача этого приложения – обработка и хранение данных. Для этих целей было выбрано использование PostgreSQL, в БД записываются данные пользователей, а также данные по записям доходов.

PostgreSQL это открытая объектно-реляционная система управления базами данных (СУБД), которая позволяет приложениям и пользователям хранить и манипулировать большими объемами данных с помощью структурированных таблиц и запросов SQL. Он предоставляет пользователям возможность использовать для своих приложений расширенные возможности и инструменты.

Для просмотра БД можно использовать PgAdmin (рис. 1) или SQL Shell (рис. 2)

 Изображение выглядит как текст

Автоматически созданное описание

Рисунок *1* – Интерфейс PgAdmin *Рисунок 2* – Интерфейс SQL Shell

В моделях были прописаны объекты User (для учёта пользователя) и Item (для учёта записей). По ним, с помощью библиотек lombok и javax.persistence, формируются таблицы user\_tb и Item:

* User (user\_tb) – хранит в себе id, username – имя пользователя (логин) и password – пароль. С помощью таблицы мы можешь хранить данные о пользователях и сохранять записи конкретного пользователя. (рис. 3)
* Item – хранит в себе id записи, type – тип записи, cost – указанная стоимость, date – дата создания записи, dest – назначение платежа или же просто комментарий, username – связь с таблицей user\_tb. (рис. 4)

Изображение выглядит как текст

Автоматически созданное описание Изображение выглядит как текст

Автоматически созданное описание

*Рисунок 3 и 4* – классы объектов User и Item

Благодаря этим моделям нет необходимости создавать таблицы напрямую в PostgreSQL и легче происходит взаимодействие программы с БД.

Для операций с БД используются методы:

* 1. save() – сделать новую запись в таблицу
  2. delete() – удалить запись из таблицы
  3. edit() – изменить запись в таблице

Все обработки на сервере выполняются с помощью Java и фреймворка Spring, шаблон был сгенерирован с помощью Spring Initializr (рис. 5)

Изображение выглядит как текст

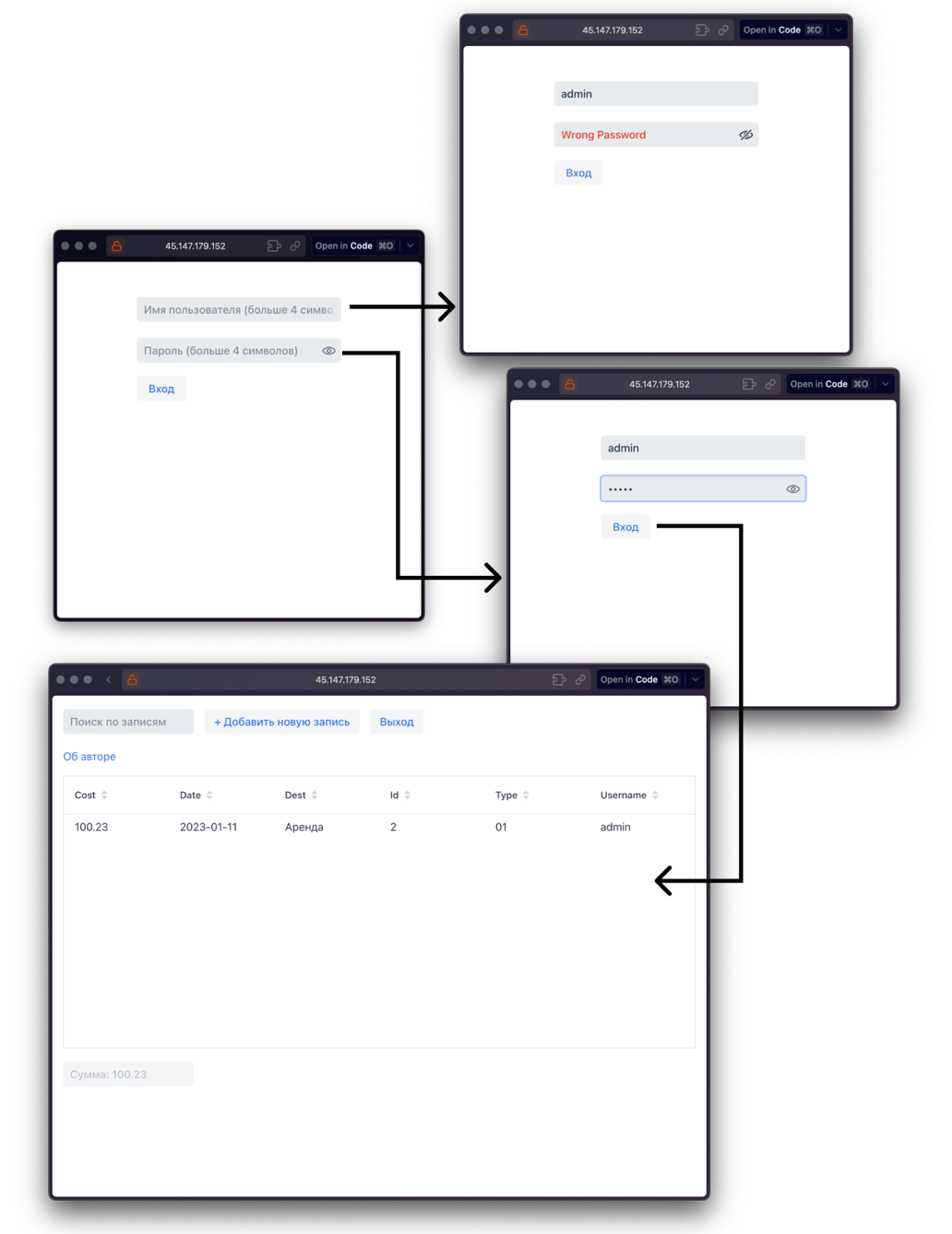
Автоматически созданное описаниеРисунок 5 - <https://start.spring.io/> с зависимостями

Для удобной разработки интерфейса был использован фреймворк Vaadin. Он позволяет быстро описать все необходимые для функционала элементы и вывести их.

Для удобства использования PostgreSQL и последующего развёртывания на сервере был создан инстанс postges версии 14 в docker-compose.yaml (рис. 5)

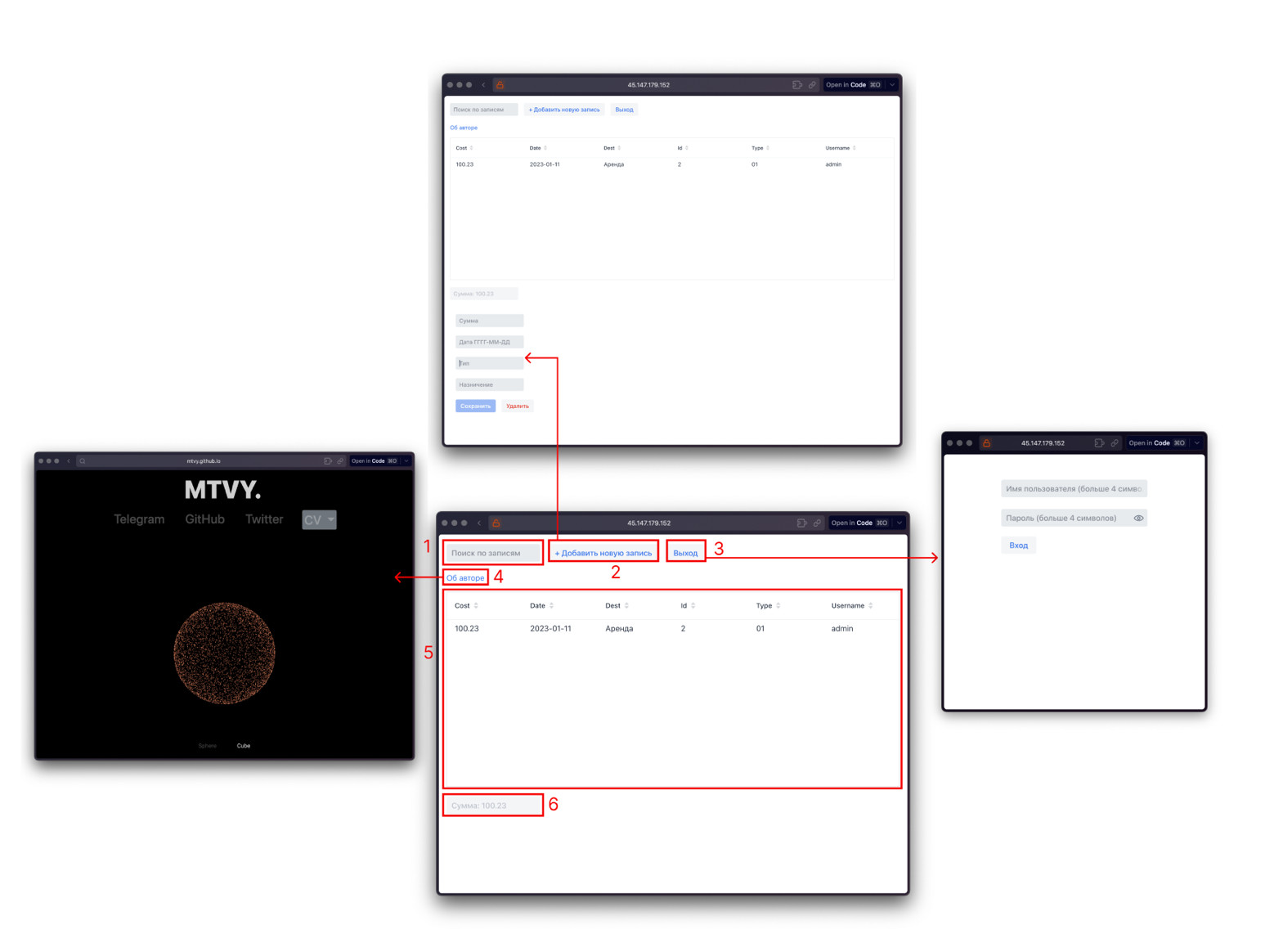
### 4.2. Описание интерфейса программы

При переходе на страницу пользователь попадает на окно входа/регистрации. Далее он проходит авторизацию как новый пользователь, если его данные новый, либо входит в существующий аккаунт. Пароль и имя пользователя проверяются на достоверность и на совпадение с критериями. Пользовательский путь представлен на рисунке 6.



*Рисунок 6* – Интерфейс входа

На странице с записями представлена основная информация. (рис. 7)



*Рисунок 7* - главная страница с записями

Рассмотрим элементы главной страницы:

1. Поиск по записям – поиск по всем параметрам записи на наличие строки, по которой происходит поиск.
2. Добавить новую запись – создаёт окно для ввода данных для новой записи.
3. Выход – вернёт на входную страницу.
4. Об авторе – откроет новую вкладку с сайтом разработчика.
5. Таблица записей.
6. Сумма денег в выведенных записях (сумма меняется в зависимости от 1 (строки поиска).

Рассмотрим сценарии создания, редактирования, удаления записи. (рис. 8, 9)

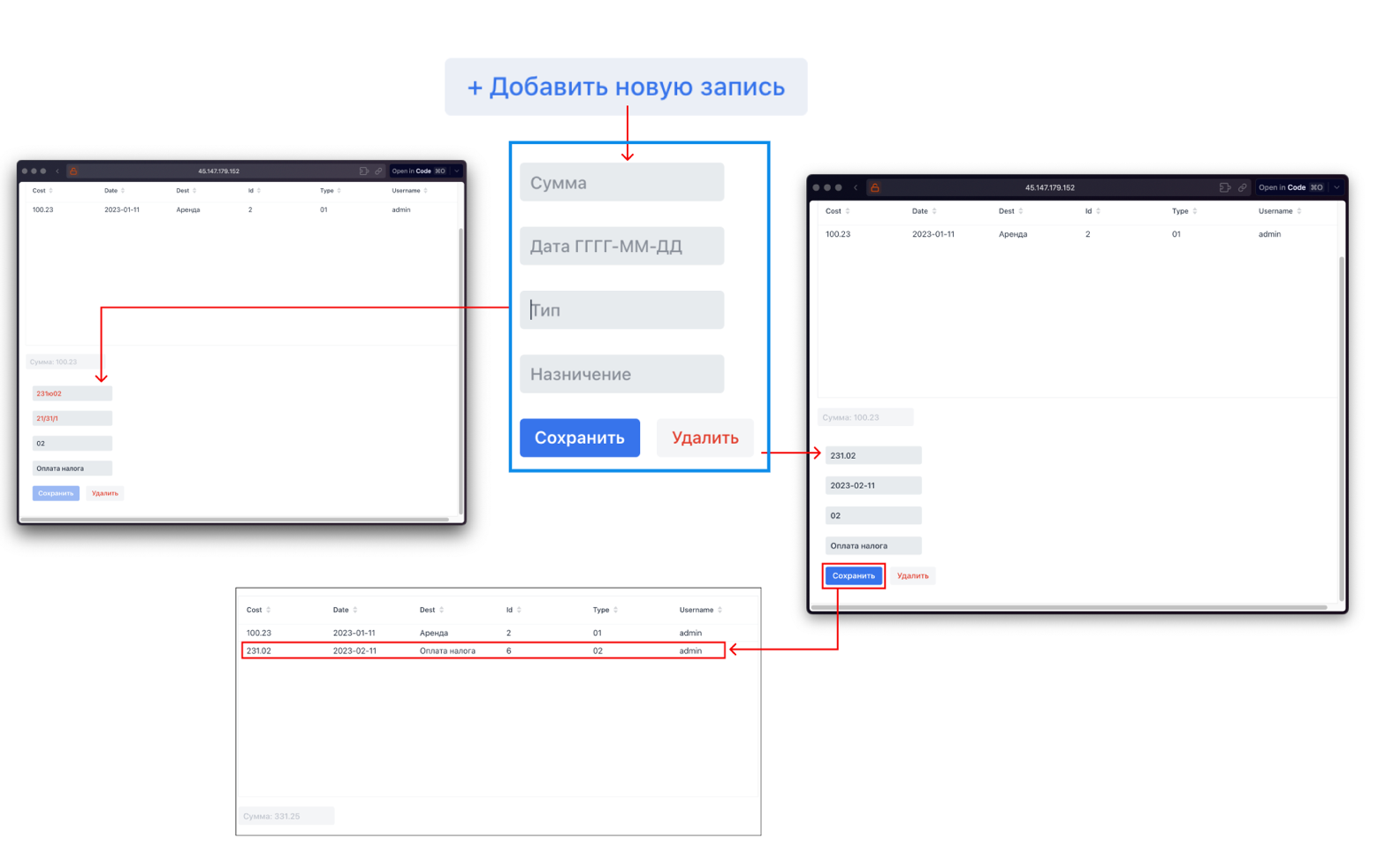
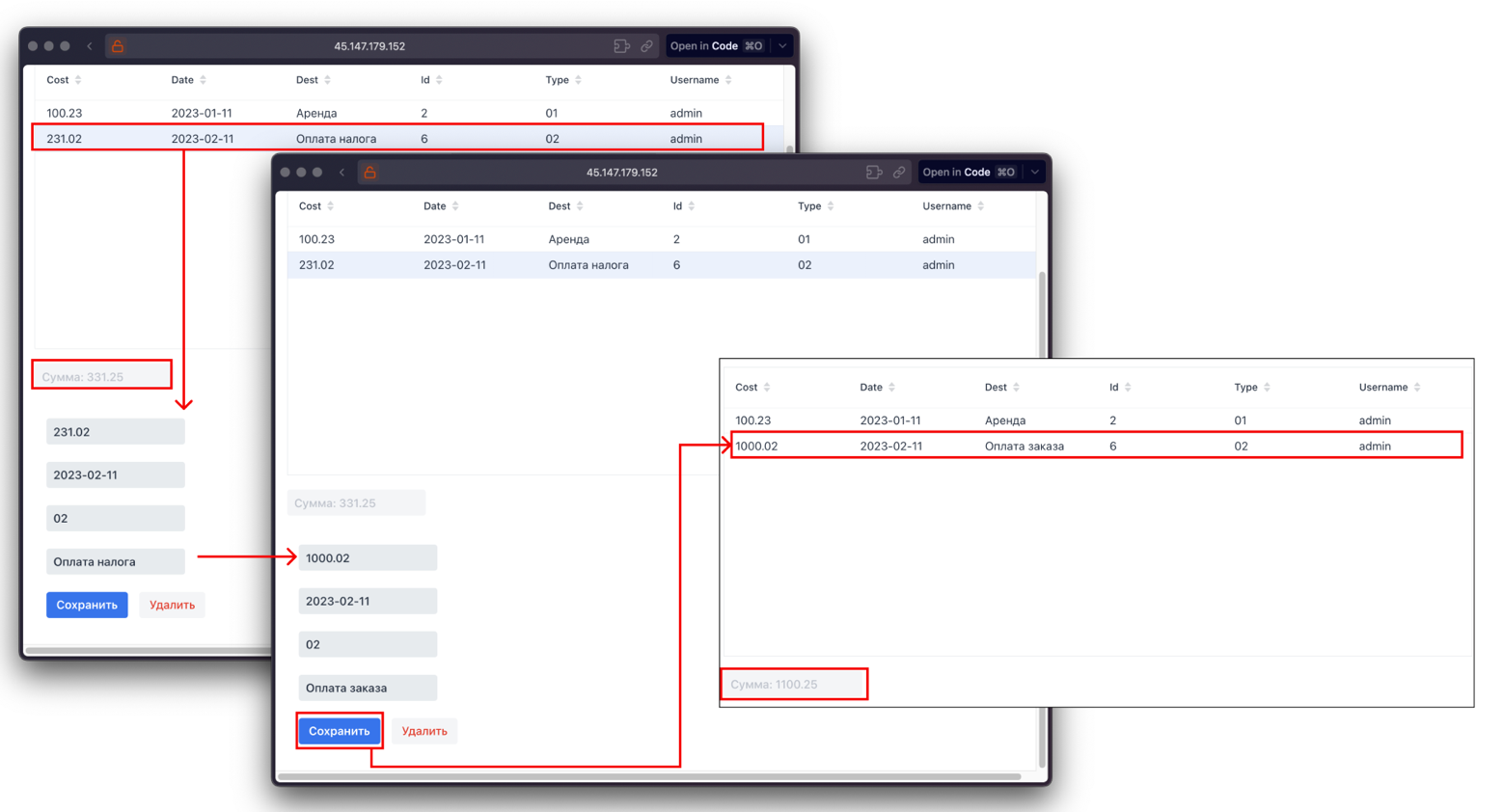


Рисунок 8 – Создание новой записи

* Создание:
  1. По нажатию на кнопку появляется окно для ввода данных.
  2. Если данные не корректны – сохранение не доступно и ошибки подсвечиваются.
  3. Иначе доступна кнопка сохранить, по нажатию которой добавляется новая запись.

Рисунок 9 – Редактирование записи

* Редактирование:
  1. По нажатию на запись в таблице появляется окно для ввода данных с данными их таблицы.
  2. После корректировки доступно сохранение.

Изображение выглядит как текст, снимок экрана, монитор

Автоматически созданное описание

*Рисунок 10* – Удаление записи

* Удаление:
  1. По нажатию на запись в таблице появляется окно для ввода данных с данными их таблицы.
  2. В этом поле доступно удаление.

Изображение выглядит как текст, снимок экрана, в помещении

Автоматически созданное описаниеРисунок *11* – *Поиск и фильтрация*

* *Поиск и фильтрация*:
  1. При вводе строки в поиск, отбираются похожие записи.
  2. По этим записям пересчитывается сумма.
  3. Доступна фильтрация по колонке (убывание/возрастание)

### 4.3. Состав приложения

Рассмотрим файловую структуру приложения

Весь шаблон приложения построенный Spring (рис. 12)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12 - Структура приложения

Для документации был использован JavaDoc и генирация его в intellij IDEA. (рис. 13)

Ссылка на документацию: <https://mtvy.github.io/money_tracker/>

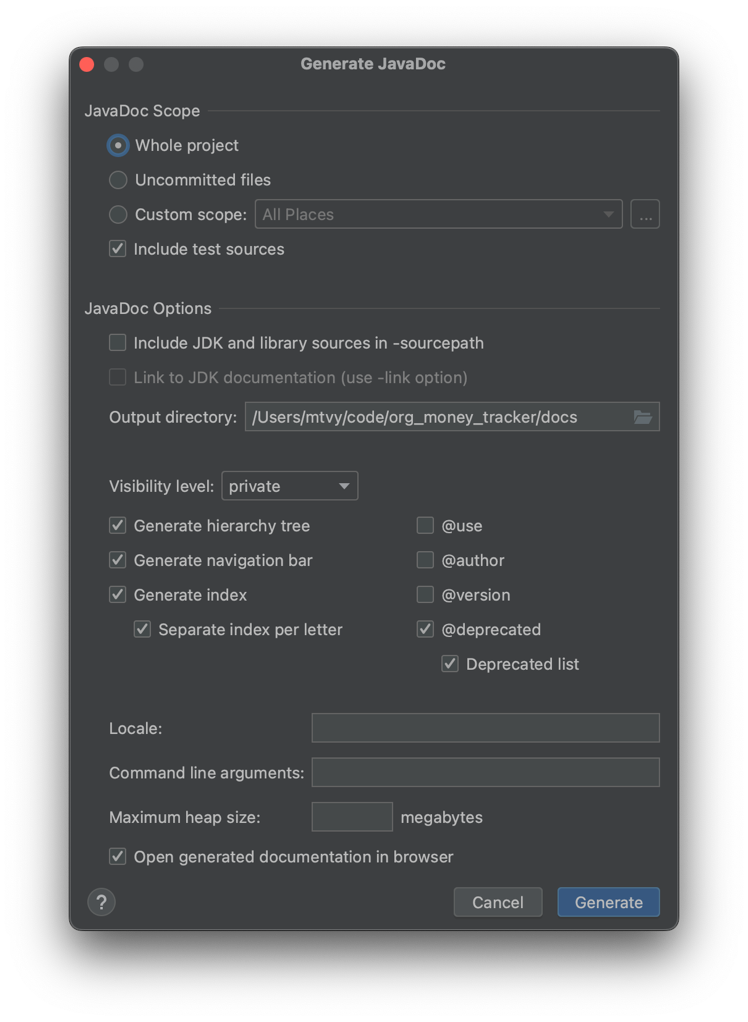


Рисунок 13 – Пример генерации JavaDoc

Пример документации:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

### 4.3. Деплой на сервер

Чтобы была возможность использовать приложении постоянно – было принято решение развернуть программу на сервере.

Для этого собрал проект, затем прописал Dockerfile (рис. 14).

Изображение выглядит как текст

Автоматически созданное описание*Рисунок 14* – Dockerfile с билдом проекта

Затем создал docker-compose.yaml с интстансом postgres и запуском самой программы (рис. 15, 16)

Изображение выглядит как текст

Автоматически созданное описаниеИзображение выглядит как текст

Автоматически созданное описание

Рисунки 15, 16 – docker-compose.yaml

Создал github репозиторий и развернул на сервере Ubuntu, благодаря этому можно с любой платформы использовать приложение по [ссылке](http://45.147.179.152:8081/), также можно посмотреть на код в [репозитории github](https://github.com/mtvy/money_tracker).

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Графическое программное обеспечение

Автоматически созданное описание

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была достигнута поставленная цель - создано программное обеспечение для учета денежных средств организации.

Были исследованы возможные подходы и методологии для разработки приложения, было настроено программное обеспечение соответствующим образом.

Приложение предоставляет пользователю возможность авторизации, реализовано разграничение пользователей и функционала для их работы. Все данные хранятся в PostgreSQL, а функция фильтрации позволяет быстро искать нужные сведения.

Приложение имеет простой, интуитивно понятный интерфейс и поддерживается на большинстве устройств. Приложение может динамически обновляться.

# СПИСОК ЛИТЕРАТУРЫ

1. Официальная документация Spring. – [Электронный ресурс], URL: <https://spring.io/guides> (дата обращения: 01.04.2023).
2. Официальная документация PostgreSQL. – [Электронный ресурс], URL: <https://www.postgresql.org/docs/> (дата обращения: 01.04.2023).
3. Информационный портал. – [Электронный ресурс], URL:<https://stackoverflow.com/> (дата обращения: 01.04.2023).
4. Информационный портал. – [Электронный ресурс],

URL: <https://habr.com/> (дата обращения: 01.04.2023).

# ПРИЛОЖЕНИЕ

* [Документация](https://mtvy.github.io/money_tracker/)
* [Github репозиторий](https://github.com/mtvy/money_tracker)
* [Сайт](http://45.147.179.152:8081/)
* src/main/resources/application.properties

server.port=8081

spring.datasource.url=jdbc:postgresql://localhost:5432/postgres

spring.datasource.username=postgres

spring.datasource.password=postgres

spring.jpa.generate-ddl=true

* src/main/java/money/tracker/MoneyTrackerApplication.java

package money.tracker;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

/\*\* Запуск Spring \*/

@SpringBootApplication

public class MoneyTrackerApplication {

public static void main(String[] args) {

SpringApplication.run(MoneyTrackerApplication.class, args);

}

}

* src/main/java/money/tracker/view/UserView.java

package money.tracker.view;

import com.vaadin.flow.component.button.Button;

import com.vaadin.flow.component.orderedlayout.VerticalLayout;

import com.vaadin.flow.component.textfield.PasswordField;

import com.vaadin.flow.component.textfield.TextField;

import com.vaadin.flow.router.Route;

import com.vaadin.flow.router.RouterLink;

import money.tracker.component.UserEditor;

import money.tracker.entity.User;

import money.tracker.repo.UserRepo;

import org.springframework.beans.factory.annotation.Autowired;

import java.util.List;

/\*\* Отображение главной страницы \*/

@Route("")

public class UserView extends VerticalLayout {

private final UserRepo repo;

Button loginBtn = new Button("Вход");

private final TextField username = new TextField("", "Имя пользователя (больше 4 символов)");

private final PasswordField password = new PasswordField("", "Пароль (больше 4 символов)");

@Autowired

public UserView(UserRepo repo, UserEditor editor) {

this.repo = repo;

setSizeFull();

username.setWidth("300px");

username.setRequired(true);

password.setWidth("300px");

password.setRequired(true);

password.setValue("");

loginBtn.addClickListener(e -> {

if (username.getValue().length() < 4) {

username.getStyle().set("color", "#F52518");

return;

}

if (password.getValue().length() < 4) {

password.getStyle().set("color", "#F52518");

return;

}

List<User> users = repo.findByUsername(username.getValue());

if (users.size() < 1) {

repo.save(new User(username.getValue(), password.getValue()));

} else if (!users.get(0).getPassword().equals(password.getValue())) {

password.getStyle().set("color", "#F52518");

return;

}

loginBtn.getUI().ifPresent(ui -> ui.navigate(

MainView.class, username.getValue() + "p=" + password.getValue()));

});

VerticalLayout fields = new VerticalLayout(username, password, loginBtn);

fields.setSpacing(true);

fields.setMargin(true);

fields.setSizeUndefined();

VerticalLayout uiLayout = new VerticalLayout(fields);

uiLayout.setSizeFull();

uiLayout.setHorizontalComponentAlignment(Alignment.CENTER, fields);

add(uiLayout);

loginBtn.addClickListener(e -> login());

editor.setChangeHandler(() -> editor.setVisible(false));

}

public void login() {

if (username.getValue().equals("")) {

username.getElement().getStyle().set("color", "#F52518");

return;

}

if (password.getValue().equals("")) {

password.getElement().getStyle().set("color", "#F52518");

return;

}

List<User> users = repo.findByUsername(username.getValue());

}

}

* src/main/java/money/tracker/view/MainView.java

package money.tracker.view;

import com.vaadin.flow.component.Text;

import com.vaadin.flow.component.button.Button;

import com.vaadin.flow.component.grid.Grid;

import com.vaadin.flow.component.html.Anchor;

import com.vaadin.flow.component.orderedlayout.HorizontalLayout;

import com.vaadin.flow.component.orderedlayout.VerticalLayout;

import com.vaadin.flow.component.textfield.TextField;

import com.vaadin.flow.data.value.ValueChangeMode;

import com.vaadin.flow.router.BeforeEvent;

import com.vaadin.flow.router.HasUrlParameter;

import com.vaadin.flow.router.Route;

import money.tracker.component.ItemEditor;

import money.tracker.entity.Item;

import money.tracker.entity.User;

import money.tracker.repo.ItemRepo;

import money.tracker.repo.UserRepo;

import org.springframework.beans.factory.annotation.Autowired;

import java.util.List;

/\*\* Отображение главной страницы \*/

@Route("table")

public class MainView extends VerticalLayout implements HasUrlParameter<String> {

private User user;

private final ItemRepo repo;

private final UserRepo urepo;

private final ItemEditor editor;

private final TextField filter = new TextField("", "Поиск по записям");

private final Button addBtn = new Button("+ Добавить новую запись");

private final Anchor aboutAnchor = new Anchor("https://mtvy.github.io/", "Об авторе");

private final Button signOutBtn = new Button("Выход");

private final HorizontalLayout toolbar = new HorizontalLayout(filter, addBtn, signOutBtn);

private final Grid<Item> grid = new Grid<>(Item.class);

private final TextField sumCostTxt = new TextField("", "Суммарный перерасчёт: 0");

@Autowired

public MainView(ItemRepo repo, ItemEditor editor, UserRepo userRepo) {

this.repo = repo;

this.editor = editor;

this.urepo = userRepo;

}

/\*\*

\* Подсчёт суммарных цен в записях

\* @return суммырный cost всех записей

\*/

public double countSum(List<Item> items) {

double summer = 0;

for (Item item : items) {

summer += item.getCost();

}

return summer;

}

/\*\*

\* Вывод записей

\* @param type - если не пустой выберет записи только с этим типом

\*/

public void showItem(String type, String username) {

/\* Если без фильтрации берём все иначе фильтруем \*/

List<Item> items = (type.isEmpty()) ? repo.findByUser(username) : repo.findByType(type, username);

grid.setItems(items);

sumCostTxt.setValue("Сумма: "+countSum(items));

}

@Override

public void setParameter(BeforeEvent event, String userpass) {

System.out.println(userpass);

String[] data = userpass.split("p=");

if (data.length < 2) {

Text txt = new Text("Нет доступа!");

add(txt);

return;

}

this.user = new User(data[0], data[1]);

List<User> users = urepo.findByUsername(this.user.getUsername());

if (users.size() < 1 || !users.get(0).getPassword().equals(this.user.getPassword())) {

Text txt = new Text("Нет доступа!");

add(txt);

return;

}

add(toolbar, aboutAnchor, grid, sumCostTxt, editor);

filter.setValueChangeMode(ValueChangeMode.EAGER);

filter.addValueChangeListener(e -> showItem(e.getValue(), this.user.getUsername()));

grid.asSingleSelect().addValueChangeListener(e -> editor.edit(e.getValue(), this.user));

addBtn.addClickListener(e -> editor.edit(new Item(), this.user));

signOutBtn.addClickListener(e -> {

signOutBtn.getUI().ifPresent(ui -> ui.navigate(UserView.class));

});

editor.setChangeHandler(() -> {

editor.setVisible(false);

showItem(filter.getValue(), this.user.getUsername());

});

aboutAnchor.getElement().setAttribute("target", "\_blank");

sumCostTxt.setEnabled(false);

showItem(filter.getValue(), this.user.getUsername());

}

}

* src/main/java/money/tracker/repo/UserRepo.java

package money.tracker.repo;

import money.tracker.entity.Item;

import money.tracker.entity.User;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

import java.util.List;

/\*\* JPA Репозиторий \*/

public interface UserRepo extends JpaRepository<User, Long> {

/\*\* Поиск записи \*/

@Query("from User u where u.username=:username")

List<User> findByUsername(@Param("username") String username);

}

* src/main/java/money/tracker/repo/ItemRepo.java

package money.tracker.repo;

import money.tracker.entity.Item;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import java.util.List;

/\*\* JPA Репозиторий \*/

public interface ItemRepo extends JpaRepository<Item, Long> {

/\*\* Поиск записи \*/

@Query(

"from Item i " +

"where concat(i.id, ' ', i.type, ' ', i.cost, ' ', i.dest, ' ', i.date) " +

"like concat('%', :type, '%', '%', '%') and i.username=:username"

)

List<Item> findByType(@Param("type") String type, @Param("username") String username);

@Query(

"from Item i where i.username=:username"

)

List<Item> findByUser(@Param("username") String username);

}

* src/main/java/money/tracker/entity/User.java

package money.tracker.entity;

import lombok.Data;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Table;

@Data

@Entity

@Table(name="user\_tb")

public class User {

/\*\* Порядковый номер записи \*/

@Id

@GeneratedValue

private Long id;

/\*\* Тип записи \*/

private String username;

/\*\* Указанная в записи стоимость \*/

private String password;

public User() {}

public User(String username, String password) {

this.username = username;

this.password = password;

}

public Long getId() {

return id;

}

public String getUsername() {

return username;

}

}

* src/main/java/money/tracker/entity/Item.java

package money.tracker.entity;

import lombok.Data;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

/\*\* Домен записи (элемента) в отчёте \*/

@Data

@Entity

public class Item {

/\*\* Порядковый номер записи \*/

@Id

@GeneratedValue

private Long id;

/\*\* Тип записи \*/

private String type;

/\*\* Указанная в записи стоимость \*/

private double cost;

/\*\* Дата записи \*/

private String date;

/\*\* Назначение записи \*/

private String dest;

/\*\* Пользователь \*/

private String username;

}

* src/main/java/money/tracker/component/UserEditor.java

package money.tracker.component;

import com.vaadin.flow.component.Key;

import com.vaadin.flow.component.KeyNotifier;

import com.vaadin.flow.component.button.Button;

import com.vaadin.flow.component.orderedlayout.HorizontalLayout;

import com.vaadin.flow.component.orderedlayout.VerticalLayout;

import com.vaadin.flow.component.textfield.TextField;

import com.vaadin.flow.data.binder.Binder;

import com.vaadin.flow.spring.annotation.SpringComponent;

import com.vaadin.flow.spring.annotation.UIScope;

import lombok.Setter;

import money.tracker.entity.Item;

import money.tracker.entity.User;

import money.tracker.repo.ItemRepo;

import money.tracker.repo.UserRepo;

import org.springframework.beans.factory.annotation.Autowired;

import java.text.SimpleDateFormat;

import com.vaadin.flow.component.Key;

import com.vaadin.flow.component.KeyNotifier;

import com.vaadin.flow.component.button.Button;

import com.vaadin.flow.component.orderedlayout.HorizontalLayout;

import com.vaadin.flow.component.orderedlayout.VerticalLayout;

import com.vaadin.flow.component.textfield.TextField;

import com.vaadin.flow.data.binder.Binder;

import com.vaadin.flow.spring.annotation.SpringComponent;

import com.vaadin.flow.spring.annotation.UIScope;

import money.tracker.entity.Item;

import money.tracker.repo.ItemRepo;

import lombok.Setter;

import org.springframework.beans.factory.annotation.Autowired;

import java.text.SimpleDateFormat;

/\*\* Editor для изменения данных в записях \*/

@SpringComponent

@UIScope

public class UserEditor extends VerticalLayout implements KeyNotifier {

private final UserRepo repo;

private User user;

@Setter

private ChangeHandler changeHandler;

public interface ChangeHandler {

void onChange();

}

@Autowired

public UserEditor(UserRepo repo) {

this.repo = repo;

// Добавляем интервалы

setSpacing(true);

// Делаем форму невидимой

setVisible(false);

}

}

* src/main/java/money/tracker/component/ItemEditor.java

package money.tracker.component;

import com.vaadin.flow.component.Key;

import com.vaadin.flow.component.KeyNotifier;

import com.vaadin.flow.component.button.Button;

import com.vaadin.flow.component.orderedlayout.HorizontalLayout;

import com.vaadin.flow.component.orderedlayout.VerticalLayout;

import com.vaadin.flow.component.textfield.TextField;

import com.vaadin.flow.data.binder.Binder;

import com.vaadin.flow.spring.annotation.SpringComponent;

import com.vaadin.flow.spring.annotation.UIScope;

import money.tracker.entity.Item;

import money.tracker.entity.User;

import money.tracker.repo.ItemRepo;

import lombok.Setter;

import org.springframework.beans.factory.annotation.Autowired;

import java.text.SimpleDateFormat;

/\*\* Editor для изменения данных в записях \*/

@SpringComponent

@UIScope

public class ItemEditor extends VerticalLayout implements KeyNotifier {

private final ItemRepo repo;

private Item item;

private User user;

private final TextField

rowCost = new TextField("", "Сумма"),

date = new TextField("", "Дата ГГГГ-ММ-ДД"),

type = new TextField("", "Тип"),

dest = new TextField("", "Назничение");

private final Button

save = new Button("Сохранить"),

delete = new Button("Удалить");

private final HorizontalLayout buttons = new HorizontalLayout(save, delete);

private final Binder<Item> binder = new Binder<>(Item.class);

@Setter

private ChangeHandler changeHandler;

public interface ChangeHandler {

void onChange();

}

@Autowired

public ItemEditor(ItemRepo repo) {

this.repo = repo;

add(rowCost, date, type, dest, buttons);

binder.bindInstanceFields(this);

// Добавляем интервалы

setSpacing(true);

// Изменяем внешний вид кнопок

save.getElement().getThemeList().add("primary");

delete.getElement().getThemeList().add("error");

// Слушаем события на кнопках

addKeyPressListener(Key.ENTER, e -> save());

// Слушаем поле и проверяем на валидность

rowCost.addValueChangeListener(e -> setValues(rowCost.getValue()));

date.addValueChangeListener(e -> setValues(rowCost.getValue()));

type.addValueChangeListener(e -> setValues(rowCost.getValue()));

dest.addValueChangeListener(e -> setValues(rowCost.getValue()));

save.addClickListener(e -> save());

delete.addClickListener(e -> delete());

// Делаем форму невидимой

setVisible(false);

// Отключаем кнопку, когда пустые поля

save.setEnabled(false);

}

/\*\* Проверка на ввод всех полей \*/

private void updateEnable() {

if (

!rowCost.getValue().equals("") &&

!date.getValue().equals("") &&

!type.getValue().equals("") &&

!dest.getValue().equals("")

) {

save.setEnabled(true);

return;

}

save.setEnabled(false);

}

/\*\*

\* Устанавливаем Cost если валидный

\* @param rowCostValue - значение в ячейке с суммой

\*/

private void setValues(String rowCostValue) {

double result;

boolean isEnable = true;

if (rowCostValue.equals("")) {

return;

}

try {

result = Math.round(Double.parseDouble(rowCostValue) \* 100.0) / 100.0;

item.setCost(result);

rowCost.getStyle().set("color", "#283646");

isEnable = true;

}

catch (Exception e) {

rowCost.getStyle().set("color", "#F52518");

isEnable = false;

}

String dt = date.getValue();

SimpleDateFormat sdf = new SimpleDateFormat("YYYY-MM-DD");

try {

sdf.parse(dt);

date.getStyle().set("color", "#283646");

isEnable = true;

} catch (Exception e) {

date.getStyle().set("color", "#F52518");

isEnable = false;

}

if (isEnable) {

updateEnable();

return;

}

save.getElement().setEnabled(isEnable);

}

/\*\* Функционал сохранения \*/

private void save() {

item.setUsername(user.getUsername());

repo.save(item);

changeHandler.onChange();

}

/\*\* Функционал удаления \*/

private void delete() {

repo.delete(item);

changeHandler.onChange();

}

/\*\*

\* Функционал изменения

\* обработка данных из таблицы или создание новой записи

\* @param it - объект записи

\*/

public void edit(Item it, User u) {

this.user = u;

/\* Если запись пустая то форма невидимая \*/

if (it == null) {

rowCost.setValue("");

rowCost.setPlaceholder("Сумма");

setVisible(false);

return;

}

/\* Обрабатываем записи \*/

if (it.getId() != null) {

this.item = repo.findById(it.getId()).orElse(it);

rowCost.setValue(Double.toString(this.item.getCost()));

} else {

this.item = it;

rowCost.setValue("");

rowCost.setPlaceholder("Сумма");

}

binder.setBean(this.item);

/\* Показываем форму \*/

setVisible(true);

type.focus();

}

}

* docker-compose.yaml

version: "3.8"

services:

postgres:

container\_name: postgres

image: postgres:14.5

command:

- "postgres"

- "-c"

- "max\_connections=50"

- "-c"

- "shared\_buffers=1GB"

- "-c"

- "effective\_cache\_size=4GB"

- "-c"

- "work\_mem=16MB"

- "-c"

- "maintenance\_work\_mem=512MB"

- "-c"

- "random\_page\_cost=1.1"

- "-c"

- "temp\_file\_limit=10GB"

- "-c"

- "log\_min\_duration\_statement=200ms"

- "-c"

- "idle\_in\_transaction\_session\_timeout=10s"

- "-c"

- "lock\_timeout=1s"

- "-c"

- "statement\_timeout=60s"

- "-c"

- "shared\_preload\_libraries=pg\_stat\_statements"

- "-c"

- "pg\_stat\_statements.max=10000"

- "-c"

- "pg\_stat\_statements.track=all"

- -p 5433

environment:

POSTGRES\_DB: "postgres"

POSTGRES\_USER: "postgres"

POSTGRES\_PASSWORD: "postgres"

PGDATA: "/var/lib/postgresql/data/pgdata"

volumes:

- ./database/postgresql.conf:/etc/postgresql.conf

- ./database:/docker-entrypoint-initdb.d

- ./:/var/lib/postgresql/data

ports:

- "5433:5433"

healthcheck:

test: [ "CMD-SHELL", "pg\_isready -U postgres -d postgres" ]

interval: 10s

timeout: 5s

retries: 5

restart: unless-stopped

deploy:

resources:

limits:

cpus: '1'

memory: 4G

networks:

- mainnet

org\_money\_tracker:

container\_name: tracker

build:

context: .

dockerfile: Dockerfile

ports:

- '8081:8081'

environment:

- SPRING\_DATASOURCE\_URL=jdbc:postgresql://postgres:5433/postgres

- SPRING\_DATASOURCE\_USERNAME=postgres

- SPRING\_DATASOURCE\_PASSWORD=postgres

- SERVER\_PORT=8081

depends\_on:

- postgres

networks:

- mainnet

networks:

mainnet:

external: true

* Dockerfile

FROM openjdk:11

VOLUME /tmp

EXPOSE 8081

COPY target/org\_money\_tracker-0.0.1-SNAPSHOT.jar org\_money\_tracker.jar

ENTRYPOINT ["java","-jar","/org\_money\_tracker.jar"]

* Makefile

build:

mvn package -DskipTests

docker network create mainnet

up:

docker-compose up --build -d

down:

docker-compose down

* pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.1.2.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>org\_money\_tracker</groupId>

<artifactId>org\_money\_tracker</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>tracker</name>

<description>Money Tracker</description>

<properties>

<java.version>11</java.version>

<vaadin.version>12.0.4</vaadin.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.vaadin</groupId>

<artifactId>vaadin-spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>javax.validation</groupId>

<artifactId>validation-api</artifactId>

<version>2.0.1.Final</version>

</dependency>

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-ui</artifactId>

<version>1.6.15</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>com.vaadin</groupId>

<artifactId>vaadin-bom</artifactId>

<version>${vaadin.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>