Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

Лабораторная работа №4
«Трекер финансов»

Выполнил:
Студент группы 350501
Козлов М.В.

Проверил:
Скиба И.Г.

МИНСК 2025

1 ПОСТАНОВКА ЗАДАЧИ

Обрабатывать ошибки валидации входных данных для всех endpoint, добавив @ControllerAdvice и кастомные исключения. Добавить логирование действий и ошибок в файл .log (аспекты). Добавить запрос, который формирует и возвращает лог-файл (из общего) для указанной даты. Подключить Swagger и добавить описание для endpoint.

2 СТРУКТУРА ПРОЕКТА

Проект основан на многоуровневой архитектуре, обеспечивающей четкое разделение ответственности между слоями. В нем выделены контроллеры для обработки HTTP-запросов, сервисный слой, содержащий бизнес-логику, репозитории для работы с данными, а также модели данных и мапперы для преобразования сущностей. Структура проекта в IDE Intellij IDEA представлена на рисунке 2.1.

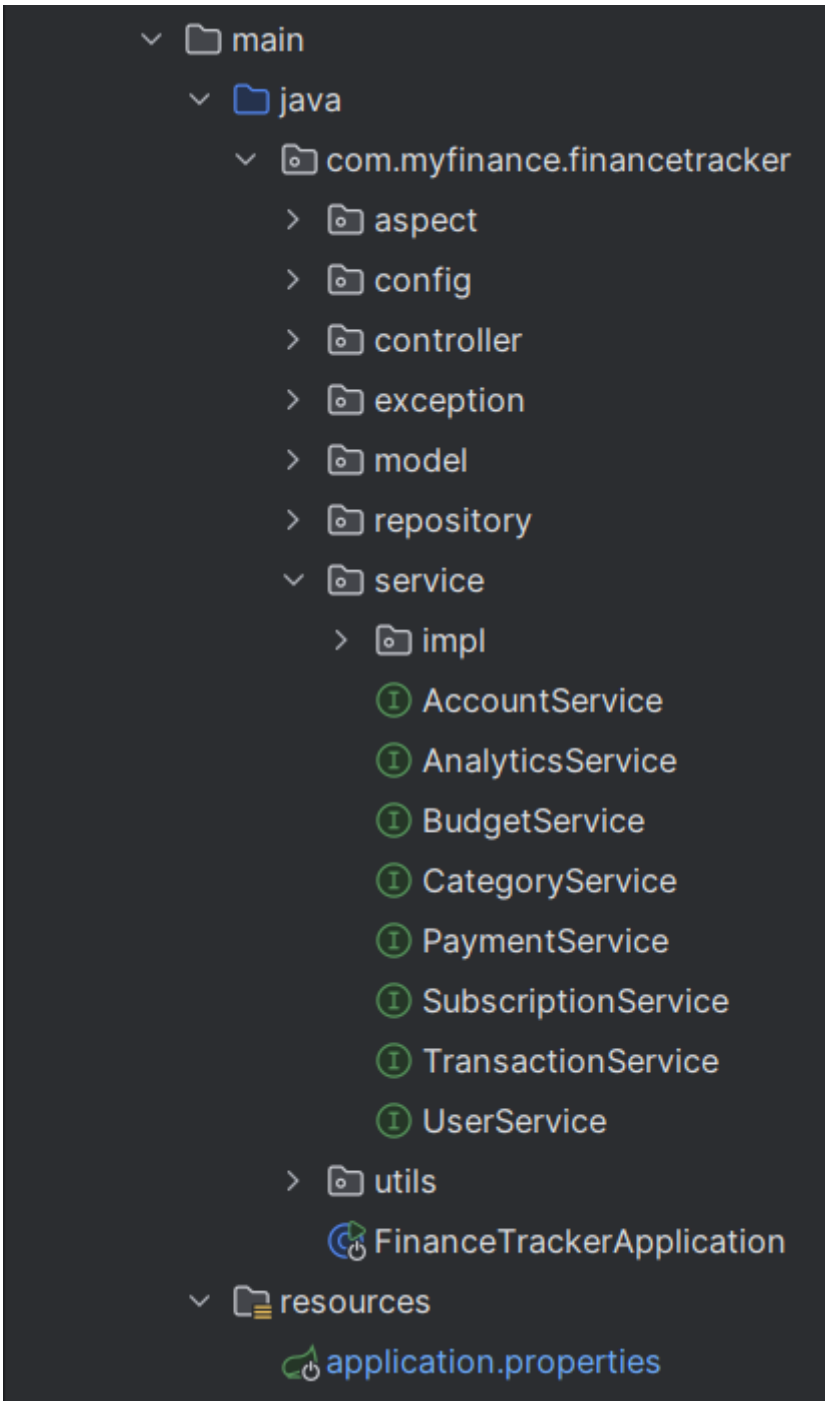

Рисунок 2.1 – Структура проекта в IDE Intellij IDEA

3 ЛИСТИНГ КОДА

Файл `BudgetController.java`

```
package com.myfinance.financetracker.controller;
import
com.myfinance.financetracker.exception.ResourceNotFoundException
;
import com.myfinance.financetracker.model.Budget;
import com.myfinance.financetracker.service.BudgetService;
import io.swagger.v3.oas.annotations.Operation;
import io.swagger.v3.oas.annotations.Parameter;
import io.swagger.v3.oas.annotations.responses.ApiResponse;
import io.swagger.v3.oas.annotations.tags.Tag;
import jakarta.validation.Valid;
import java.util.List;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import
org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import
org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import
org.springframework.web.bind.annotation.RestController;
@RestController
@RequestMapping("/api/budgets")
@Tag(name = "Budget Controller", description = "API для
управления бюджетами")
public class BudgetController {
    private final BudgetService budgetService;
    @Autowired
    public BudgetController(BudgetService budgetService) {
        this.budgetService = budgetService;
    }
    @GetMapping("/{id}")
    @Operation(summary = "Получить бюджет по ID", description
= "Возвращает бюджет по указанному ID")
    @ApiResponse(responseCode = "200", description = "Бюджет
найден")
    @ApiResponse(responseCode = "404", description = "Бюджет
не найден")
    public                              ResponseEntity<Budget>
getBudgetById(@PathVariable Long id) {
        Budget budget = budgetService.getBudgetById(id)
```

```
                .orElseThrow(() -> new
ResourceNotFoundException("Budget not found with id " + id));
            return ResponseEntity.ok(budget);
        }
        @GetMapping
        @Operation(summary = "Получить все бюджеты", description
= "Возвращает список всех бюджетов")
        @ApiResponse(responseCode = "200", description = "Список
бюджетов успешно получен")
        public ResponseEntity<List<Budget>> getAllBudgets() {
            List<Budget> budgets =
budgetService.getAllBudgets();
            return ResponseEntity.ok(budgets);
        }
        @PostMapping("/with-categories")
        @Operation(summary = "Создать бюджет с категориями",
description = "Создает новый бюджет с указанными категориями")
        @ApiResponse(responseCode = "200", description = "Бюджет
найден")
        @ApiResponse(responseCode = "404", description = "Бюджет
не найден")
        public ResponseEntity<Budget>
createBudgetWithCategories(
            @RequestBody Budget budget,
            @Parameter(description = "Список ID категорий",
required = true) @RequestParam List<Long> categoryIds) {
            Budget createdBudget =
budgetService.createOrUpdateBudgetWithCategoryIds(budget,
categoryIds);
            return ResponseEntity.ok(createdBudget);
        }
        @PostMapping
        @Operation(summary = "Создать бюджет", description =
"Создает новый бюджет")
        @ApiResponse(responseCode = "200", description = "Бюджет
найден")
        @ApiResponse(responseCode = "404", description = "Бюджет
не найден")
        public ResponseEntity<Budget> createBudget(@Valid
@RequestBody Budget budget) {
            Budget createdBudget =
budgetService.createOrUpdateBudget(budget);
            return ResponseEntity.ok(createdBudget);
        }
        @PutMapping("/{id}")
        @Operation(summary = "Обновить бюджет", description =
"Обновляет существующий бюджет по ID")
        @ApiResponse(responseCode = "200", description = "Бюджет
найден")
        @ApiResponse(responseCode = "404", description = "Бюджет
не найден")
```

```
        public ResponseEntity<Budget> updateBudget(@PathVariable
Long id, @Valid @RequestBody Budget budgetDetails) {
            Budget budget = budgetService.getBudgetById(id)
                .orElseThrow(()              ->              new
ResourceNotFoundException("Budget not found with id " + id));
            budget.setName(budgetDetails.getName());
          budget.setLimitAmount(budgetDetails.getLimitAmount());
            Budget                updatedBudget                =
budgetService.createOrUpdateBudget(budget);
            return ResponseEntity.ok(updatedBudget);
        }
        @DeleteMapping("/{id}")
        @Operation(summary = "Удалить бюджет", description =
"Удаляет бюджет по ID")
        @ApiResponse(responseCode = "200", description = "Бюджет
найден")
        @ApiResponse(responseCode = "404", description = "Бюджет
не найден")
        public  ResponseEntity<Void>  deleteBudget(@PathVariable
Long id) {
            budgetService.deleteBudget(id);
            return ResponseEntity.noContent().build();
        }
        @GetMapping("/by-limit")
        @Operation(summary = "Получить бюджеты по лимиту",
description = "Возвращает бюджеты с лимитом меньше или равным
указанному")
        @ApiResponse(responseCode = "200", description = "Список
бюджетов успешно получен")
        public                       ResponseEntity<List<Budget>>
getBudgetsByLimitLessThanOrEqual(
            @Parameter(description = "Лимит бюджета", required =
true) @RequestParam Double limit) {
            List<Budget>                budgets                =
budgetService.getBudgetsByLimitLessThanOrEqual(limit);
            return ResponseEntity.ok(budgets);
        }
    }
```

Файл `Budget.java`

```
    package com.myfinance.financetracker.model;
    import com.fasterxml.jackson.annotation.JsonBackReference;
    import
com.fasterxml.jackson.annotation.JsonIgnoreProperties;
    import jakarta.persistence.CascadeType;
    import jakarta.persistence.Column;
    import jakarta.persistence.Entity;
    import jakarta.persistence.FetchType;
    import jakarta.persistence.GeneratedValue;
    import jakarta.persistence.GenerationType;
    import jakarta.persistence.Id;
```

```
import jakarta.persistence.JoinColumn;
import jakarta.persistence.JoinTable;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.OneToMany;
import jakarta.persistence.Table;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.PositiveOrZero;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
@Entity
@Table(name = "budgets")
public class Budget {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NotBlank(message = "Name is required")
    private String name;
    @NotNull(message = "Limit amount is required")
    @PositiveOrZero(message = "Limit amount must be positive
or zero")
    @Column(name = "limit_amount")
    private Double limitAmount;
    private Double spent = 0.0;
    @ManyToMany(
        cascade = {CascadeType.PERSIST, CascadeType.MERGE},
        fetch = FetchType.LAZY)
    @JoinTable(
        name = "budget_category",
        joinColumns = @JoinColumn(name = "budget_id"),
        inverseJoinColumns     =     @JoinColumn(name     =
"category_id"))
    @JsonIgnoreProperties({"hibernateLazyInitializer",
"handler"})
    private List<Category> categories = new ArrayList<>();
    @OneToMany(mappedBy    =    "budget",    cascade    =
CascadeType.ALL, fetch = FetchType.LAZY)
    @JsonBackReference
    private    List<Transaction>    transactions    =    new
ArrayList<>();
    public Budget() {}
    public Budget(String name, Double limitAmount) {
        this.name = name;
        this.limitAmount = limitAmount;
    }
    public Double getRemaining() {
        if (limitAmount == null) {
            return null; // или вернуть 0.0, если это имеет
смысл в вашем контексте
        }
        return limitAmount - spent;
```

```
    }
    public Long getId() {
        return id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Double getLimitAmount() {
        return limitAmount;
    }
    public void setLimitAmount(Double limitAmount) {
        this.limitAmount = limitAmount;
    }
    public Double getSpent() {
        return spent;
    }
    public void setSpent(Double spent) {
        this.spent = spent;
    }
    public List<Category> getCategories() {
        return categories;
    }
    public void setCategories(List<Category> categories) {
        this.categories = categories;
    }
    public List<Transaction> getTransactions() {
        return transactions;
    }
    public      void      setTransactions(List<Transaction>
transactions) {
        this.transactions = transactions;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        Budget budget = (Budget) o;
        return Objects.equals(id, budget.id);
    }
    @Override
    public int hashCode() {
        return Objects.hash(id);
    }
    @Override
    public String toString() {
```

```
            return "Budget{"
                + "id="
                + id
                + ", name='"
                + name
                + '\''
                + ", limitAmount="
                + limitAmount
                + ", spent="
                + spent
                + '}';
        }
    }
```

Файл BudgetRepository.java

```
    package com.myfinance.financetracker.repository;
    import com.myfinance.financetracker.model.Budget;
    import java.util.List;
    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;
    @Repository
    public       interface       BudgetRepository       extends
JpaRepository<Budget, Long> {
        @Query(value = "SELECT * FROM budgets WHERE limit_amount
<= :limit", nativeQuery = true)
        List<Budget>
findBudgetsByLimitLessThanOrEqual(@Param("limit") Double limit);
    }
```

Файл BudgetService.java

```
    package com.myfinance.financetracker.service;
    import com.myfinance.financetracker.model.Budget;
    import java.util.List;
    import java.util.Optional;
    public interface BudgetService {
        Optional<Budget> getBudgetById(Long id);
        List<Budget> getAllBudgets();
        Budget createOrUpdateBudget(Budget budget);
        Budget       createOrUpdateBudgetWithCategoryIds(Budget
budget, List<Long> categoryIds);
        void deleteBudget(Long id);
        List<Budget>     getBudgetsByLimitLessThanOrEqual(Double
limit);
    }
```

Файл BudgetServiceImpl.java

```
    package com.myfinance.financetracker.service.impl;
```

```
import com.myfinance.financetracker.model.Budget;
import com.myfinance.financetracker.model.Category;
import
com.myfinance.financetracker.repository.BudgetRepository;
import
com.myfinance.financetracker.repository.CategoryRepository;
import com.myfinance.financetracker.service.BudgetService;
import java.util.List;
import java.util.Optional;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
@Service
public class BudgetServiceImpl implements BudgetService {
    private final BudgetRepository budgetRepository;
    private final CategoryRepository categoryRepository; //
Добавляем репозиторий категорий
    @Autowired
    public                BudgetServiceImpl(BudgetRepository
budgetRepository,
                                  CategoryRepository
categoryRepository) {
        this.budgetRepository = budgetRepository;
        this.categoryRepository  =  categoryRepository;  //
Инициализируем репозиторий категорий
    }
    @Override
    public Optional<Budget> getBudgetById(Long id) {
        return budgetRepository.findById(id);
    }
    @Override
    public List<Budget> getAllBudgets() {
        return budgetRepository.findAll();
    }
    @Override
    public Budget createOrUpdateBudget(Budget budget) {
        // Присоединяем категории к текущей сессии
        if (budget.getCategories() != null) {
            List<Long>          listCategoriesId          =
budget.getCategories().stream()
                    .map(Category::getId).toList();
            List<Category>        managedCategories        =
categoryRepository.findAllById(listCategoriesId);
            budget.setCategories(managedCategories);
        }
        return budgetRepository.save(budget);
    }
    @Override
    public Budget createOrUpdateBudgetWithCategoryIds(Budget
budget, List<Long> categoryIds) {
        // Загружаем категории по их ID
```

```
            List<Category>              categories              =
categoryRepository.findAllById(categoryIds);
            budget.setCategories(categories);
            return budgetRepository.save(budget);
        }
        @Override
        public void deleteBudget(Long id) {
            budgetRepository.deleteById(id);
        }
        @Override
        public                                          List<Budget>
getBudgetsByLimitLessThanOrEqual(Double limit) {
            return
budgetRepository.findBudgetsByLimitLessThanOrEqual(limit);
        }
    }
```

Файл `TransactionController.java`

```
    package com.myfinance.financetracker.controller;
    import
com.myfinance.financetracker.exception.ResourceNotFoundException
;
    import com.myfinance.financetracker.model.Transaction;
    import
com.myfinance.financetracker.service.TransactionService;
    import io.swagger.v3.oas.annotations.Operation;
    import io.swagger.v3.oas.annotations.Parameter;
    import io.swagger.v3.oas.annotations.responses.ApiResponse;
    import io.swagger.v3.oas.annotations.tags.Tag;
    import jakarta.validation.Valid;
    import java.util.List;
    import
org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.http.ResponseEntity;
    import
org.springframework.web.bind.annotation.DeleteMapping;
    import org.springframework.web.bind.annotation.GetMapping;
    import org.springframework.web.bind.annotation.PathVariable;
    import org.springframework.web.bind.annotation.PostMapping;
    import org.springframework.web.bind.annotation.PutMapping;
    import org.springframework.web.bind.annotation.RequestBody;
    import
org.springframework.web.bind.annotation.RequestMapping;
    import org.springframework.web.bind.annotation.RequestParam;
    import
org.springframework.web.bind.annotation.RestController;
    /**
     * REST-контроллер для управления транзакциями.
     */
    @RestController
    @RequestMapping("/api/transactions")
```

```
    @Tag(name = "Transaction Controller", description = "API для управления транзакциями")
    public class TransactionController {
        private final TransactionService transactionService;
        @Autowired
        public TransactionController(final TransactionService transactionService) {
            this.transactionService = transactionService;
        }
        @GetMapping("/{id}")
        @Operation(summary = "Получить транзакцию по ID", description = "Возвращает транзакцию по указанному ID")
        @ApiResponse(responseCode = "200", description = "Транзакция найдена")
        @ApiResponse(responseCode = "404", description = "Транзакция не найдена")
        public ResponseEntity<Transaction> getTransactionById(@PathVariable final Long id) {
            Transaction transaction = transactionService.getTransactionById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Transaction not found with id " + id));
            return ResponseEntity.ok(transaction);
        }
        @GetMapping
        @Operation(summary = "Получить транзакции по диапазону дат", description = "Возвращает список транзакций в указанном диапазоне дат")
        @ApiResponse(responseCode = "200", description = "Список транзакций успешно получен")
        @ApiResponse(responseCode = "400", description = "Некорректные параметры дат")
        public ResponseEntity<List<Transaction>> getTransactionsByDateRange(
            @Parameter(description = "Начальная дата (формат yyyy-MM-dd)", required = false) @RequestParam(required = false) final String startDate,
            @Parameter(description = "Конечная дата (формат yyyy-MM-dd)", required = false) @RequestParam(required = false) final String endDate) {
            List<Transaction> transactions = transactionService.getTransactionsByDateRange(startDate, endDate);
            return ResponseEntity.ok(transactions);
        }
        @PostMapping
        @Operation(summary = "Создать транзакцию", description = "Создает новую транзакцию")
        @ApiResponse(responseCode = "200", description = "Транзакция успешно создана")
        @ApiResponse(responseCode = "400", description = "Некорректные данные")
```

```
        public                          ResponseEntity<Transaction>
createTransaction(@Valid @RequestBody Transaction transaction) {
            Transaction          createdTransaction          =
transactionService.createOrUpdateTransaction(transaction);
            return ResponseEntity.ok(createdTransaction);
        }
        @PutMapping("/{id}")
        @Operation(summary = "Обновить транзакцию", description
= "Обновляет существующую транзакцию по ID")
        @ApiResponse(responseCode   =   "200",   description   =
"Транзакция успешно обновлена")
        @ApiResponse(responseCode   =   "404",   description   =
"Транзакция не найдена")
        public                          ResponseEntity<Transaction>
updateTransaction(@PathVariable  Long  id,  @Valid  @RequestBody
Transaction transactionDetails) {
            Transaction              transaction              =
transactionService.getTransactionById(id)
                .orElseThrow(()              ->              new
ResourceNotFoundException("Transaction not found with id " + id));
transaction.setAmount(transactionDetails.getAmount());
            transaction.setDate(transactionDetails.getDate());
transaction.setDescription(transactionDetails.getDescription());
            Transaction          updatedTransaction          =
transactionService.createOrUpdateTransaction(transaction);
            return ResponseEntity.ok(updatedTransaction);
        }
        @DeleteMapping("/{id}")
        @Operation(summary = "Удалить транзакцию", description =
"Удаляет транзакцию по ID")
        @ApiResponse(responseCode   =   "204",   description   =
"Транзакция успешно удалена")
        @ApiResponse(responseCode   =   "404",   description   =
"Транзакция не найдена")
        public                                ResponseEntity<Void>
deleteTransaction(@PathVariable Long id) {
            transactionService.deleteTransaction(id);
            return ResponseEntity.noContent().build();
        }
        @GetMapping("/by-user-and-date")
        @Operation(summary = "Получить транзакции по пользователю
и диапазону дат", description = "Возвращает список транзакций для
указанного пользователя в указанном диапазоне дат")
        @ApiResponse(responseCode = "200", description = "Список
транзакций успешно получен")
        @ApiResponse(responseCode   =   "400",   description   =
"Некорректные параметры")
        @ApiResponse(responseCode   =   "404",   description   =
"Пользователь не найден")
        public                 ResponseEntity<List<Transaction>>
getTransactionsByUserAndDateRange(
```

```
            @Parameter(description = "ID пользователя", required
= true) @RequestParam Long userId,
            @Parameter(description = "Начальная дата (формат
yyyy-MM-dd)", required = true) @RequestParam String startDate,
            @Parameter(description = "Конечная дата (формат yyyy-
MM-dd)", required = true) @RequestParam String endDate) {
            List<Transaction>          transactions          =
transactionService.getTransactionsByUserAndDateRange(userId,
startDate, endDate);
            return ResponseEntity.ok(transactions);
        }
    }
```

Файл Transaction.java

```
    package com.myfinance.financetracker.model;
import com.fasterxml.jackson.annotation.JsonBackReference;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.PositiveOrZero;
import java.util.Objects;
@Entity
@Table(name = "transactions")
public class Transaction {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NotNull(message = "Amount is required")
    @PositiveOrZero(message = "Amount must be positive or zero")
    private Double amount;
    @NotBlank(message = "Date is required")
    private String date;
    @NotBlank(message = "Description is required")
    private String description;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "budget_id")
    @JsonIgnoreProperties({"hibernateLazyInitializer",
"handler"}) // Игнорируем прокси
    private Budget budget;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    @JsonBackReference
    private User user; // Связь с пользователем
```

```
    public Transaction() {
    }
    public Transaction(Double amount, String date, String
description, Budget budget, User user) {
        this.amount = amount;
        this.date = date;
        this.description = description;
        this.budget = budget;
        this.user = user;
    }
    // Геттеры и сеттеры
    public Long getId() {
        return id;
    }
    public Double getAmount() {
        return amount;
    }
    public void setAmount(Double amount) {
        this.amount = amount;
    }
    public String getDate() {
        return date;
    }
    public void setDate(String date) {
        this.date = date;
    }
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public Budget getBudget() {
        return budget;
    }
    public void setBudget(Budget budget) {
        this.budget = budget;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (!(o instanceof Transaction)) {
            return false;
        }
```

```
        Transaction that = (Transaction) o;
        return Objects.equals(getId(), that.getId())
            &&
            Objects.equals(getDate(), that.getDate());
    }
    @Override
    public int hashCode() {
        return Objects.hash(getId(), getDate());
    }
    @Override
    public String toString() {
        return "Transaction{"
            +
            "id="
            + id
            +
            ", amount="
            + amount
            +
            ", date='"
            + date
            + '\''
            +
            ", description='"
            + description
            + '\''
            +
            '}';
    }
}
```

Файл TransactionRepository.java

```
package com.myfinance.financetracker.repository;
import com.myfinance.financetracker.model.Transaction;
import java.util.List;
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;
@Repository
public interface TransactionRepository extends
JpaRepository<Transaction, Long> {
    @Query("SELECT t FROM Transaction t WHERE t.user.id
= :userId AND"
        +
        " t.date BETWEEN :startDate AND :endDate")
    List<Transaction> findTransactionsByUserAndDateRange(
        @Param("userId") Long userId,
        @Param("startDate") String startDate,
        @Param("endDate") String endDate
```

```
        );
    }
```

Файл TransactionService.java

```
    package com.myfinance.financetracker.service;
import com.myfinance.financetracker.model.Transaction;
import java.util.List;
import java.util.Optional;
public interface TransactionService {
    Optional<Transaction> getTransactionById(Long id);
    List<Transaction>          getTransactionsByDateRange(String
startDate, String endDate);
    Transaction            createOrUpdateTransaction(Transaction
transaction);
    void deleteTransaction(Long id);
    List<Transaction>     getTransactionsByUserAndDateRange(Long
userId,
                                                          String
startDate, String endDate);
}
```

Файл TransactionServiceImpl.java

```
    package com.myfinance.financetracker.service.impl;

    import com.myfinance.financetracker.model.Budget;
    import com.myfinance.financetracker.model.Transaction;
    import
com.myfinance.financetracker.repository.BudgetRepository;
    import
com.myfinance.financetracker.repository.TransactionRepository;
    import
com.myfinance.financetracker.service.TransactionService;
    import com.myfinance.financetracker.utils.InMemoryCache;
    import
org.springframework.beans.factory.annotation.Autowired;
    import
org.springframework.beans.factory.annotation.Qualifier;
    import org.springframework.stereotype.Service;
    import java.util.List;
    import java.util.Optional;
    import java.util.stream.Collectors;
    @Service
    public     class     TransactionServiceImpl     implements
TransactionService {
        private           final           TransactionRepository
transactionRepository;
        private final BudgetRepository budgetRepository;
        private    final    InMemoryCache<Long,    Transaction>
singleTransactionCache;
        private  final  InMemoryCache<String,  List<Transaction>>
transactionListCache;
```

```
        @Autowired
        public      TransactionServiceImpl(TransactionRepository
transactionRepository,
                                           BudgetRepository
budgetRepository,

@Qualifier("singleTransactionCache")          InMemoryCache<Long,
Transaction> singleTransactionCache,

@Qualifier("transactionListCache")          InMemoryCache<String,
List<Transaction>> transactionListCache) {
            this.transactionRepository = transactionRepository;
            this.budgetRepository = budgetRepository;
            this.singleTransactionCache                          =
singleTransactionCache;
            this.transactionListCache = transactionListCache;
        }
        @Override
        public Optional<Transaction> getTransactionById(Long id)
{
            // Пробуем получить из кэша
            Transaction           cachedTransaction           =
singleTransactionCache.get(id);
            if (cachedTransaction != null) {
                return Optional.of(cachedTransaction);
            }
            // Если нет в кэше - получаем из БД и кэшируем
            Optional<Transaction>         transaction         =
transactionRepository.findById(id);
            transaction.ifPresent(t                             ->
singleTransactionCache.put(t.getId(), t));
            return transaction;
        }
        @Override
        public                                  List<Transaction>
getTransactionsByDateRange(String startDate, String endDate) {
            // Формируем ключ кэша
            String cacheKey = "transactions_" + (startDate !=
null ? startDate : "null") +
                "_" + (endDate != null ? endDate : "null");
            // Пробуем получить из кэша
            List<Transaction>       cachedTransactions       =
transactionListCache.get(cacheKey);
            if (cachedTransactions != null) {
                return cachedTransactions;
            }
            // Если нет в кэше - получаем из БД
            List<Transaction>         allTransactions         =
transactionRepository.findAll();
            // Фильтруем по датам, если они указаны
            List<Transaction>       filteredTransactions       =
allTransactions.stream()
```

```
                .filter(tx -> {
                    String date = tx.getDate();
                    boolean afterStart = startDate == null ||
date.compareTo(startDate) >= 0;
                    boolean beforeEnd = endDate == null ||
date.compareTo(endDate) <= 0;
                    return afterStart && beforeEnd;
                })
                .collect(Collectors.toList());
            // Сохраняем в кэш
            transactionListCache.put(cacheKey,
filteredTransactions);
            return filteredTransactions;
        }
        @Override
        public Transaction createOrUpdateTransaction(Transaction
transaction) {
            boolean isNew = (transaction.getId() == null);
            Transaction savedTransaction =
transactionRepository.save(transaction);
            // Обновляем бюджет, если транзакция новая и
привязана к бюджету
            if (isNew && transaction.getBudget() != null) {
                Budget budget = transaction.getBudget();
                budget.setSpent(budget.getSpent() +
transaction.getAmount());
                budgetRepository.save(budget);
            }
            // Обновляем кэш
            singleTransactionCache.put(savedTransaction.getId(),
savedTransaction);
            // Инвалидируем кэш списков, так как добавили новую
транзакцию
            transactionListCache.clear();
            return savedTransaction;
        }
        @Override
        public void deleteTransaction(Long id) {
            // Удаляем из БД
            transactionRepository.deleteById(id);
            // Удаляем из кэша
            singleTransactionCache.evict(id);
            // Инвалидируем кэш списков
            transactionListCache.clear();
        }
        @Override
        public List<Transaction>
getTransactionsByUserAndDateRange(Long userId, String startDate,
String endDate) {
            // Формируем ключ кэша
            String cacheKey = "user_" + userId + "_" +
                (startDate != null ? startDate : "null") +
```

```
                "_" + (endDate != null ? endDate : "null");
            // Пробуем получить из кэша
            List<Transaction>        cachedTransactions        =
transactionListCache.get(cacheKey);
            if (cachedTransactions != null) {
                return cachedTransactions;
            }
            // Если нет в кэше - получаем из БД
            List<Transaction>           transactions           =
transactionRepository.findTransactionsByUserAndDateRange(userId,
startDate, endDate);
            // Сохраняем в кэш
            transactionListCache.put(cacheKey, transactions)
            return transactions;
        }
    }
```

Файл `CacheConfig.java`

```
    package com.myfinance.financetracker.config;
    import com.myfinance.financetracker.model.Transaction;
    import com.myfinance.financetracker.utils.InMemoryCache;
    import java.util.List;
    import org.springframework.context.annotation.Bean;
    import org.springframework.context.annotation.Configuration;
    @Configuration
    public class CacheConfig {
        @Bean
        public InMemoryCache<Long, Object> accountCache() {
            return new InMemoryCache<>(300_000, 200);
        }
        @Bean("singleTransactionCache")
        public          InMemoryCache<Long,          Transaction>
singleTransactionCache() {
            return new InMemoryCache<>(300_000, 100);
        }
        @Bean("transactionListCache")
        public     InMemoryCache<String,     List<Transaction>>
transactionListCache() {
            return new InMemoryCache<>(300_000, 100);
        }
    }
```

Файл `InMemoryCache.java`

```
package com.myfinance.financetracker.utils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
```

```
import java.util.concurrent.TimeUnit;
@Component
public class InMemoryCache<K, V> {
    private static final Logger logger =
LoggerFactory.getLogger(InMemoryCache.class);
    private static class CacheEntry<V> {
        private final V value;
        private final long expiryTime;
        public CacheEntry(V value, long expiryTime) {
            this.value = value;
            this.expiryTime = expiryTime;
        }
        public V getValue() {
            return value;
        }
        public boolean isExpired() {
            return System.currentTimeMillis() >= expiryTime;
        }
    }
    private final Map<K, CacheEntry<V>> cache = new
ConcurrentHashMap<>();
    private final long ttlMillis;
    private final int maxSize;
    private final ScheduledExecutorService scheduler =
Executors.newSingleThreadScheduledExecutor();
    public InMemoryCache() {
        this(300_000, 100);
    }
    public InMemoryCache(long ttlMillis, int maxSize) {
        this.ttlMillis = ttlMillis;
        this.maxSize = maxSize;
        logger.info("InMemoryCache instance created with TTL {}
milliseconds and maxSize {}", ttlMillis, maxSize);
        scheduler.scheduleAtFixedRate(this::evictExpiredEntries,
ttlMillis, ttlMillis, TimeUnit.MILLISECONDS);
    }
    public V get(K key) {
        CacheEntry<V> entry = cache.get(key);
        if (entry == null) {
            logger.info("Cache miss for key: {}", key);
            return null;
        }
        if (entry.isExpired()) {
            logger.info("Cache entry expired for key: {}", key);
            cache.remove(key);
            return null;
        }
        logger.info("Cache hit for key: {}", key);
        return entry.getValue();
    }
    public void put(K key, V value) {
        // Если кэш достиг максимального размера, очищаем его
```

```
        if (cache.size() >= maxSize) {
            logger.info("Cache maximum size reached. Clearing
cache.");
            clear();
        }
        CacheEntry<V> entry = new CacheEntry<>(value,
System.currentTimeMillis() + ttlMillis);
        cache.put(key, entry);
        logger.info("Cache put for key: {}", key);
    }
    public void evict(K key) {
        cache.remove(key);
        logger.info("Cache evict for key: {}", key);
    }
    public void clear() {
        cache.clear();
        logger.info("Cache cleared");
    }
    private void evictExpiredEntries() {
        for (Map.Entry<K, CacheEntry<V>> entry : cache.entrySet())
{
            if (entry.getValue().isExpired()) {
                cache.remove(entry.getKey());
                logger.info("Cache entry evicted for key: {}",
entry.getKey());
            }
        }
    }
}
```

Файл SwaggerConfig.java

```
 package com.myfinance.financetracker.config;
 import io.swagger.v3.oas.models.OpenAPI;
 import io.swagger.v3.oas.models.info.Info;
 import org.springframework.context.annotation.Bean;
 import org.springframework.context.annotation.Configuration;
 @Configuration
 public class SwaggerConfig {
    @Bean
    public OpenAPI customOpenAPI() {
        return new OpenAPI()
            .info(new Info()
                .title("Finance Tracker API")
                .version("1.0")
                .description("API для управления финансами."));
    }
}
```

Файл GlobalExceptionHandler.java

```
package com.myfinance.financetracker.exception;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.core.annotation.Order;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
import
org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import java.util.HashMap;
import java.util.Map;
@ControllerAdvice
@Order(0)
public class GlobalExceptionHandler {
    private      static      final      Logger      logger      =
LoggerFactory.getLogger(GlobalExceptionHandler.class);
    @ExceptionHandler(MethodArgumentNotValidException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public          ResponseEntity<Map<String,          String>>
handleValidationExceptions(MethodArgumentNotValidException ex) {
        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getAllErrors().forEach(error -> {
            String fieldName = ((FieldError) error).getField();
            String errorMessage = error.getDefaultMessage();
            errors.put(fieldName, errorMessage);
        });
        logger.error("Validation error: {}", errors);
        return            new            ResponseEntity<>(errors,
HttpStatus.BAD_REQUEST);
    }
    @ExceptionHandler(ValidationException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public                                  ResponseEntity<String>
handleValidationException(ValidationException ex) {
        // Логируем ошибку с уровнем ERROR
        logger.error("Validation error: {}", ex.getMessage());
        return       new       ResponseEntity<>(ex.getMessage(),
HttpStatus.BAD_REQUEST);
    }
    @ExceptionHandler(Exception.class)
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    public                                  ResponseEntity<String>
handleGenericException(Exception ex) {
        if        (ex.getMessage()        !=        null        &&
ex.getMessage().contains("org.springdoc")) {
            return new ResponseEntity<>("Swagger error ignored",
HttpStatus.OK);
        }
        logger.error("Internal      server      error:      {}",
ex.getMessage(), ex);
```

```
        return new ResponseEntity<>("An error occurred: " +
ex.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Файл ResourceNotFoundException.java

```
package com.myfinance.financetracker.exception;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;
@ResponseStatus(HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {
    public ResourceNotFoundException(final String message) {
        super(message);
    }
}
```

Файл SwaggerExceptionHandler.java

```
 package com.myfinance.financetracker.exception;
import org.springframework.core.annotation.Order;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.http.HttpStatus;
@ControllerAdvice
@Order(1) // Более высокий приоритет
public class SwaggerExceptionHandler {
    @ExceptionHandler(Exception.class)
    @ResponseStatus(HttpStatus.OK)
    public                                  ResponseEntity<String>
handleSwaggerException(Exception ex) {
        if        (ex.getMessage()        !=        null        &&
ex.getMessage().contains("org.springdoc")) {
            return new ResponseEntity<>("Swagger error ignored",
HttpStatus.OK);
        }
        return null; // Пропускаем обработку, если ошибка не
связана со Swagger
    }
}
```

Файл ValidationException.java

```
 package com.myfinance.financetracker.exception;
public class ValidationException extends RuntimeException {
    public ValidationException(String message) {
        super(message);
    }
}
```

Файл LoggingAspect.java

```
 package com.myfinance.financetracker.aspect;
```

```
    import org.aspectj.lang.JoinPoint;
    import org.aspectj.lang.annotation.AfterReturning;
    import org.aspectj.lang.annotation.AfterThrowing;
    import org.aspectj.lang.annotation.Aspect;
    import org.aspectj.lang.annotation.Before;
    import org.slf4j.Logger;
    import org.slf4j.LoggerFactory;
    import org.springframework.stereotype.Component;
    @Aspect
    @Component
    public class LoggingAspect {
        private final Logger logger =
LoggerFactory.getLogger(this.getClass());
        // Логирование перед выполнением метода
        @Before("execution(*
com.myfinance.financetracker.controller.*.*(..))")
        public void logBefore(JoinPoint joinPoint) {
            if (logger.isInfoEnabled()) { // Проверяем, включен
ли уровень INFO
                logger.info("Executing method: {}",
joinPoint.getSignature().toShortString());
            }
        }
        // Логирование после успешного выполнения метода
        @AfterReturning(pointcut = "execution(*
com.myfinance.financetracker.controller.*.*(..))", returning =
"result")
        public void logAfterReturning(JoinPoint joinPoint, Object
result) {
            if (logger.isInfoEnabled()) { // Проверяем, включен
ли уровень INFO
                logger.info("Method {} executed successfully.
Result: {}", joinPoint.getSignature().toShortString(), result);
            }
        }
        // Логирование ошибок
        @AfterThrowing(pointcut = "execution(*
com.myfinance.financetracker.controller.*.*(..))", throwing =
"error")
        public void logAfterThrowing(JoinPoint joinPoint,
Throwable error) {
            if (logger.isErrorEnabled()) { // Проверяем, включен
ли уровень ERROR
                logger.error("Error in method: {}. Error: {}",
joinPoint.getSignature().toShortString(), error.getMessage());
            }
        }
    }
```

4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

На рисунке 4.1 приведен результат работы программы: при POST-запросе к эндпоинту /api/budgets, с некорректными данными: сервер возвращает 400 Bad Request:

{

"limitAmount": "Limit amount must be positive or zero"

}.

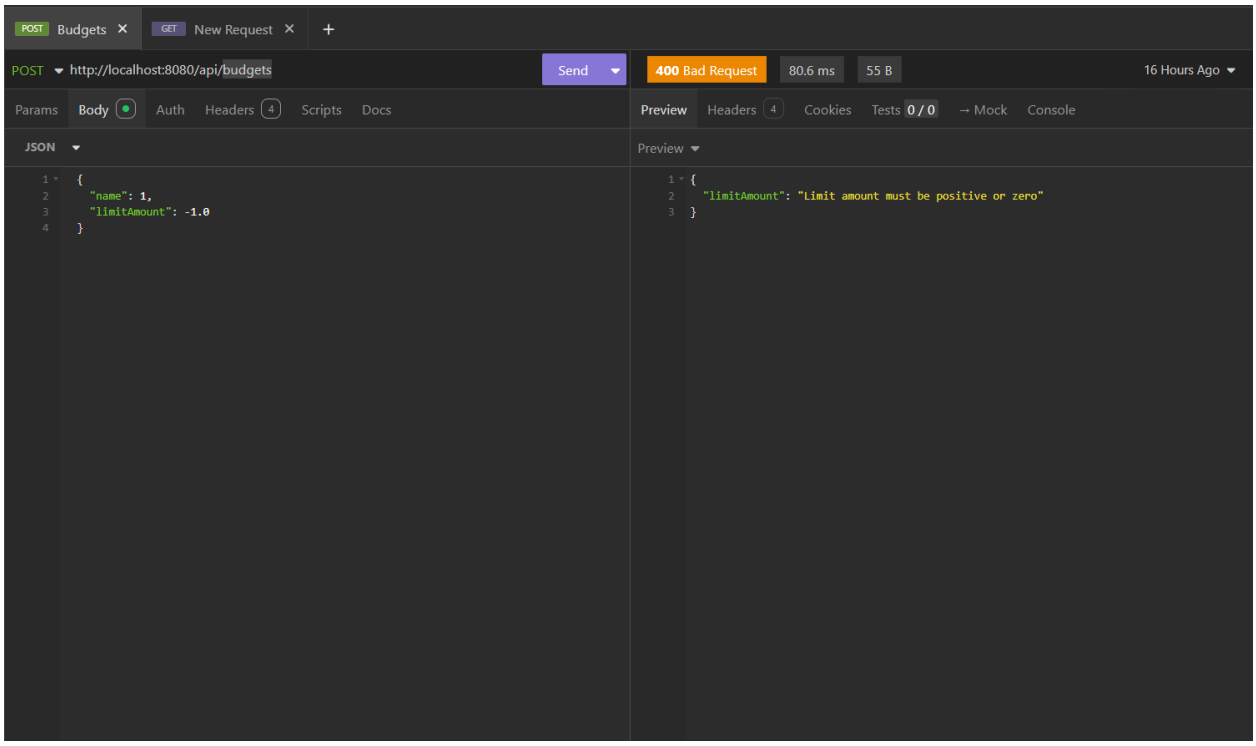

Рисунок 4.1 – Результат работы эндпоинта /api/users

На рисунке 4.2 представлены логи приложенпя, после отправки на сервер некорректных данных

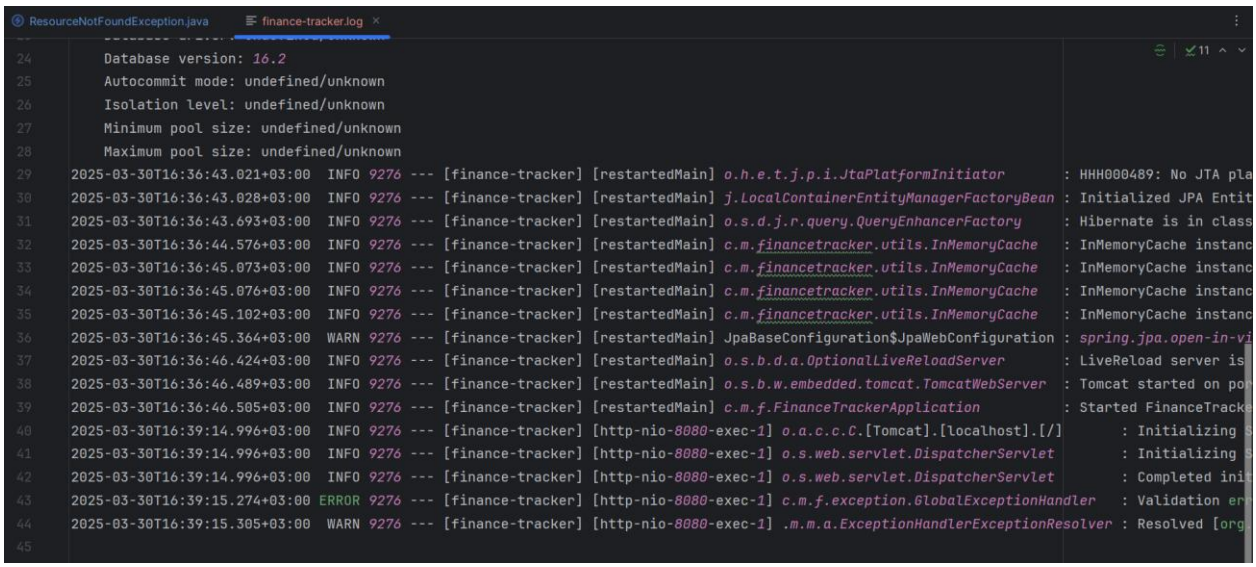

Рисунок 4.2 – Результат работы логирования

На рисунке 4.3 приведен настроенный Swager UI.

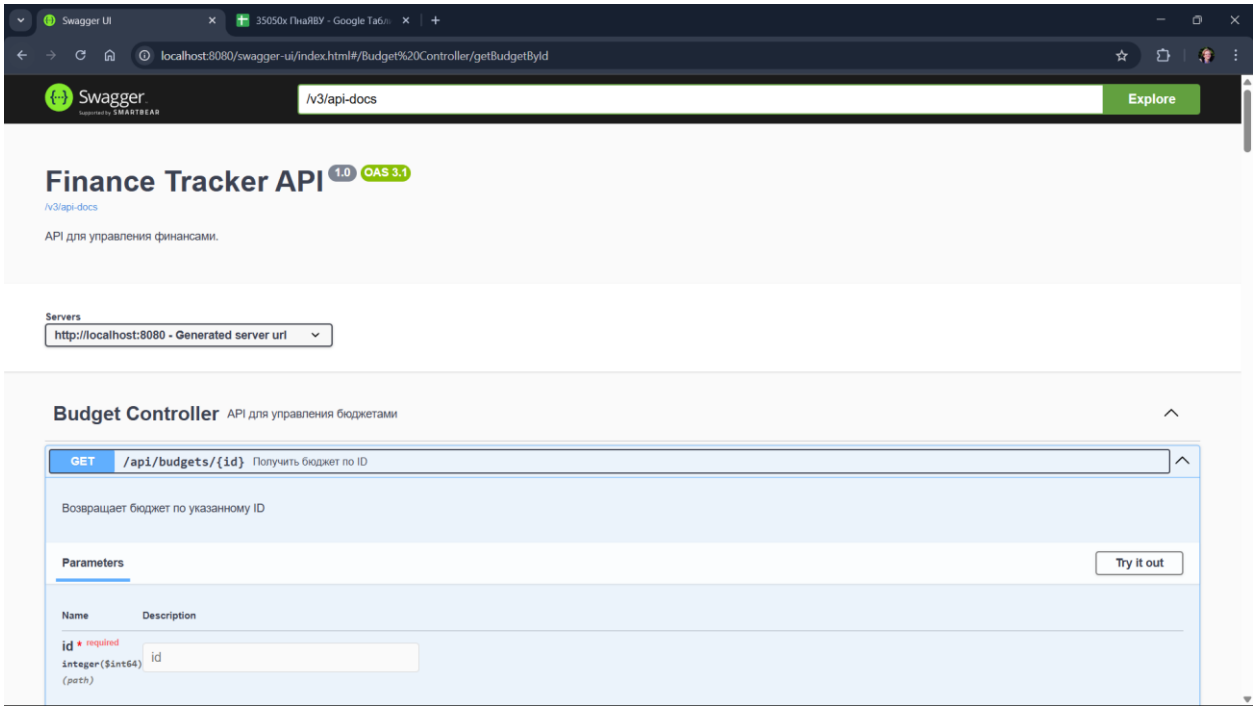

Рисунок 4.3 – SwagerUi

На рисунке 4.4 представлен результат работы эндпоинта /api/logs/download?date=2025-03-30, для скачивания логов за определённое время.

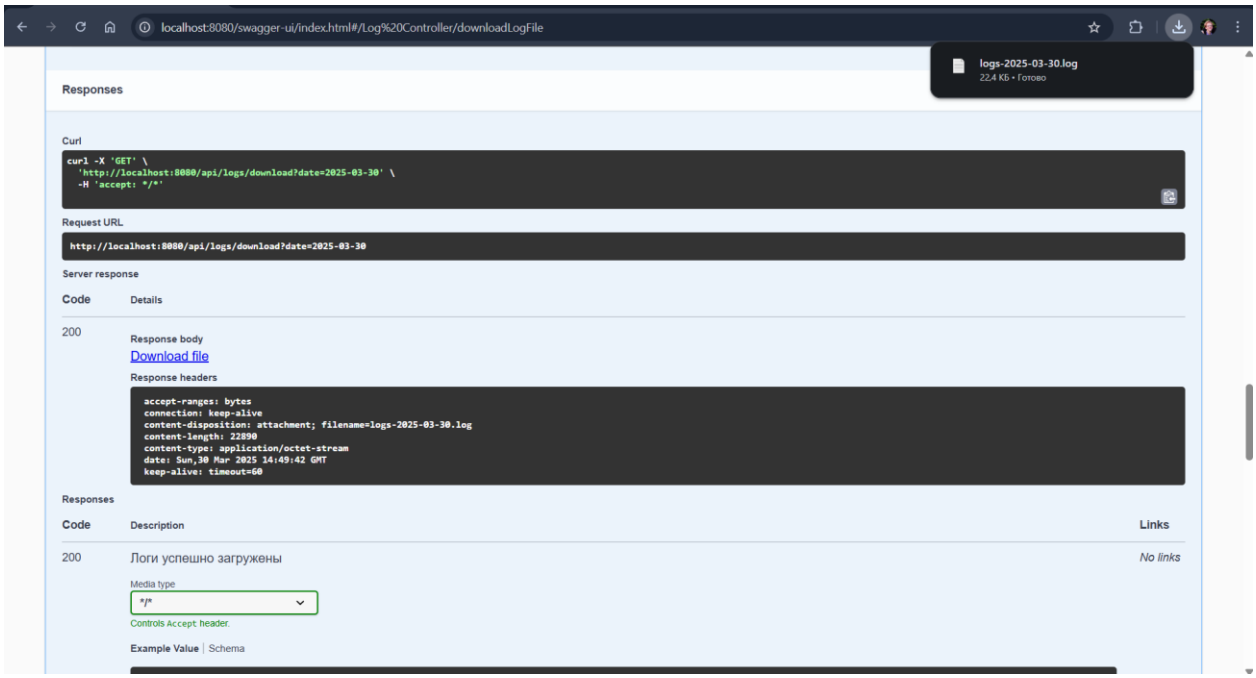

Рисунок 4.4 – Результат обработки запроса с ошибкой 404

5 ЗАКЛЮЧЕНИЕ

В ходе работы был подключен Swager для авто документирования эндпоинтов. Была реализована валидация ошибок пользовательского ввода. Также было добавлено логирования, и возможность скачивания и просмотра логов за определённый день.