

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский экономический университет имени Г.В. Плеханова»

Московский приборостроительный техникум

ОТЧЕТ

по учебной практике

УП.04.01 Внедрение и поддержка программного обеспечения
_____.

Профессионального модуля ПМ.04 Сопровождение и обслуживание
программного обеспечения компьютерных систем _____.

Специальность 09.02.07 Информационные системы и программирование
_____:

Студент Хренов Данила Сергеевич.

(фамилия, имя, отчество)

Группа П50-9-21

Руководитель по практической подготовке от техникума

Образцова Ксения Сергеевна

(фамилия, имя, отчество)

«__» _____ 2024 года

Оглавление

Практическая работа № 1 – Калькулятор и Конвертер валют	3
Практическая работа № 2 – CRUD с локальными данными	11
Практическая работа № 3 – СУБД	20
Практическая работа № 4 – Калькулятор и Конвертер валют	25
Практическая работа № 5 – Авторизация, Регистрация, Разграничение прав доступа.....	30
Практическая работа № 6 – RestFull Api	36
Индивидуальный проект	48

Практическая работа № 1 – Калькулятор и Конвертер валют

Цель работы: Научиться создать контроллеры, работать с GetMapping, PostMapping и RequestParam.

1. Создадим main page.

Создаем контроллер MainController

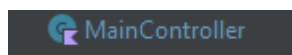


Рисунок 1 – Создание контроллера

В него добавляем метод, который будет возвращать главную страницу по адресу “/”

```
@Controller
class MainController {

    @GetMapping("/")
    fun mainPage(): String{
        return "html/index"
    }
}
```

Рисунок 2 – Возвращение главной страницы

Далее, создаем файл main.html

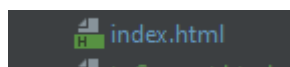


Рисунок 3 –main.html

Далее создаем папку templates

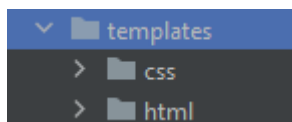


Рисунок 4 – Static

В html файле пишем следующую разметку

```
<body>

<div class="main-div">
  <a class="buttons" th:href="@{/calculator}">Calculator</a>
  <a class="buttons" th:href="@{/forex}">Converter</a>
</div>
</body>
```

Рисунок 5 – Верстка index.html

Результат main.html

Calculator Converter

Рисунок 6 – main.html

2. Обычный калькулятор.

В MainController создаем 2 метода:

• “/calculator” GET запрос возвращает html страницу с калькулятором. И на ней же рассчитывает результат

```
@Controller
class CalculatorController {

    @GetMapping("/calculator")
    fun calculator(
        @RequestParam("first", required = false, defaultValue = "0") first: Double,
        @RequestParam("second", required = false, defaultValue = "0") second: Double,
        @RequestParam("symbol", required = false, defaultValue = "+") symbol: String,
        model: Model
    ): String {

        var result = 0.0

        if (symbol == "-"){
            result = first - second
        }
        else if (symbol == "+"){
            result = first + second
        }
        else if (symbol == "*"){
            result = first * second
        }
        else if (symbol == "/"){
            result = first / second
        }

        model.addAttribute("result", result)

        return "html/calculator"
    }
}
```

Рисунок 7 - calculatorBase

Теперь создаем html страницы под каждый метод.

```

<body>
<form action="/calculator" method="get">
  <input type="number" name="first" placeholder="Первое число" required>
  <select name="symbol">
    <option value="+">+</option>
    <option value="-">-</option>
    <option value="*">*</option>
    <option value="/">/</option>
  </select>
  <input type="number" name="second" placeholder="Второе число" required>
  <input type="submit" formmethod="get" value="Посчитать">
</form>
<p th:text="${result}"></p>
</body>
</html>

```

Рисунок 8 – calculator.html

3. Калькулятор валют.

В MainController создаем еще 2 метода:

- “/forex” GET запрос, возвращает html страницу калькулятора валют.

```

@GetMapping("/forex") new *
fun calculatorForexGet(model: Model): String{
    return "html/convert"
}

```

Рисунок 9 – calculator_currency

- “/calculator_currency” POST запрос, возвращает результат работы калькулятора валют.

From – валюта, которую мы переводим

To – валюта, в которую мы переводим

Count – значение, которое мы переводим из одной валюты в другую.

После проверяем какую валюту мы переводим в какую, производим перевод и возвращаем на html страницу результат перевода.

```

@PostMapping("/forex") new *
fun calculatorForexPost(
    @RequestParam("from") from: String,
    @RequestParam("to") to: String,
    @RequestParam("count") count: Double,
    model: Model
): String{
    val converter = Converter()
    val converted = converter.convert(Forex(
        count = count,
        typeForex = from.lowercase()
    ), to.lowercase())

    model.addAttribute(attributeName: "converted", converted.toString())

    return "html/toConvert"
}

```

Рисунок 10 – ConverterController

```

class Converter {
    val fromTo = mapOf<String, Double>(
        "rub-usd" to 0.011111,
        "usd-rub" to 90.0,
        "rub-eur" to 99.26,
        "eur-rub" to 100.76,
        "eur-usd" to 1.11,
        "usd-eur" to 0.90
    )

    fun convert(from: Forex, to: String): Double{
        if (fromTo.containsKey(from.typeForex + "-" + to)){
            return from.count * fromTo[from.typeForex + "-" + to]!!
        }
        else if (from.typeForex == to){
            return from.count.toDouble()
        }

        return 0.0
    }
}

```

Рисунок 11 – класс конвертера

Верстаем html страницы для методов.

```
<body>
<form action="/forex" method="post">
  <div class="form-example">
    <label for="name">Enter count: </label>
    <input type="text" name="count" id="name" required />
  </div>
  <div class="form-example">
    <label>from:</label>
    <label for="from-select"></label><select name="from" id="from-select">
      <option value="EUR">EUR</option>
      <option value="RUB">RUB</option>
      <option value="USD">USD</option>
    </select>
  </div>
  <div class="form-example">
    <label>to:</label>
    <label for="to-select"></label><select name="to" id="to-select">
      <option value="EUR">EUR</option>
      <option value="RUB">RUB</option>
      <option value="USD">USD</option>
    </select>
  </div>
  <div class="form-example">
    <input type="submit" value="convert!" />
  </div>
</form>
```

Рисунок 12 – converter.html


```

    <label for="name">Enter count: </label>
    <input type="text" name="count" id="name" required />
</div>
<div class="form-example">
    <label>from:</label>
    <label for="from-select"></label><select name="from" id="from-select">

    <option value="EUR">EUR</option>
    <option value="RUB">RUB</option>
    <option value="USD">USD</option>

</select>
</div>
<div class="form-example">
    <label>to:</label>
    <label for="to-select"></label><select name="to" id="to-select">

    <option value="EUR">EUR</option>
    <option value="RUB">RUB</option>
    <option value="USD">USD</option>

</select>
</div>
<div class="form-example">
    <input type="submit" value="convert!" />
</div>

</form>

<div class="form-example">
    <p th:text="${converted}"></p>
</div>
</div>

```

Рисунок 13 – toConverter.html

Результат:

[Calculator Converter](#)

Рисунок 14 – main.page



0.0

Рисунок 15 –калькулятор



126.0

Рисунок 16 – Результат работы калькулятора

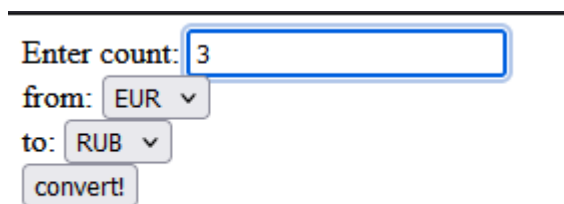
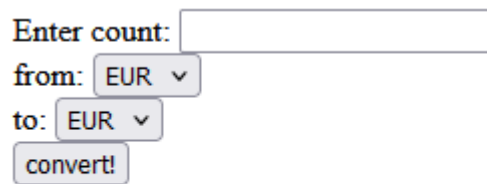


Рисунок 17 – Калькулятор валют



302.280000000000003

Рисунок 18 – Результат работы калькулятора валют

Вывод: Научился создать контроллеры, работать с GetMapping, PostMapping и RequestParam.

Практическая работа № 2 – CRUD с локальными данными

Цель работы: Научиться создать репозитории, сущности и сервисы

Создадим модельки.

Создаем сущности

```
public class StudentModel { 65 usages

    private int id; 3 usages
    private String name; 3 usages
    private String firstName; 3 usages

    @Nullable 3 usages
    private String lastName;

    @Nullable 3 usages
    private String nickname;
```

Рисунок 19 – Создание студента

Создаём сущность менеджера

```
public class ManagerModel { no usages
    private int id; 3 usages
    private String firstName; 3 usages
    private String surName; 3 usages
    private int sell; 3 usages
```

Рисунок 20 – Менеджеры

Далее, создаем сущность оценки

```
public class RankModel { 24 usages

    private int id; 3 usages
    private int rank; 3 usages
    private StudentModel student; 3 usages
    private String obj; 3 usages
    private boolean isDelete = false; 3 usages
```

Рисунок 21 - Оценки

Далее создаём сервисы и имплементацию для них

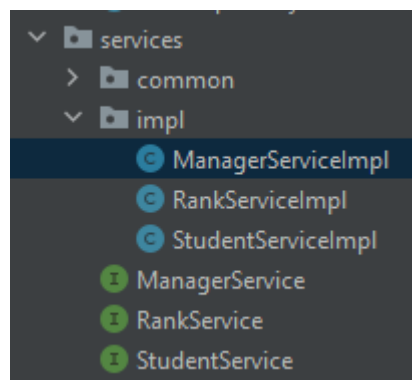


Рисунок 22 – Сервисы

Далее создаём репозитории для наших сервисов

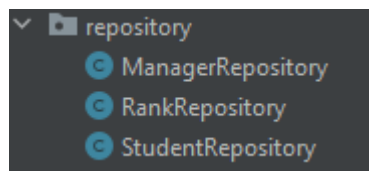


Рисунок 23 – Репозитории

Создаём верстку для студентов

```
<div>
  <div class="studentListContainer" th:each="student : ${students}">
    <p th:text="${student.name}"></p>
    <p th:text="${student.firstName}"></p>
  </div>
</div>

<div>
  <div th:each="page : ${pages}">
    <a th:text="${page}" th:href="@{/testStudents/{page}(page=${page})}"></a>
  </div>
</div>

<form action="/testAddStudent/1" method="post">
  <input placeholder="name" name="name"/>
  <input placeholder="firstName" name="firstName"/>
  <input placeholder="lastName" name="lastName"/>
  <input placeholder="nickname" name="nickname"/>

  <button type="submit">add</button>
</form>
```

Рисунок 24 – studentList

Создаём верстку для оценок

```
<div>
  <div class="studentListContainer" th:each="student : ${students}">
    <a th:href="@{/filterRank/{studentId}(studentId=${student.id})}" th:text="${student.id}"></a>
    <p th:text="${student.name}"></p>
  </div>
</div>

<form action="/testAddRank" method="post">
  <input placeholder="rank" name="rank"/>
  <input placeholder="stud id" name="studId"/>
  <input placeholder="obj" name="obj"/>

  <button type="submit">add</button>
</form>

<form action="/testManyDelete" method="post">
  <input placeholder="studentId" name="studentId"/>

  <button type="submit">many delete</button>
</form>

<form action="/LogicalDeleteRank" method="post">
  <input placeholder="rank id" name="rankId"/>

  <button type="submit">logical delete</button>
</form>
```

Рисунок 25 – rankList

Отфильтрованные оценки

```

<div>
  <div th:each="rank : ${ranks}">
    <p th:text="${rank.rank}"></p>
    <p th:text="${rank.obj}"></p>
  </div>
</div>

```

Рисунок 26 – filteredRanks

Верстка для менеджеров

```

<form action="/manager/add" method="post">
  <input type="text" name="firstname" placeholder="Имя" required>
  <input type="text" name="lastname" placeholder="Фамилия" required>
  <input type="text" name="sell" placeholder="продажи" required>

  <button type="submit" formmethod="post">Добавить менеджера</button>

  <th:block th:each="manager : ${managers}">
    <li>
      <span th:text="${manager.firstName} + ' ' + ${manager.lastName} + ' ' + ${manager.sell}"></span>
      <form action="/manager/update" method="post" style="display:inline;">
        <input th:value="${manager.getID()}" type="hidden" name="id">
        <input th:value="${manager.firstName}" type="text" name="firstName" placeholder="Имя" required>
        <input th:value="${manager.lastName}" type="text" name="lastName" placeholder="Фамилия" required>
        <input th:value="${manager.sell}" type="text" name="sell" placeholder="Продажи" required>

        <button type="submit" formmethod="post">Обновить</button>
      </form>
    </li>
  </th:block>
</form>

```

Рисунок 27 – managerList

Теперь делаем контроллеры для наших страничек и сервисов. Начнём со студентов

```

@GetMapping("/{testStudents}/{page}")
public String testStudents(
    @PathVariable int page,
    Model model
){
    List<StudentModel> students = studentService.findAllStudents();

    Pager<StudentModel> pager = new Pager<>();

    Map<Integer, List<StudentModel>> pages = pager.paginate(students, pageSize: 10);
    model.addAttribute("students", pages.getOrDefault(page, students));

    model.addAttribute("pages", Arrays.stream(pages.keySet().toArray()).sorted());

    return "studentList";
}

@PostMapping("/{testAddStudent}/{page}")
public String testAddStudent(
    @PathVariable int page,
    @RequestParam("name") String name,
    @RequestParam("firstName") String firstName,
    @RequestParam("lastName") String lastName,
    @RequestParam("nickname") String nickname,
    Model model
){
    studentService.addStudent(new StudentModel(id: 0, name, firstName, lastName, nickname));

    List<StudentModel> students = studentService.findAllStudents();

    Pager<StudentModel> pager = new Pager<>();

    Map<Integer, List<StudentModel>> pages = pager.paginate(students, pageSize: 10);
    model.addAttribute("students", pages.getOrDefault(page, students));
}

```

Рисунок 28 – контроллер студентов

Теперь создаём оценки

```

@GetMapping("/testRanks")
public String testRanks(Model model){
    model.addAttribute("students", this.sc.findAllStudents());
    return "rankList";
}

@PostMapping("/testAddRank")
public String testAddStudent(
    @RequestParam("rank") int rank,
    @RequestParam("studId") int studId,
    @RequestParam("obj") String obj,
    Model model
){
    var student = this.sc.findStudentById(studId);
    this.rc.addNewRank(new RankModel(id: 0, rank, student, obj));
    model.addAttribute("students", this.sc.findAllStudents());
    return "rankList";
}

@GetMapping("/filterRank/{studentId}")
public String testFilterRank(
    @PathVariable int studentId,
    Model model
){
    var student = this.sc.findStudentById(studentId);

    model.addAttribute("ranks", this.rc.findRanksByStudent(student));

    return "filteredRanks";
}

```

Рисунок 29 – контроллер оценок

```

@PostMapping("/{LogicalDeleteRank}")
public String logicalDelete(
    @RequestParam("rankId") int rankId,
    Model model
){
    this.rc.deleteRankById(rankId, logical: true);

    model.addAttribute( attributeName: "students", this.sc.findAllStudents());
    return "rankList";
}

@PostMapping("/{deleteRank}")
public String delete(
    @RequestParam("rankId") int rankId,
    Model model
){
    this.rc.deleteRankById(rankId, logical: false);

    model.addAttribute( attributeName: "students", this.sc.findAllStudents());
    return "rankList";
}

@PostMapping("/{testManyDelete}")
public String deleteManyByStudent(
    @RequestParam("studentId") int studentId,
    Model model
){
    this.rc.deleteByStudent(this.sc.findStudentById(studentId));

    model.addAttribute( attributeName: "students", this.sc.findAllStudents());
    return "rankList";
}

```

Рисунок 30 – контроллер оценок

Создаю контроллер менеджеров

```

@GetMapping("/{manager}")
public String getAllStudents(Model model) {
    model.addAttribute( attributeName: "managers", managerService.findAllManagers());
    return "managerList";
}

@PostMapping("/{manager/add}")
public String addClient(@RequestParam String firstname,
    @RequestParam String lastname,
    @RequestParam int sell) {
    ManagerModel newManager = new ManagerModel( id: 0,firstname,lastname,sell);
    managerService.addManager(newManager);
    return "redirect:/manager";
}

```

Рисунок 31 – контроллер менеджеров

Результат:

ИвановИван

ПетровПетр

СидоровСидор

СмирновАлексей

КузнецовДмитрий

ЛебедевАндрей

ПоповСтепан

ВасильевАнтон

ЗайцевСергей

НовиковКонстантин

1

2

name

firstName

lastName

nickname

add

Рисунок 32 – студенты с пагинацией

МорозовНикита

СоловьевВиктор

СеменовЕгор

ПавловФилипп

КовалевМатвей

1

2

name

firstName

lastName

nickname

add

Рисунок 33 - студенты с пагинацией

213

33

false

5

asd

false

Рисунок 34 – оценки студента

1	Иванов
2	Петров
3	Сидоров
4	Смирнов
5	Кузнецов
6	Лебедев
7	Попов
8	Васильев
9	Зайцев
10	Новиков
11	Морозов
12	Соловьев
13	Семенов
14	Павлов
15	Ковалев

rank	stud id	obj	add
2	many delete		

Рисунок 35 – множественное удаление

Рисунок 36 - множественное удаление

Ковалев			
rank	stud id	obj	add
studentId	many delete		
2	logical delete		

Рисунок 37 – логическое удаление

2

ыва

true

Рисунок 38 – логическое удаление

Имя	Фамилия	продажи	Добавить менеджера
фыв ыва 444	фыв	ыва	444
			Обновить

Рисунок 39 – менеджеры

Вывод: научился создавать сервисы, локальные репозитории и сущности для своего приложения.

Практическая работа № 3 – СУБД

Цель работы: Научиться работать с СУБД в spring boot, а так же spring jpa repository

Создаю модель энтити для своей таблицы

```
@Entity
public class StudentModel {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Size(min = 3, max = 50, message = "Длина должна быть от 3 до 50 символов")
    private String name;

    @Size(min = 3, max = 50, message = "Длина должна быть от 3 до 50 символов")
    private String firstName;

    public void setFirstName(String firstName) { this.firstName = firstName; }

    public void setDeleted(boolean deleted) { this.isDeleted = deleted; }

    @Column(name="isDelete", columnDefinition = "boolean default false")
    private boolean isDeleted;

    public StudentModel() { }

}
```

Рисунок 40 – энтити класс

Делаю DTO класс для своей основной таблицы



```

public class StudentDto { 9 usages  ▲ kliker +1

    private String name; 3 usages

    private String firstName; 3 usages

    public StudentDto(String name, String firstName) { 1 usage  ▲ Данила Хренов
        this.name = name;
        this.firstName = firstName;
    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

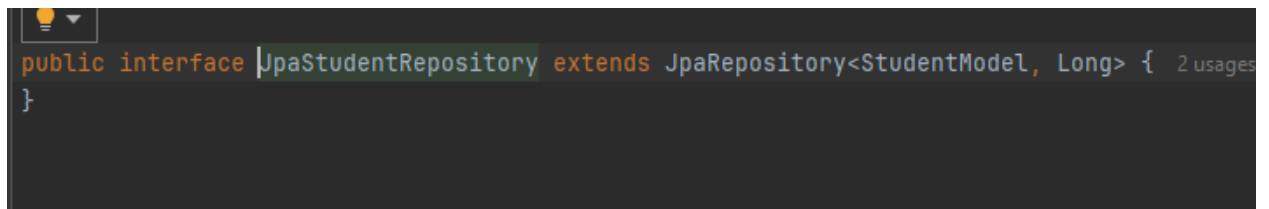
    public String getFirstName() { return firstName; }

    public void setFirstName(String firstName) { this.firstName = firstName; }
}

```

Рисунок 41 – дто класс

Создаю jpa репозиторий для своей таблички



```

public interface JpaStudentRepository extends JpaRepository<StudentModel, Long> { 2 usages
}

```

Рисунок 42 – jpa репозиторий

Создаю дженерик класс своего сервиса

```

public abstract class ModelService<Entity, ModelDTO> { 5 usages 1 inheritor 1 kliker
    protected JpaRepository<Entity, Long> repository; 8 usages

    ModelService( 1 usage 1 kliker
        JpaRepository<Entity, Long> repository
    ){
        this.repository = repository;
    }

    public List<Entity> FindAll(){ 2 usages 1 Данила Хренов
        return repository.findAll();
    }

    @Nullable 3 usages 1 kliker
    public Entity GetById(Long ID)
    {
        return repository.findById(ID).orElse( other: null);
    }

    public boolean Create(ModelDTO modelDTO) { 1 usage 1 kliker
        Entity entity = ConvertDTOToEntity(modelDTO);
        repository.save(entity);
        return true;
    }
}

```

Рисунок 43 – дженерик

Делаю jpa сервис для своей модельки

```

service = данила хренов
public class JpaStudentService extends ModelService<StudentModel, StudentDto>{

    JpaStudentService(JpaStudentRepository repository) { 1 Данила Хренов
        super(repository);
    }

    @Override no usages 1 Данила Хренов
    public boolean Delete(Long ID) {
        StudentModel student = GetById(ID);

        if (student != null){
            if (student.isDeleted()){
                repository.deleteById(ID);
            }
            else{
                student.setDeleted(true);
                repository.save(student);
            }
        }

        return true;
    }

    @Override 1 usage 1 Данила Хренов
    protected StudentModel ConvertDTOToEntity(StudentDto studentDto) {
        return new StudentModel(studentDto.getName(), studentDto.getFirstName());
    }
}

```

Рисунок 44 – jpa сервис

Добавляю в свой контроллер сервисы в свой контроллер

```
@Controller
public class StudentController{

    @Autowired
    JpaStudentService jpaStudentService;

    @Autowired
    ModelService<StudentModel, StudentDto> studentService;
```

Рисунок 45 - контроллер

Проверяю:

localhost:8080/students/1

ssd
dfdf
ssd
dfdf

1

фыв	ыва	add
id	find	

Рисунок 46 - начало

ssddfdf
ssddfdf
фывыва

1

name	firstName	add
id	find	

Рисунок 47 – добавление

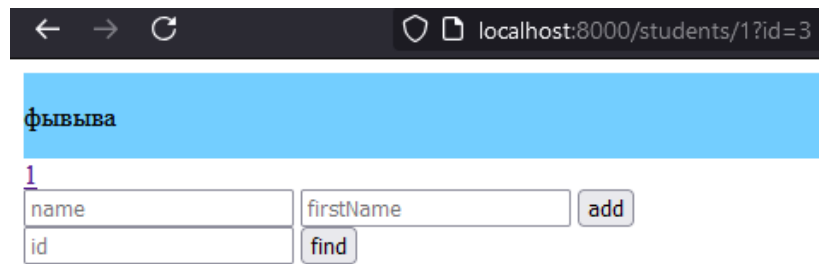


Рисунок 48 - поиск

Вывод: Научился работать с СУБД в spring boot, а так же создавать таблицы и работать с jpa репозиторием.

Практическая работа № 4 – Калькулятор и Конвертер валют

Цель работы: Научиться создавать модели с @OneToOne, @ManyToOne, @OneToMany, @ManyToMany, @JoinTable, @Table.

Изменяю свои модели так, что в них были все нужные мне аннотации

```
@Entity 40 usages Данила Хренов *
@Table(name = "ranks")
public class RankModel {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "rank", nullable = false) 4 usages
    private int rank;

    @ManyToOne 4 usages
    @JoinColumn(name = "student_id")
    private StudentModel student;

    @Column(name="isDelete", columnDefinition = "boolean default false") 3
    private boolean isDelete = false;
```

Рисунок 49 – таблица студентов

```
@Entity 40 usages Данила Хренов *
@Table(name = "ranks")
public class RankModel {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "rank", nullable = false) 4 usages
    private int rank;

    @ManyToOne 4 usages
    @JoinColumn(name = "student_id")
    private StudentModel student;

    @Column(name="isDelete", columnDefinition = "boolean default false") 3 usag
    private boolean isDelete = false;
```

Рисунок 50 – таблица оценок

```

@Entity 26 usages Данила Хренов *
@Table(name = "managers")
public class ManagerModel {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne 4 usages
    @JoinColumn(name = "student_id")
    private StudentModel student;

    @Column(name = "sell", nullable = false) 4 usages
    private int sell;
}

```

Рисунок 51 – таблица менеджеров

Создаю дополнительную модель учебных классов, для создания связи
ManyToMany

```

@Entity 14 usages Данила Хренов +1
@Table(name = "rooms")
public class Classrooms {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Size(min = 1, max = 4) 3 usages
    @Column(name = "room_number")
    private String roomNumber;

    @ManyToMany 3 usages
    @JoinTable(
        name = "student_rooms",
        joinColumns = @JoinColumn(name = "student_id"),
        inverseJoinColumns = @JoinColumn(name = "classroom_id")
    )
    private List<StudentModel> studentsInRoom;
}

```

Рисунок 52 – таблица классных комнат

Создаю контроллер для классрумов

```

@PostMapping("/classroom") new *
public String addNewClassroom(
    @RequestParam("classNumber") String classNumber,
    @RequestParam("students") String students
){
    List<String> studentsId = List.of(students.split(regex: " "));
    List<StudentModel> studentsList = new ArrayList<>();

    for (String id : studentsId){
        studentsList.add(studentService.getByID(Long.parseLong(id)));
    }

    classroomService.Create(new ClassroomDto(classNumber, studentsList));

    return "redirect:/classroom";
}

@GetMapping("/classroom/{classroomId}") new *
public String getAllClassrooms(
    @PathVariable Long classroomId,
    Model model
){
    model.addAttribute("classroom_var", classroomService.getByID(classroomId));

    return "studentsInClassroom";
}

```

Рисунок 53 – контроллер классов

Создаю темплейты

```

</head>
<body>
<p th:text="${classroom_var.getRoom_number()}"></p>
<div class="classroom-container" th:each="student : ${classroom_var.getStudentsInRoom()}">
<p th:text="${student.name}"></p>
</div>

```

Рисунок 54 – студенты в классе

```

</head>
<body>
<div>
<div th:each="classroom : ${classrooms}">
<a th:href="@{/classroom/{classRoom}(classRoom=${classroom.getId()})}">
<p th:text="${classroom.getRoom_number()}"></p>
</div>

<form action="/classroom" method="post">
<input placeholder="номер класса" name="classNumber"/>
<input placeholder="id студентов через запятую" name="students"/>
<button type="submit">add</button>
</form>
</div>

```

Рисунок 55 - классы

Объединяем модели по принципу помещения нужных объектов в поле экземпляра вызываемого объекта, так что бы мы могли воспользоваться им, как бы вызывая метод join и решая проблему request+1

Результат:

1
123
2
123
3
233
4
222

номер класса	id студентов через запят	add
--------------	--------------------------	-----

Рисунок 56 - классы

123

ssd
ssd
фыыв

Рисунок 57 – студенты в классе

ssddfdf
ssddfdf
фыыва
qweert
23ывыв

1

name	firstName	add
id	find	

Рисунок 58 – студенты

Вывод: Научился создавать модели с @OneToOne, @ManyToOne, @OneToMany, @ManyToMany, @JoinTable, @Table.

Практическая работа № 5 – Авторизация, Регистрация, Разграничение прав доступа

Цель: Реализовать авторизацию пользователя посредством заполнения форм авторизации регистрации

Создаём папку конфиг и перечисленные ниже в ней файлы

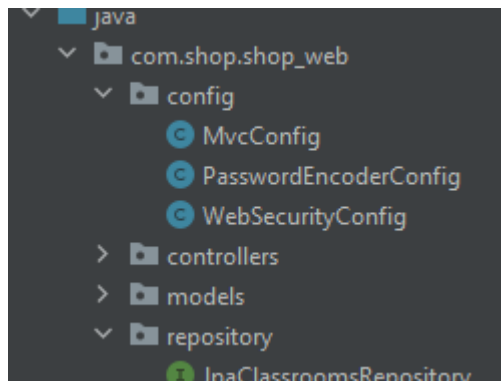


Рисунок 51 - config

Создаём базовый контроллер логина

```
package com.shop.shop_web.config;

import ...

@Configuration no usages
public class MvcConfig implements WebMvcConfigurer {

    @
    public void addViewControllers(ViewControllerRegistry registry) { no usages
        registry.addViewController( urlPathOrPattern: "/login").setViewName("login");
    }

}
```

Рисунок 52 – MvcConfig

Создаём хэширование пароля для нашей моделькой

```

package com.shop.shop_web.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
public class PasswordEncoderConfig {

    @Bean
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(8); }

}

```

Рисунок 53 – PasswordEncoderConfig

Первый метод проверяет наличие переданного пользователя в бд и, если такого нету, то добавляет его. А второй метод разграничивает наши эндпоинты на доступные до и после авторизации, а так же отключает cors и csrf токен

```

}

@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(username -> {
        UserModel user = userRepository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException("Такой пользователь не существует!");
        }
        return new org.springframework.security.core.userdetails.User(
            user.getUsername(),
            user.getPassword(),
            enabled: true,
            accountNonExpired: true,
            credentialsNonExpired: true,
            accountNonLocked: true,
            user.getRole()
        );
    })
    .passwordEncoder(passwordEncoder);
}

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http.authorizeHttpRequests(authorize -> {
        authorize.requestMatchers("/login", "/registration").permitAll()
        .anyRequest().authenticated()
    })
    .formLogin(form -> form.loginPage("/login").defaultSuccessUrl("/").permitAll())
    .logout(logout -> logout.permitAll())
    .csrf(csrf -> csrf.disable())
    .cors(cors -> cors.disable());

    return http.build();
}

```

Рисунок 54 – WebSecurityConfig

Создаём registrationController

```

@Controller
public class RegistrationController {

    @Autowired
    private JpaUserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @GetMapping("/registration")
    public String registrationView() { return "regis"; }

    @PostMapping("/registration")
    public String registrationUser(
        UserModel user,
        Model model,
        HttpServletRequest request
    ){
        if (userRepository.existsByUsername(user.getUsername())) {
            model.addAttribute("message", "Пользователь уже существует");
            return "regis";
        }
        user.setRole(Collections.singleton(RoleEnum.USER));
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        userRepository.save(user);
        return "redirect:/login";
    }
}

```

Рисунок 55 – RegistrationController

Создаём JpaUserRepository

```

public interface JpaUserRepository extends CrudRepository<UserModel, Long> {
    UserModel findByUsername(String username);
    boolean existsByUsername(String username);
}

```

Рисунок 56 – JpaUserRepository

Создаём RoleEnum

```

public enum RoleEnum implements GrantedAuthority {
    USER, ADMIN, MANAGER;

    @Override
    public String getAuthority() { return name(); }
}

```

Рисунок 57 – RoleEnum

Создаём шаблоны для наших форм


```

</head>
<body>
<form method="POST" action="/login">
  <h2>Вход в систему</h2>
  <div>
    <input name="username" type="text" placeholder="Username"
      autofocus="true"/>
    <input name="password" type="password" placeholder="Password"/>
    <button type="submit">Log In</button>
  </div>
</form>
<a th:href="@{/registration}">регистрация</a>
</body>

```

Рисунок 58 – login

```

<title>title</title>
</head>
<body>
<form method="POST" th:action="@{/registration}">
  <h2>Регистрация</h2>
  <div>
    <input name="username" type="text" placeholder="Username"
      autofocus="true"/>
    <input name="password" type="password" placeholder="Passw
    <button type="submit">Registration</button>
  </div>
</form>
<p th:text="${message}"></p>
</body>

```

Рисунок 59 – regis

Расставляю пользователей для эндпоинтов

```

@Controller  Данила Хренов
@RequestMapping("/classroom")
@PreAuthorize("hasAnyAuthority('USER', 'ADMIN')")
public class ClassroomsController {

```

```

@GetMapping("/calculator")
@PreAuthorize("hasAnyAuthority('MANAGER', 'ADMIN')")

@Controller
@PreAuthorize("hasAuthority('ADMIN')")

```

Проверка:

Вход в систему

[регистрация](#)

Рисунок 60 – Авторизация

Регистрация

Рисунок 61 – Регистрация

Вход в систему

[регистрация](#)

Рисунок 59 – вход под админом

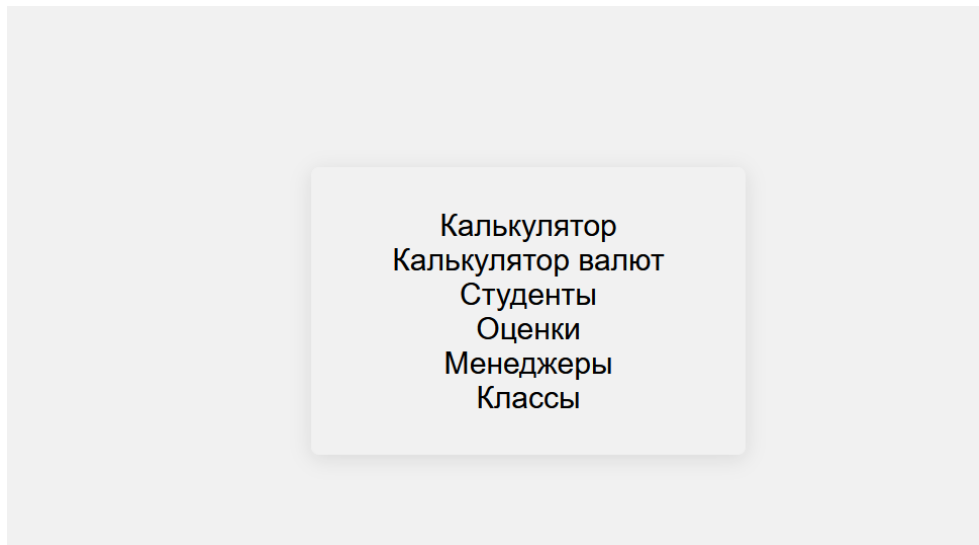


Рисунок 60 – главная под админом

A screenshot of a web application interface. It features a light blue background. On the left side, there is a vertical list of text: 'ssddfdf', 'ssddfdf', 'фыывыва', 'qweert', and '23ыывыв'. Below this list, there is a small red '1' followed by a table with two rows and two columns. The first row contains 'name' and 'firstName' with an 'add' button to the right. The second row contains 'id' and 'find' with a 'find' button to the right.

Рисунок 63 - Успешная авторизация

Вывод: Реализовал авторизацию пользователя посредством заполнения форм авторизации регистрации

Практическая работа № 6 – RestFull Api

Цель: Реализовать рест апи на спринг, создать бэкэнд и фронтэнд

Сначала создам бэкэнд часть приложения

Создаю модели своих таблиц

```
@Entity
public class CoffeeShop {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long shopId;

    4 usages
    @Column(nullable = false)
    private String name;

    4 usages
    private String location;
    4 usages
    private String phone;
    4 usages
    private String email;
```

Рисунок 61 – пример модели

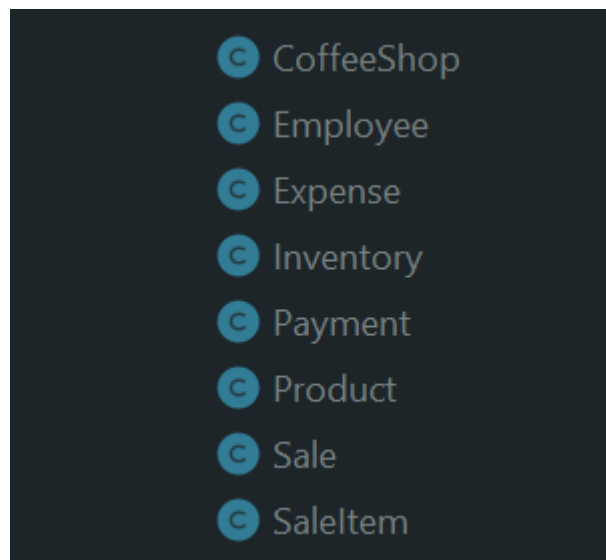


Рисунок 62 – модели бэка

Для каждой модели создаю Jpa репозиторий для реализации crud операций

```

3 usages  Данила Хренов
public interface CoffeeShopRepository extends JpaRepository<CoffeeShop, Long> {
}

```

Рисунок 63 – пример репозитория

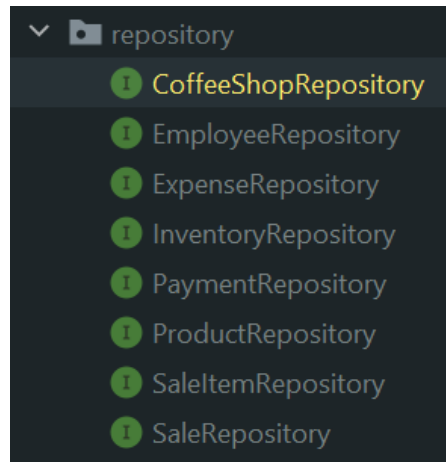


Рисунок 64 - репозитории

Далее создаю сервисы для каждого репозитория, которые пользуются методами этих репозиторияев

```

5 usages
private final CoffeeShopRepository coffeeShopRepository;

Данила Хренов
@Autowired
public CoffeeShopService(CoffeeShopRepository coffeeShopRepository) {
    this.coffeeShopRepository = coffeeShopRepository;
}

Данила Хренов
public List<CoffeeShop> getAll() { return coffeeShopRepository.findAll(); }

Данила Хренов
public CoffeeShop getById(Long id) {
    return coffeeShopRepository.findById(id).orElseThrow(() -> new RuntimeException("Shop not found"));
}

Данила Хренов
public CoffeeShop save(CoffeeShop shop) { return coffeeShopRepository.save(shop); }

Данила Хренов
public void delete(Long id) { coffeeShopRepository.deleteById(id); }

```

Рисунок 65 – сервис магазинов

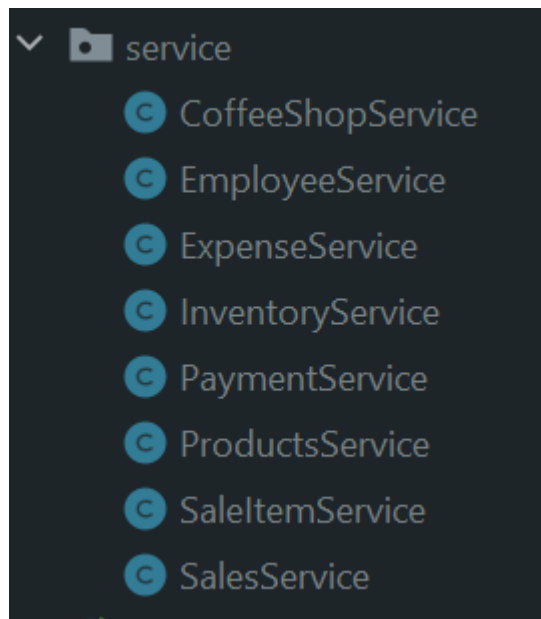


Рисунок 66 - сервисы

Создаю контроллеры для передачи своих данных

```
@Autowired
public CoffeeShopController(CoffeeShopService coffeeShopService) { this.coffeeShopService = coffeeShopService; }

Данила Хренов
@GetMapping
public List<CoffeeShop> getAllShops() { return coffeeShopService.getAll(); }

Данила Хренов
@GetMapping("/{id}")
public CoffeeShop getShopById(@PathVariable Long id) { return coffeeShopService.getById(id); }

Данила Хренов
@PostMapping
public CoffeeShop createShop(@RequestBody CoffeeShop shop) { return coffeeShopService.save(shop); }

Данила Хренов
@PostMapping("/update/{id}")
public CoffeeShop updateShop(@PathVariable Long id, @RequestBody CoffeeShop details) {
    CoffeeShop shop = coffeeShopService.getById(id);
    shop.setName(details.getName());
    shop.setLocation(details.getLocation());
    shop.setPhone(details.getPhone());
    shop.setEmail(details.getEmail());
}
```

Рисунок 67 - контроллеры

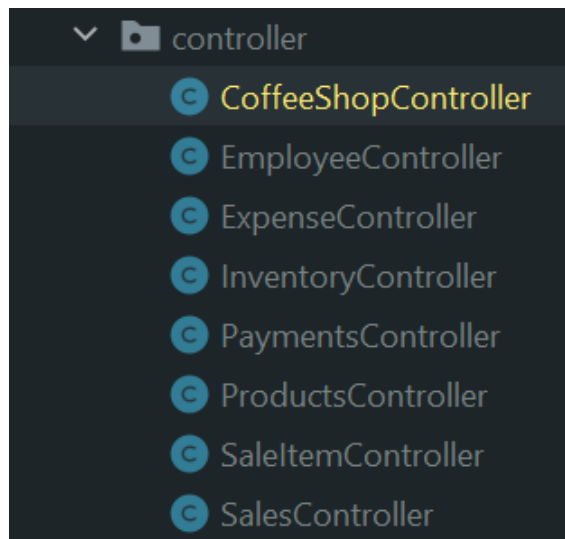


Рисунок 68 – контроллеры бэка

Создаю конфиг для разрешения корсов

```
@Configuration
public class WebConfiguration implements WebMvcConfigurer {
    no usages  Данила Хренов
    @Override
    public void addCorsMappings(CorsRegistry registry) { registry.addMapping( pathPattern: "/*") .al
}
```

Рисунок 69 - конфигурация бэка

Теперь создам фронтенд

Дублирую модели в фронттовую часть приложения

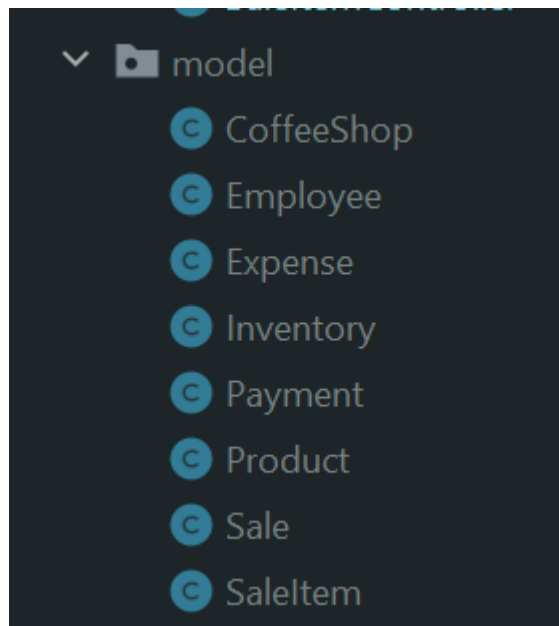


Рисунок 70 - модели

Создаю сервисы для работы с эндпоинтами

```
@Service
public class CoffeeShopApi {

    6 usages
    private final RestTemplate restTemplate;

    Данила Хренов *
    @Autowired
    public CoffeeShopApi(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    3 usages Данила Хренов *
    public CoffeeShop[] getCoffeeShops() {
        String url = "http://localhost:8000/api/v1/coffeeShops";
        return restTemplate.getForObject(url, CoffeeShop[].class);
    }

    Данила Хренов *
    public void save(CoffeeShop shop) {
        String url = "http://localhost:8000/api/v1/coffeeShops";
        restTemplate.postForObject(url, shop, CoffeeShop.class);
    }
}
```

Рисунок 71 – взаимодействие с апи

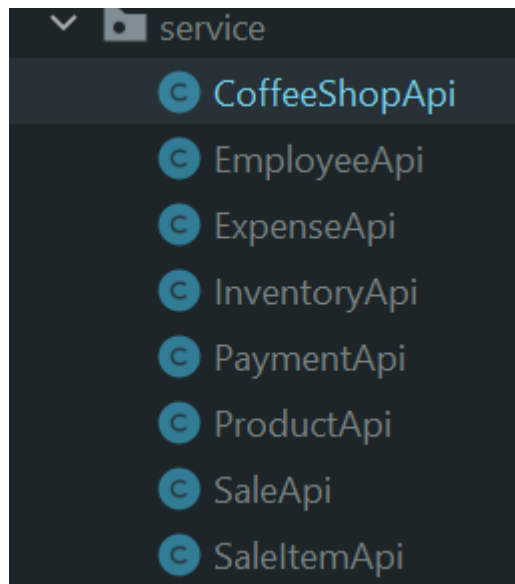


Рисунок 72 - апишки

Создаю контроллеры для взаимодействия с апишкой

```
@Controller
@RequestMapping("/coffeeShops")
public class CoffeeShopController {
    @Autowired
    private CoffeeShopApi coffeeShopApi;

    @GetMapping
    public String getAllShops(Model model) {
        CoffeeShop[] shops = coffeeShopApi.getCoffeeShops();
        model.addAttribute("shops", shops);

        return "shops/coffeeShops";
    }
}
```

Рисунок 73 – все магазины

```

@GetMapping("/{id}")

public String getShopById(@PathVariable Long id, Model model) {

    CoffeeShop shop = coffeeShopApi.getById(id);

    model.addAttribute(attributeName: "shop", shop);

    return "shops/coffeeShop";

}

```

Рисунок 74 - поиск

Данила Хренов *

```

@GetMapping("/create")
public String createShop(Model model) {
    model.addAttribute(attributeName: "shop", new CoffeeShop());

    return "shops/createCoffeeShop";
}

```

Рисунок 75 - создание

Данила Хренов *

```

@PostMapping("/create")
public String saveShop(CoffeeShop shop) {
    coffeeShopApi.save(shop);

    return "redirect:/coffeeShops";
}

```

Рисунок 76 - создание

```

Данила Хренов *
@GetMapping("/{id}")
public String updateShop(@PathVariable Long id, Model model) {
    CoffeeShop shop = coffeeShopApi.getById(id);
    model.addAttribute("shop", shop);

    return "shops/updateCoffeeShop";
}

```

Рисунок 77 - обновление

```

Данила Хренов *
@GetMapping("/{id}")
public String deleteShop(@PathVariable Long id) {
    coffeeShopApi.delete(id);

    return "redirect:/coffeeShops";
}

```

Рисунок 78 - удаление

Создаю статику для своего контроллера

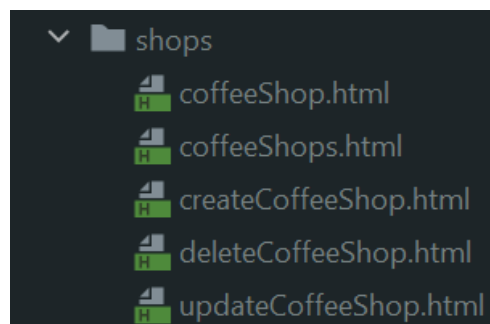


Рисунок 79 – статика магазинов

Повторяю это каждой модели

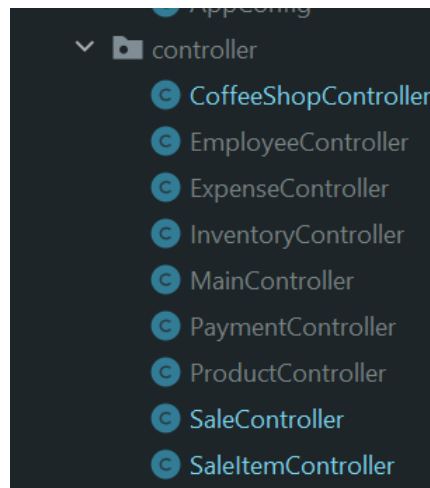


Рисунок 80 - контроллеры

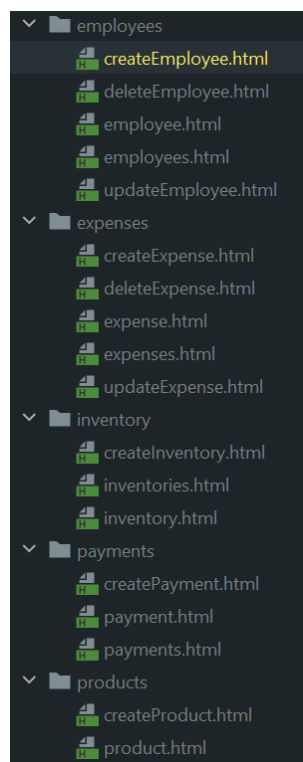


Рисунок 81 - статика

Проверяю:

GET http://localhost:8000/api/v1/coffeeShops

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value
Key	Value

body Cookies (1) Headers (8) Test Results 200

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "shopId": 1,
4     "name": "Coffee Haven",
5     "location": "123 Brew St, CoffeeTown",
6     "phone": "123-456-7890",
7     "email": "contact@coffeehaven.com"
8   },
9   {
10    "shopId": 2,
11    "name": "The Daily Grind",
12    "location": "456 Espresso Ave, BeanCity",
13    "phone": "234-567-8901",
14    "email": "hello@dailgrind.com"
```

Рисунок 82 – возврат данных 1

GET http://localhost:8000/api/v1/employee

Params Authorization Headers (7) Body Pre-request Script Tests Setting

Query Params

Key	Value
Key	Value

body Cookies (1) Headers (8) Test Results 200

Pretty Raw Preview Visualize JSON

```
1 {
2   "employeeId": 1,
3   "coffeeShop": {
4     "shopId": 1,
5     "name": "Coffee Haven",
6     "location": "123 Brew St, CoffeeTown",
7     "phone": "123-456-7890",
8     "email": "contact@coffeehaven.com"
9   },
10  "name": "Alice Smith",
11  "position": "Barista",
12  "hireDate": "2020-01-15",
13  "salary": 28000.00
```

Рисунок 83 – возврат данных 2

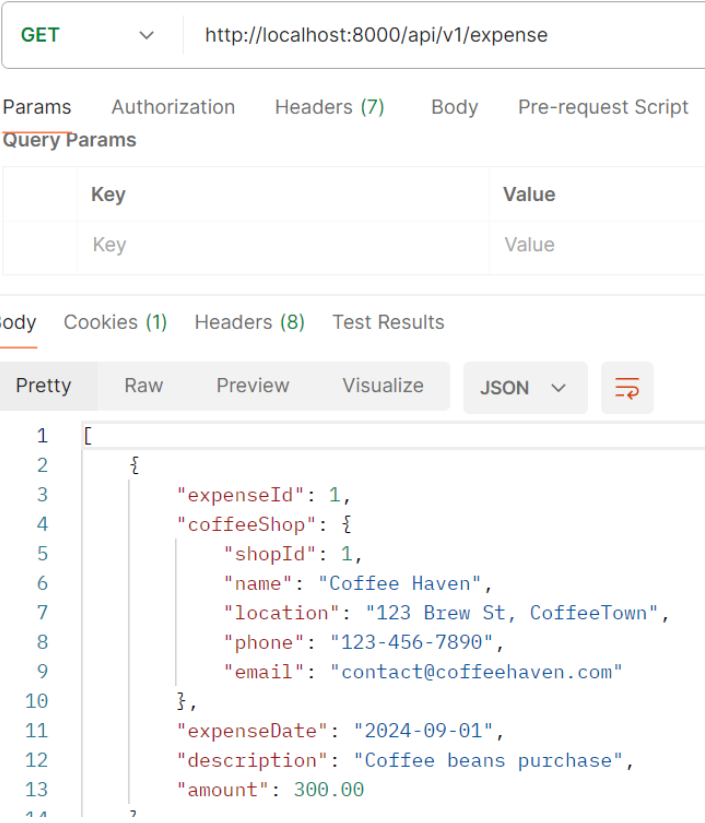


Рисунок 84 – возврат данных 3

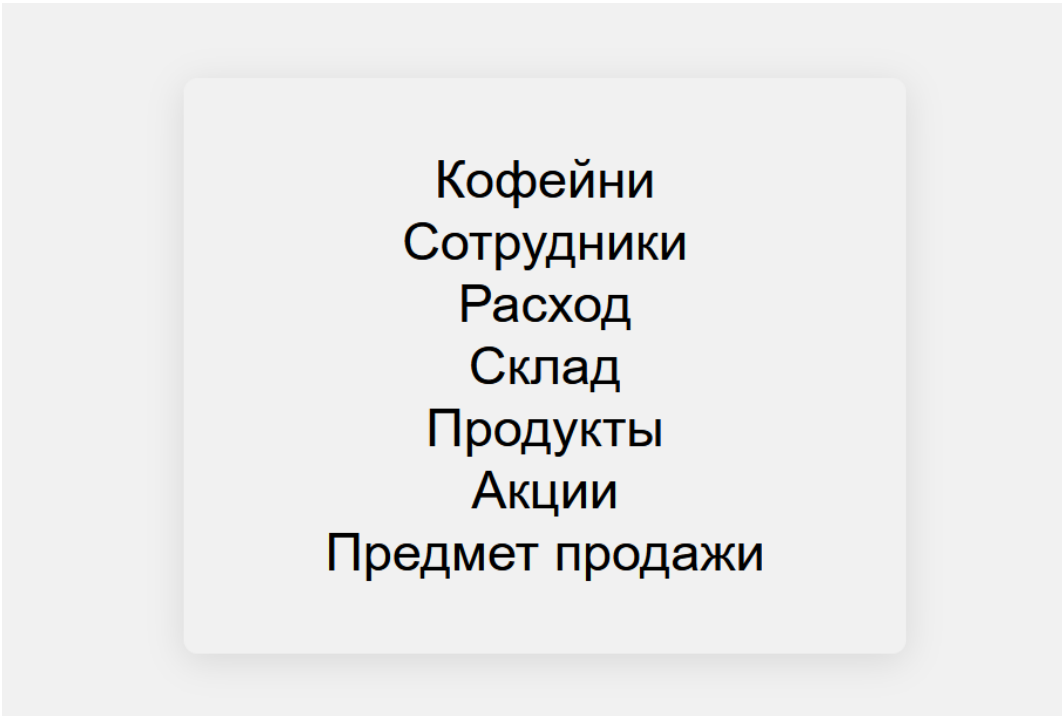


Рисунок 85 - главная

Coffee Shops					
Id	Name	Location	Phone	Email	Actions
1	Coffee Haven	123 Brew St, CoffeeTown	123-456-7890	contact@coffeehaven.com	View Update Delete
2	The Daily Grind	456 Espresso Ave, BeanCity	234-567-8901	hello@dailygrind.com	View Update Delete
3	Java Junction	789 Mug Blvd, CuppaLand	345-678-9012	info@javacoffee.com	View Update Delete
4	Brewed Awakenings	321 Latte Ln, SipCity	456-789-0123	support@brewedawak.com	View Update Delete
5	Bean There	654 Coffee Rd, MochaTown	567-890-1234	sales@beanthere.com	View Update Delete

Create New Shop

Рисунок 86 – все данные

localhost:8081/coffeeShops/1

Coffee Shop

Name:

Location:

Phone:

Email:

Рисунок 87 - уточнение

localhost:8081/coffeeShops/create

Create Coffee Shop

Name:

Location:

Phone:

Email:

Рисунок 88 - добавление

Coffee Shops					
Id	Name	Location	Phone	Email	Actions
1	Coffee Haven	123 Brew St, CoffeeTown	123-456-7890	contact@coffeehaven.com	View Update Delete
2	The Daily Grind	456 Espresso Ave, BeanCity	234-567-8901	hello@dailygrind.com	View Update Delete
3	Java Junction	789 Mug Blvd, CuppaLand	345-678-9012	info@javacoffee.com	View Update Delete
4	Brewed Awakenings	321 Latte Ln, SipCity	456-789-0123	support@brewedawak.com	View Update Delete
5	Bean There	654 Coffee Rd, MochaTown	567-890-1234	sales@beanthere.com	View Update Delete
6	ПриАрбатский	Арбат	88005553535	spp@gmail.com	View Update Delete

Create New Shop

Рисунок 89 - добавление

Вывод: Научился создавать RestApi в spring boot, а так же реализовал бэкэнд и фронтенд приложения

Индивидуальный проект

Тема: Автоматизированная бухгалтерия

Цель: Реализовать онлайн бухгалтерию для кофейни на spring boot, с бэкэнд и фронтенд частями приложения

Проект "Бухгалтерия кофейни" направлен на автоматизацию учета финансовых операций, сопутствующих бизнес-процессам кофейни. Это решение позволит управлять доходами, расходами, клиентской базой и товарными запасами, обеспечивая удобный интерфейс для сотрудников кофейни и владельцев бизнеса.

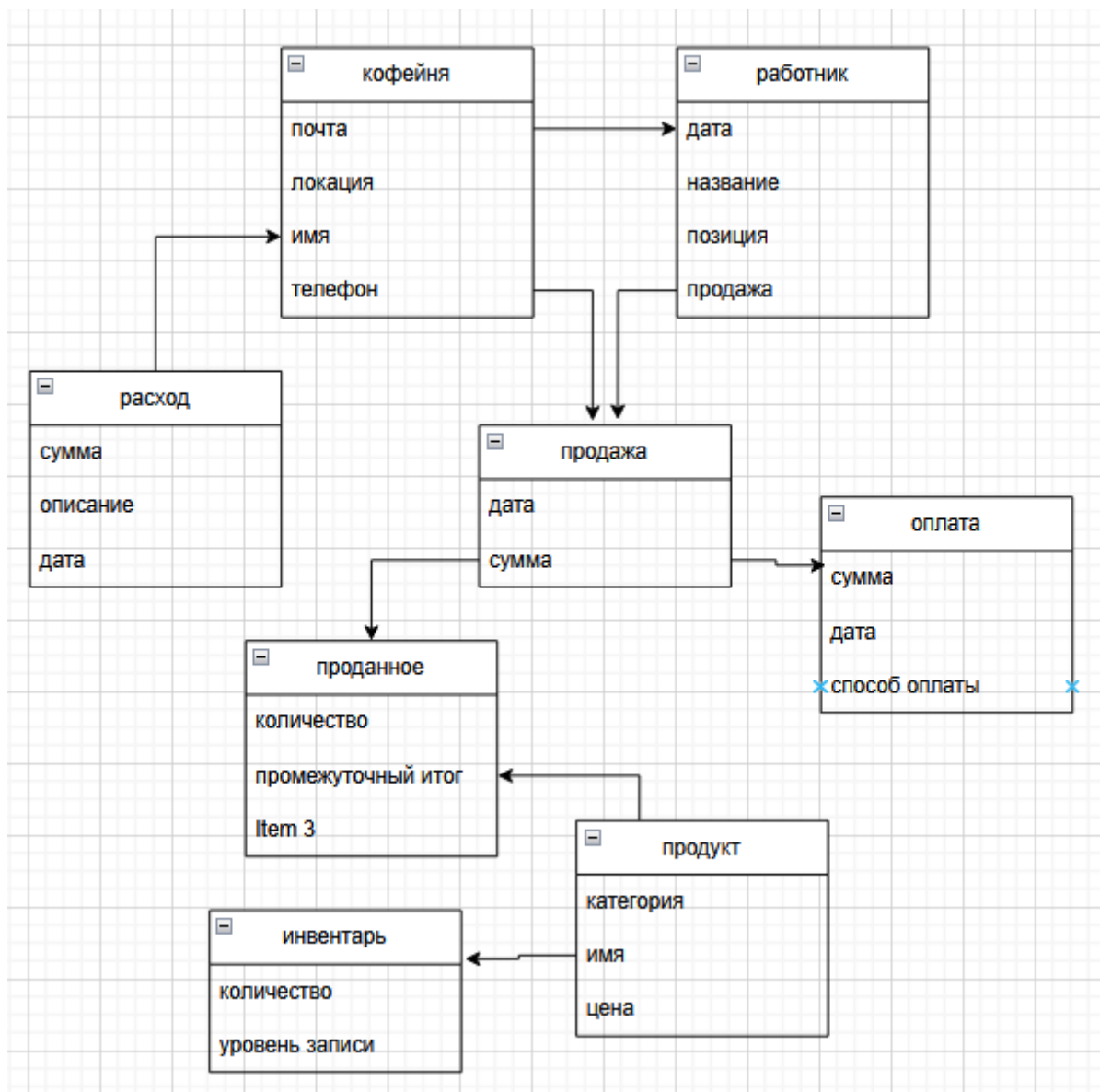


Рисунок 90 – инфологическая схема

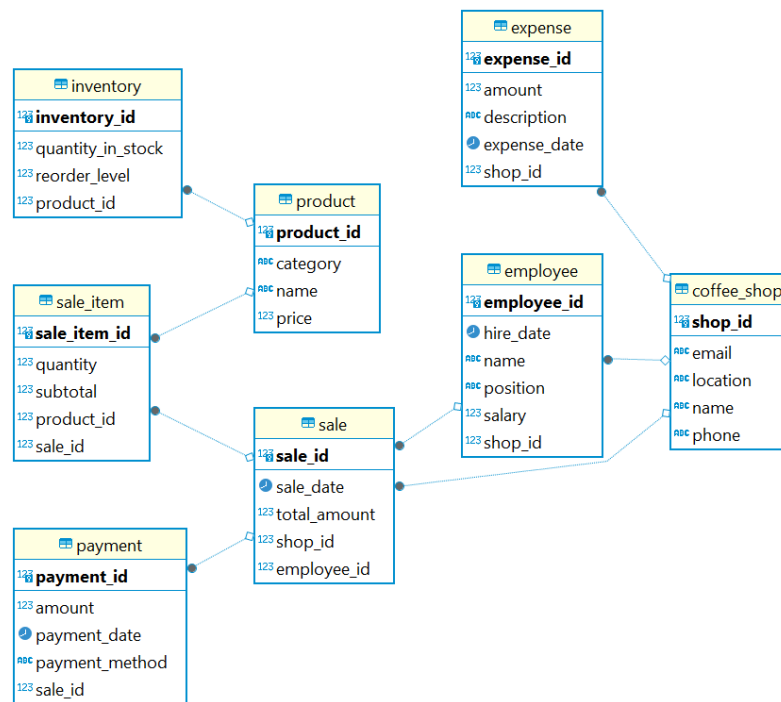


Рисунок 91 – даталогическая модель

Таблица 1 - expense

Поле	Тип	Формат	Ключ
Expense_id	int	int	pk
description	nvarchar	nvarchar	
Expense_date	date	date	
Shop_id	int	int	fk

Таблица 2 – coffee_chop

Поле	Тип	Формат	Ключ
shop_id	int	int	pk

email	nvarchar	nvarchar	
location	nvarchar	nvarchar	
name	nvarchar	nvarchar	
phone	nvarchar	nvarchar	

Таблица 3 - employee

Поле	Тип	Формат	Ключ
employee_id	int	int	pk
Hire_date	date	date	
name	nvarchar	nvarchar	
position	nvarchar	nvarchar	
salary	int	int	
Shop_id	int	int	fk

Таблица 4 – sale

Поле	Тип	Формат	Ключ
sale_id	int	int	pk
sale_date	date	date	
total_amount	int	int	
Shop_id	int	int	fk

Поле	Тип	Формат	Ключ
Employee_id	int	int	fk

Таблица 5 - payment

Поле	Тип	Формат	Ключ
payment_id	int	int	pk
amount	int	int	
Payment_date	date	date	
Payment_method	nvarchar	nvarchar	
sale_id	int	int	fk

Таблица 6 – sale_item

Поле	Тип	Формат	Ключ
Sale_item_id	int	int	pk
quantity	int	int	
subtotal	int	int	
Product_id	int	int	fk
sale_id	int	int	fk

Таблица 7 - product

Поле	Тип	Формат	Ключ
product_id	int	int	pk
category	nvarchar	nvarchar	
name	int	int	
price	int	int	

Таблица 8 - inventory

Поле	Тип	Формат	Ключ
inventory_id	int	int	pk
Quantity_in_stock	int	int	
Reorder_level	int	int	
Product_id	int	int	

Скрипт базы данных:

CREATE TABLE coffee_shop (

shop_id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

location VARCHAR(255),

phone VARCHAR(15),

email VARCHAR(100)

);

-- Таблица сотрудников

CREATE TABLE employee (

employee_id SERIAL PRIMARY KEY,

shop_id INTEGER REFERENCES coffee_shops(shop_id),

name VARCHAR(100) NOT NULL,

position VARCHAR(50),

hire_date DATE,

salary DECIMAL(10, 2)

);

-- Таблица продуктов

```
CREATE TABLE product (  
  
    product_id SERIAL PRIMARY KEY,  
  
    name VARCHAR(100) NOT NULL,  
  
    price DECIMAL(10, 2) NOT NULL,  
  
    category VARCHAR(50)  
  
);
```

-- Таблица продаж

```
CREATE TABLE sale (  
  
    sale_id SERIAL PRIMARY KEY,  
  
    shop_id INTEGER REFERENCES coffee_shops(shop_id),  
  
    employee_id INTEGER REFERENCES employees(employee_id),  
  
    sale_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  
    total_amount DECIMAL(10, 2) NOT NULL
```

);

-- Таблица товаров на продажах

CREATE TABLE sale_item (

sale_item_id SERIAL PRIMARY KEY,

sale_id INTEGER REFERENCES sales(sale_id),

product_id INTEGER REFERENCES products(product_id),

quantity INTEGER NOT NULL,

subtotal DECIMAL(10, 2) NOT NULL

);

-- Таблица расходов

CREATE TABLE expense (

expense_id SERIAL PRIMARY KEY,


```
shop_id INTEGER REFERENCES coffee_shops(shop_id),

expense_date DATE,

description VARCHAR(255),

amount DECIMAL(10, 2) NOT NULL

);
```

-- Таблица запасов

```
CREATE TABLE inventory (

inventory_id SERIAL PRIMARY KEY,

product_id INTEGER REFERENCES products(product_id),

quantity_in_stock INTEGER NOT NULL,

reorder_level INTEGER

);
```

-- Таблица платежей

```
CREATE TABLE payment (  
  
    payment_id SERIAL PRIMARY KEY,  
  
    sale_id INTEGER REFERENCES sales(sale_id),  
  
    payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  
    payment_method VARCHAR(50),  
  
    amount DECIMAL(10, 2) NOT NULL  
  
);
```

CoffeShopController.java:

```
package com.example.front_buhg.controller;  
  
import com.example.front_buhg.model.CoffeeShop;  
import com.example.front_buhg.service.CoffeeShopApi;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
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.RequestMapping;
```

@Controller

```

@RequestMapping("/coffeeShops")
public class CoffeeShopController {
    @Autowired
    private CoffeeShopApi coffeeShopApi;

    @GetMapping
    public String getAllShops(Model model) {
        CoffeeShop[] shops = coffeeShopApi.getCoffeeShops();
        model.addAttribute("shops", shops);

        return "shops/coffeeShops";
    }

    @GetMapping("/{id}")
    public String getShopById(@PathVariable Long id, Model model) {
        CoffeeShop shop = coffeeShopApi.getById(id);
        model.addAttribute("shop", shop);

        return "shops/coffeeShop";
    }

    @GetMapping("/create")
    public String createShop(Model model) {
        model.addAttribute("shop", new CoffeeShop());

        return "shops/createCoffeeShop";
    }

    @PostMapping("/create")
    public String saveShop(CoffeeShop shop) {
        coffeeShopApi.save(shop);
    }

```

```
        return "redirect:/coffeeShops";
    }
}
```

```
@GetMapping("/update/{id}")
public String updateShop(@PathVariable Long id, Model model) {
    CoffeeShop shop = coffeeShopApi.getById(id);
    model.addAttribute("shop", shop);

    return "shops/updateCoffeeShop";
}
```

```
@PostMapping("/update/{id}")
public String updateShop(@PathVariable Long id, CoffeeShop shop) {
    coffeeShopApi.update(shop, id);

    return "redirect:/coffeeShops";
}
```

```
@GetMapping("/delete/{id}")
public String deleteShop(@PathVariable Long id) {
    coffeeShopApi.delete(id);

    return "redirect:/coffeeShops";
}

}
```

EmployeeController.java:

```
package com.example.front_buhg.controller;
```

```
import com.example.front_buhg.model.Employee;
import com.example.front_buhg.service.CoffeeShopApi;
import com.example.front_buhg.service.EmployeeApi;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
```

```
@Controller
```

```
@RequestMapping("/employee")
```

```
public class EmployeeController {
```

```
    @Autowired
```

```
    private EmployeeApi employeeApi;
```

```
    @Autowired
```

```
    private CoffeeShopApi shopApi;
```

```
    @GetMapping
```

```
    public String getAll(Model model) {
```

```
        Employee[] employee = employeeApi.getAll();
```

```
        model.addAttribute("employees", employee);
```

```
        return "employees/employees";
```

```
    }
```

```
    @GetMapping("/{id}")
```

```
    public String getById(@PathVariable Long id, Model model) {
```

```
        Employee employee = employeeApi.getById(id);
```

```
        model.addAttribute("employee", employee);

        return "employees/employee";

    }
}
```

```
@GetMapping("/create")
```

```
public String create(Model model) {
```

```
    model.addAttribute("employee", new Employee());
```

```
    return "employees/createEmployee";
```

```
}
```

```
@PostMapping("/create")
```

```
public String save(@RequestParam Long shopId, Employee employee) {
```

```
    employee.setCoffeeShop(shopApi.getById(shopId));
```

```
    employeeApi.save(employee);
```

```
    return "redirect:/employees";
```

```
}
```

```
@GetMapping("/update/{id}")
```

```
public String update(@PathVariable Long id, Model model) {
```

```
    Employee employee = employeeApi.getById(id);
```

```

        model.addAttribute("employee", employee);

        return "employees/updateEmployee";

    }

    @PostMapping("/update/{id}")
    public String update(@PathVariable Long id, @RequestParam Long shopId, Employee
employee) {

        employee.setCoffeeShop(shopApi.getById(shopId));
        employeeApi.update(employee, id);

        return "redirect:/employees";

    }

    @GetMapping("/delete/{id}")

    public String delete(@PathVariable Long id) {

        employeeApi.delete(id);

        return "redirect:/employees";

    }
}

```

ExpenseController.java:

```

package com.example.front_buhg.controller;

import com.example.front_buhg.model.Expense;

```

```
import com.example.front_buhg.service.CoffeeShopApi;
import com.example.front_buhg.service.ExpenseApi;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
```

```
@Controller
```

```
@RequestMapping("/expenses")
```

```
public class ExpenseController {
```

```
    @Autowired
```

```
    private ExpenseApi expenseApi;
```

```
    @Autowired
```

```
    private CoffeeShopApi shopApi;
```

```
@GetMapping
```

```
public String getAllExpenses(Model model) {
```

```
    Expense[] expenses = expenseApi.getAll();
```

```
    model.addAttribute("expenses", expenses);
```

```
    return "expenses/expenses";
```

```
}
```

```
@GetMapping("/{id}")
```

```
public String getExpenseById(@PathVariable Long id, Model model) {
```

```
    Expense expense = expenseApi.getById(id);
```

```
    model.addAttribute("expense", expense);
```

```
    return "expenses/expense";
```

```
}
```



```
@GetMapping("/create")
public String createExpense(Model model) {
    model.addAttribute("expense", new Expense());

    return "expenses/createExpense";
}
```

```
@PostMapping("/create")
public String saveExpense(@RequestParam Long shopId, Expense expense) {
    expense.setCoffeeShop(shopApi.getById(shopId));
    expenseApi.save(expense);

    return "redirect:/expenses";
}
```

```
@GetMapping("/update/{id}")
public String updateExpense(@PathVariable Long id, Model model) {
    Expense expense = expenseApi.getById(id);
    model.addAttribute("expense", expense);

    return "expenses/updateExpense";
}
```

```
@PostMapping("/update/{id}")
public String updateExpense(@PathVariable Long id, @RequestParam Long shopId, Expense
expense) {
    expense.setCoffeeShop(shopApi.getById(shopId));
    expenseApi.update(expense, id);

    return "redirect:/expenses";
}
```

```

    @GetMapping("/delete/{id}")
    public String deleteExpense(@PathVariable Long id) {
        expenseApi.delete(id);

        return "redirect:/expenses";
    }
}

```

InventoryController.java:

```

package com.example.front_buhg.controller;

import com.example.front_buhg.model.Inventory;
import com.example.front_buhg.model.Product;
import com.example.front_buhg.service.InventoryApi;
import com.example.front_buhg.service.ProductApi;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping("/inventories")
public class InventoryController {

    @Autowired
    private InventoryApi inventoryApi;

    @Autowired
    private ProductApi productApi;

    @GetMapping
    public String getAllInventories(Model model) {

```

```

    Inventory[] inventories = inventoryApi.getAll();
    model.addAttribute("inventories", inventories);
    return "inventory/inventories";
}

```

```

@GetMapping("/{id}")
public String getInventoryById(@PathVariable Long id, Model model) {
    Inventory inventory = inventoryApi.getById(id);
    Product[] products = productApi.getAll();
    model.addAttribute("inventory", inventory);
    model.addAttribute("products", products);
    return "inventory/inventory";
}

```

```

@GetMapping("/create")
public String createInventory(Model model) {
    Product[] products = productApi.getAll();
    model.addAttribute("inventory", new Inventory());
    model.addAttribute("products", products);
    return "inventory/createInventory";
}

```

```

@PostMapping("/create")
public String saveInventory(@RequestParam Long prodId, Inventory inventory) {
    inventory.setProduct(productApi.getById(prodId));
    inventoryApi.save(inventory);
    return "redirect:/inventories";
}

```

```

@PostMapping("/update/{id}")
public String updateInventory(
    @PathVariable Long id,
    @RequestParam Long prodId,
    @RequestParam Integer quantityInStock,
    @RequestParam Integer reorderLevel

```

```

        ) {
            Inventory updated = inventoryApi.getById(id);
            updated.setProduct(productApi.getById(prodId));
            updated.setQuantityInStock(quantityInStock);
            updated.setReorderLevel(reorderLevel);

            inventoryApi.update(updated, id);
            return "redirect:/inventories";

        }

        @GetMapping("/delete/{id}")
        public String deleteInventory(@PathVariable Long id) {
            inventoryApi.delete(id);
            return "redirect:/inventories";
        }
    }

```

MainController.java:

```

package com.example.front_buhg.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping
public class MainController {

    @GetMapping
    public String mainStart(){
        return "main";
    }
}

```

ProductController.java:

```
package com.example.front_buhg.controller;

import com.example.front_buhg.model.Product;
import com.example.front_buhg.service.ProductApi;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
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.RequestMapping;
```

@Controller

@RequestMapping("/products")

```
public class ProductController {
```

@Autowired

```
    private ProductApi productApi;
```

@GetMapping

```
    public String getAllProducts(Model model) {
        Product[] products = productApi.getAll();
        model.addAttribute("products", products);
        return "products/products";
    }
```

@GetMapping("/{id}")

```
    public String getProductById(@PathVariable Long id, Model model) {
        Product product = productApi.getById(id);
        model.addAttribute("product", product);
        return "products/product";
    }
```

@GetMapping("/create")

```
    public String createProduct(Model model) {
```

```

        model.addAttribute("product", new Product());
        return "products/createProduct";
    }

```

```

    @PostMapping("/create")
    public String saveProduct(Product product) {
        productApi.save(product);
        return "redirect:/products";
    }

```

```

    @PostMapping("/update/{id}")
    public String updateProduct(@PathVariable Long id, Product product) {
        productApi.update(product, id);
        return "redirect:/products";
    }

```

```

    @GetMapping("/delete/{id}")
    public String deleteProduct(@PathVariable Long id) {
        productApi.delete(id);
        return "redirect:/products";
    }
}

```

SaleController.java:

```

package com.example.front_buhg.controller;

import com.example.front_buhg.model.Sale;
import com.example.front_buhg.service.CoffeeShopApi;
import com.example.front_buhg.service.EmployeeApi;
import com.example.front_buhg.service.SaleApi;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;

```

```
import org.springframework.web.bind.annotation.*;
```

```
@Controller
```

```
@RequestMapping("/sales")
```

```
public class SaleController {
```

```
    @Autowired
```

```
    private SaleApi saleApi;
```

```
    @Autowired
```

```
    private CoffeeShopApi coffeeShopApi;
```

```
    @Autowired
```

```
    private EmployeeApi employeeApi;
```

```
    @GetMapping
```

```
    public String getAllProducts(Model model) {
```

```
        Sale[] products = saleApi.getAll();
```

```
        model.addAttribute("sales", products);
```

```
        return "sales/sales";
```

```
    }
```

```
    @GetMapping("/{id}")
```

```
    public String getProductById(@PathVariable Long id, Model model) {
```

```
        Sale sale = saleApi.getById(id);
```

```
        model.addAttribute("sale", sale);
```

```
        model.addAttribute("shops", coffeeShopApi.getCoffeeShops());
```

```
        model.addAttribute("employees", employeeApi.getAll());
```

```
        return "sales/sale";
```

```
    }
```

```
    @GetMapping("/create")
```

```
    public String createProduct(Model model) {
```

```
        model.addAttribute("sale", new Sale());
```

```

        model.addAttribute("shops", coffeeShopApi.getCoffeeShops());
        model.addAttribute("employees", employeeApi.getAll());
        return "sales/createSale";
    }

    @PostMapping("/create")
    public String saveProduct(@RequestParam Long shopId, @RequestParam Long employeeId,
        Sale sale) {
        sale.setCoffeeShop(coffeeShopApi.getById(shopId));
        sale.setEmployee(employeeApi.getById(employeeId));
        saleApi.save(sale);
        return "redirect:/sales";
    }

    @PostMapping("/update/{id}")
    public String updateProduct(@PathVariable Long id, @RequestParam Long shopId,
        @RequestParam Long employeeId, Sale sale) {
        sale.setCoffeeShop(coffeeShopApi.getById(shopId));
        sale.setEmployee(employeeApi.getById(employeeId));
        saleApi.update(sale, id);
        return "redirect:/sales";
    }

    @GetMapping("/delete/{id}")
    public String deleteProduct(@PathVariable Long id) {
        saleApi.delete(id);
        return "redirect:/sales";
    }
}

```

SaleItemController.java:

```

package com.example.front_buhg.controller;

```



```
import com.example.front_buhg.model.SaleItem;
import com.example.front_buhg.service.CoffeeShopApi;
import com.example.front_buhg.service.ProductApi;
import com.example.front_buhg.service.SaleApi;
import com.example.front_buhg.service.SaleItemApi;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
```

```
@Controller
```

```
@RequestMapping("/saleItems")
```

```
public class SaleItemController {
```

```
    @Autowired
```

```
    private SaleItemApi saleItemApi;
```

```
    @Autowired
```

```
    private SaleApi saleApi;
```

```
    @Autowired
```

```
    private ProductApi productApi;
```

```
    @GetMapping
```

```
    public String getAllProducts(Model model) {
        SaleItem[] products = saleItemApi.getAll();
        model.addAttribute("saleItems", products);
        return "saleItems/saleItems";
    }
```

```
    @GetMapping("/{id}")
```

```
    public String getProductById(@PathVariable Long id, Model model) {
        SaleItem saleItem = saleItemApi.getById(id);
        model.addAttribute("saleItem", saleItem);
    }
```

```
    model.addAttribute("sales", saleApi.getAll());
    model.addAttribute("products", productApi.getAll());
    return "saleItems/saleItem";
}
```

```
@GetMapping("/create")
public String createProduct(Model model) {
    model.addAttribute("saleItem", new SaleItem());

    model.addAttribute("sales", saleApi.getAll());
    model.addAttribute("products", productApi.getAll());
    return "saleItems/createSaleItem";
}
```

```
@PostMapping("/create")
public String saveProduct(@RequestParam Long productId, @RequestParam Long saleId,
SaleItem saleItem) {
    saleItem.setProduct(productApi.getById(productId));
    saleItem.setSale(saleApi.getById(saleId));
    saleItemApi.save(saleItem);
    return "redirect:/saleItems";
}
```

```
@PostMapping("/update/{id}")
public String updateProduct(@PathVariable Long id, @RequestParam Long productId,
@PathVariable Long saleId, SaleItem saleItem) {
    saleItem.setProduct(productApi.getById(productId));
    saleItem.setSale(saleApi.getById(saleId));
    saleItemApi.update(saleItem, id);
    return "redirect:/saleItems";
}
```

```
@GetMapping("/delete/{id}")
```

```

public String deleteProduct(@PathVariable Long id) {
    saleItemApi.delete(id);
    return "redirect:/saleItems";
}
}

```

Сначала создам бэкэнд часть приложения

Создаю модели своих таблиц

```

@Entity
public class CoffeeShop {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long shopId;

    4 usages
    @Column(nullable = false)
    private String name;

    4 usages
    private String location;

    4 usages
    private String phone;

    4 usages
    private String email;
}

```

Рисунок 92 – пример модели

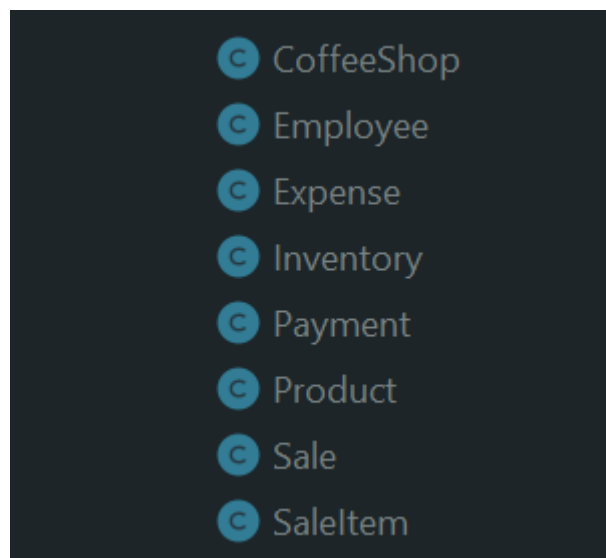


Рисунок 93 – модели бэка

Для каждой модели создаю Jpa репозиторий для реализации crud операций

```

3 usages  Данила Хренов
public interface CoffeeShopRepository extends JpaRepository<CoffeeShop, Long> {
}

```

Рисунок 94 – пример репозитория

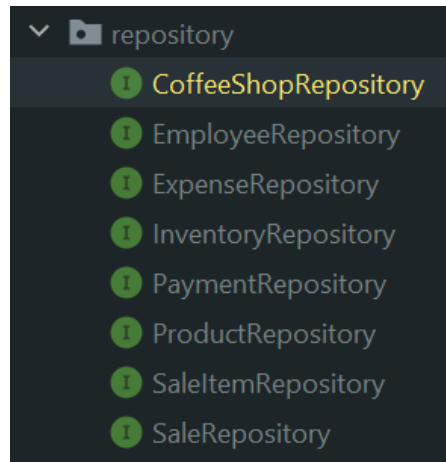


Рисунок 95 - репозитории

Далее создаю сервисы для каждого репозитория, которые пользуются методами этих репозиторий

```

5 usages
private final CoffeeShopRepository coffeeShopRepository;

Данила Хренов
@Autowired
public CoffeeShopService(CoffeeShopRepository coffeeShopRepository) {
    this.coffeeShopRepository = coffeeShopRepository;
}

Данила Хренов
public List<CoffeeShop> getAll() { return coffeeShopRepository.findAll(); }

Данила Хренов
public CoffeeShop getById(Long id) {
    return coffeeShopRepository.findById(id).orElseThrow(() -> new RuntimeException("Shop not found"));
}

Данила Хренов
public CoffeeShop save(CoffeeShop shop) { return coffeeShopRepository.save(shop); }

Данила Хренов
public void delete(Long id) { coffeeShopRepository.deleteById(id); }

```

Рисунок 96 – сервис магазинов

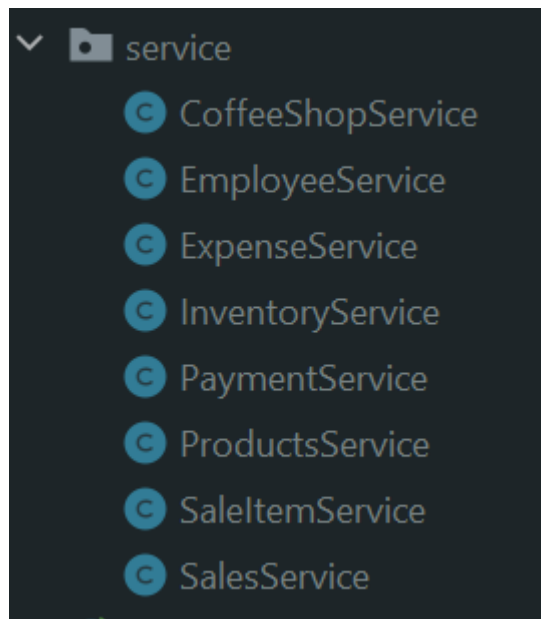


Рисунок 97 - сервисы

Создаю контроллеры для передачи своих данных

```
@Autowired
public CoffeeShopController(CoffeeShopService coffeeShopService) { this.coffeeShopService = coffeeShopService; }

Данила Хренов
@GetMapping
public List<CoffeeShop> getAllShops() { return coffeeShopService.getAll(); }

Данила Хренов
@GetMapping("/{id}")
public CoffeeShop getShopById(@PathVariable Long id) { return coffeeShopService.getById(id); }

Данила Хренов
@PostMapping
public CoffeeShop createShop(@RequestBody CoffeeShop shop) { return coffeeShopService.save(shop); }

Данила Хренов
@PostMapping("/update/{id}")
public CoffeeShop updateShop(@PathVariable Long id, @RequestBody CoffeeShop details) {
    CoffeeShop shop = coffeeShopService.getById(id);
    shop.setName(details.getName());
    shop.setLocation(details.getLocation());
    shop.setPhone(details.getPhone());
    shop.setEmail(details.getEmail());
}
```

Рисунок 98 - контроллеры

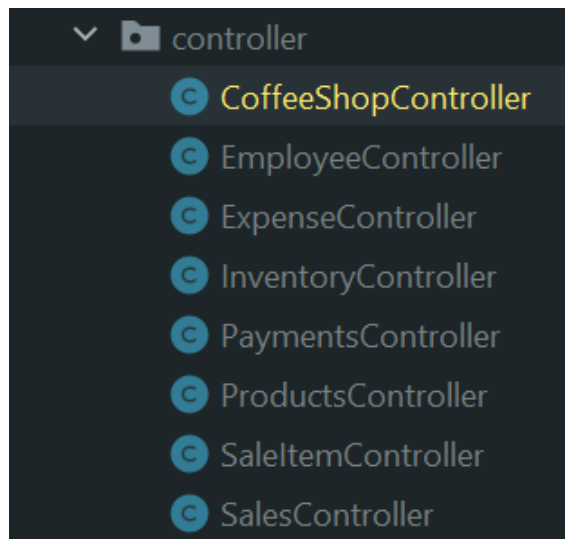


Рисунок 99 – контроллеры бэка

Создаю конфиг для разрешения курсов

```
@Configuration
public class WebConfiguration implements WebMvcConfigurer {
    no usages  Данила Хренов
    @Override
    public void addCorsMappings(CorsRegistry registry) { registry.addMapping( pathPattern: "/*") .al
}
```

Рисунок 100 - конфигурация бэка

Теперь создам фронтенд

Дублирую модели в фронттовую часть приложения

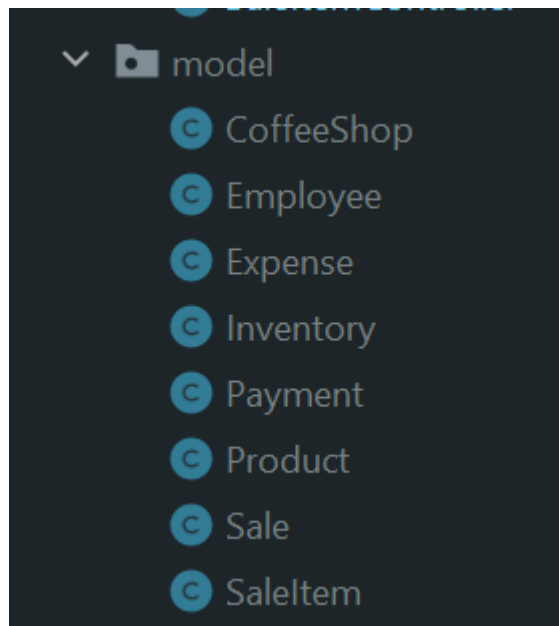


Рисунок 101 - модели

Создаю сервисы для работы с эндпоинтами

```
@Service
public class CoffeeShopApi {

    6 usages
    private final RestTemplate restTemplate;

    Данила Хренов *
    @Autowired
    public CoffeeShopApi(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    3 usages Данила Хренов *
    public CoffeeShop[] getCoffeeShops() {
        String url = "http://localhost:8000/api/v1/coffeeShops";
        return restTemplate.getForObject(url, CoffeeShop[].class);
    }

    Данила Хренов *
    public void save(CoffeeShop shop) {
        String url = "http://localhost:8000/api/v1/coffeeShops";
        restTemplate.postForObject(url, shop, CoffeeShop.class);
    }
}
```

Рисунок 102 – взаимодействие с апи

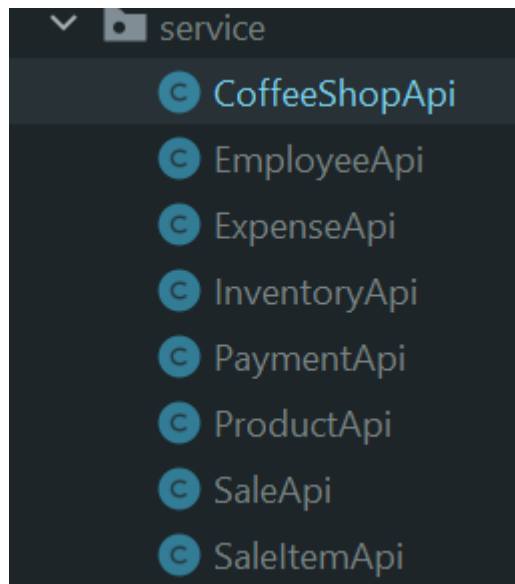


Рисунок 103 - апишки

Создаю контроллеры для взаимодействия с апишкой

```
@Controller
@RequestMapping("/coffeeShops")
public class CoffeeShopController {
    @Autowired
    private CoffeeShopApi coffeeShopApi;

    @GetMapping
    public String getAllShops(Model model) {
        CoffeeShop[] shops = coffeeShopApi.getCoffeeShops();
        model.addAttribute("shops", shops);

        return "shops/coffeeShops";
    }
}
```

Рисунок 104 – все магазины


```

@GetMapping("/{id}")

public String getShopById(@PathVariable Long id, Model model) {

    CoffeeShop shop = coffeeShopApi.getById(id);

    model.addAttribute(attributeName: "shop", shop);

    return "shops/coffeeShop";

}

```

Рисунок 105 - поиск

Данила Хренов *

```

@GetMapping("/create")
public String createShop(Model model) {
    model.addAttribute(attributeName: "shop", new CoffeeShop());

    return "shops/createCoffeeShop";
}

```

Рисунок 106 - создание

Данила Хренов *

```

@PostMapping("/create")
public String saveShop(CoffeeShop shop) {
    coffeeShopApi.save(shop);

    return "redirect:/coffeeShops";
}

```

Рисунок 107 - создание

```

Данила Хренов *
@GetMapping("/{id}")
public String updateShop(@PathVariable Long id, Model model) {
    CoffeeShop shop = coffeeShopApi.getById(id);
    model.addAttribute("shop", shop);

    return "shops/updateCoffeeShop";
}

```

Рисунок 108 - обновление

```

Данила Хренов *
@GetMapping("/{id}")
public String deleteShop(@PathVariable Long id) {
    coffeeShopApi.delete(id);

    return "redirect:/coffeeShops";
}

```

Рисунок 10992 - удаление

Создаю статику для своего контроллера

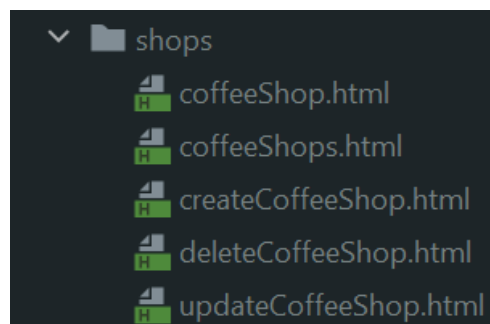


Рисунок 110 – статика магазинов

Повторяю это каждой модели

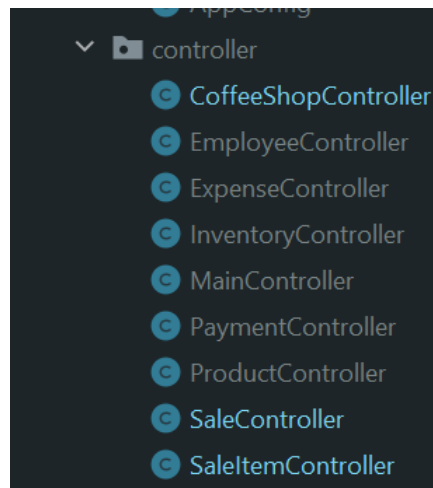


Рисунок 111 - контроллеры

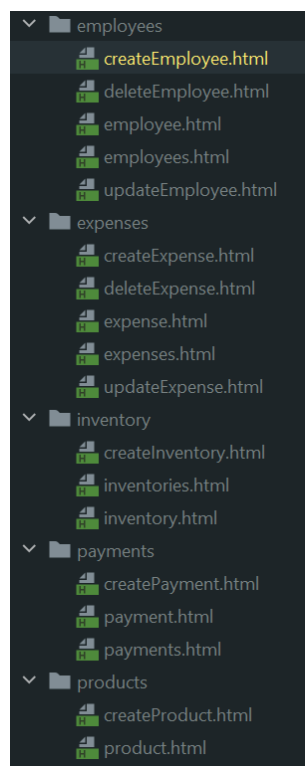


Рисунок 112 - статика

Проверяю:

GET http://localhost:8000/api/v1/coffeeShops

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value
Key	Value

body Cookies (1) Headers (8) Test Results 200

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "shopId": 1,
4     "name": "Coffee Haven",
5     "location": "123 Brew St, CoffeeTown",
6     "phone": "123-456-7890",
7     "email": "contact@coffeehaven.com"
8   },
9   {
10    "shopId": 2,
11    "name": "The Daily Grind",
12    "location": "456 Espresso Ave, BeanCity",
13    "phone": "234-567-8901",
14    "email": "hello@dailgrind.com"
```

Рисунок 113 – возврат данных 1

GET http://localhost:8000/api/v1/employee

Params Authorization Headers (7) Body Pre-request Script Tests Setting

Query Params

Key	Value
Key	Value

body Cookies (1) Headers (8) Test Results 200

Pretty Raw Preview Visualize JSON

```
1 {
2   "employeeId": 1,
3   "coffeeShop": {
4     "shopId": 1,
5     "name": "Coffee Haven",
6     "location": "123 Brew St, CoffeeTown",
7     "phone": "123-456-7890",
8     "email": "contact@coffeehaven.com"
9   },
10  "name": "Alice Smith",
11  "position": "Barista",
12  "hireDate": "2020-01-15",
13  "salary": 28000.00
```

Рисунок 114 – возврат данных 2

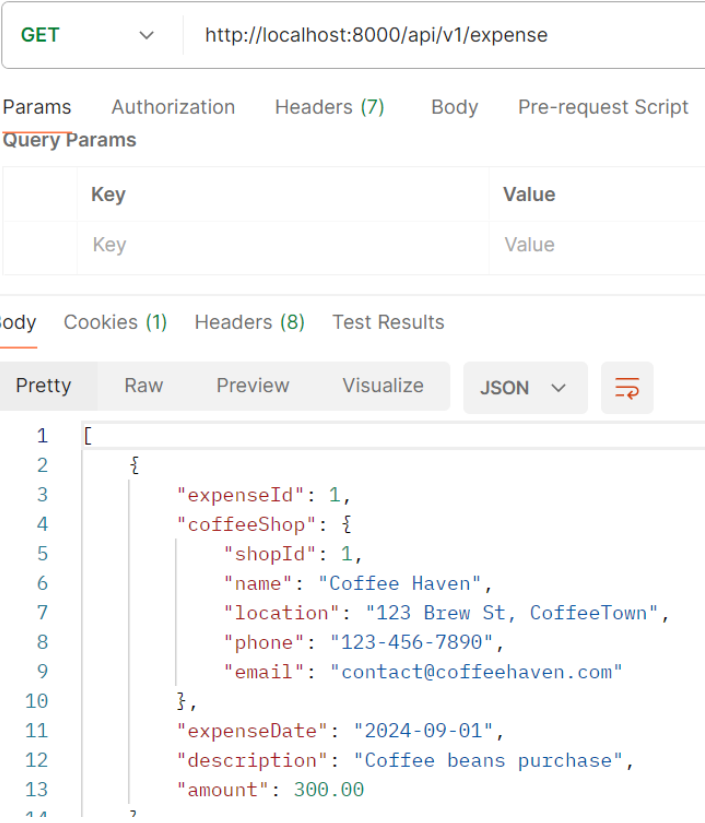


Рисунок 115 – возврат данных 3

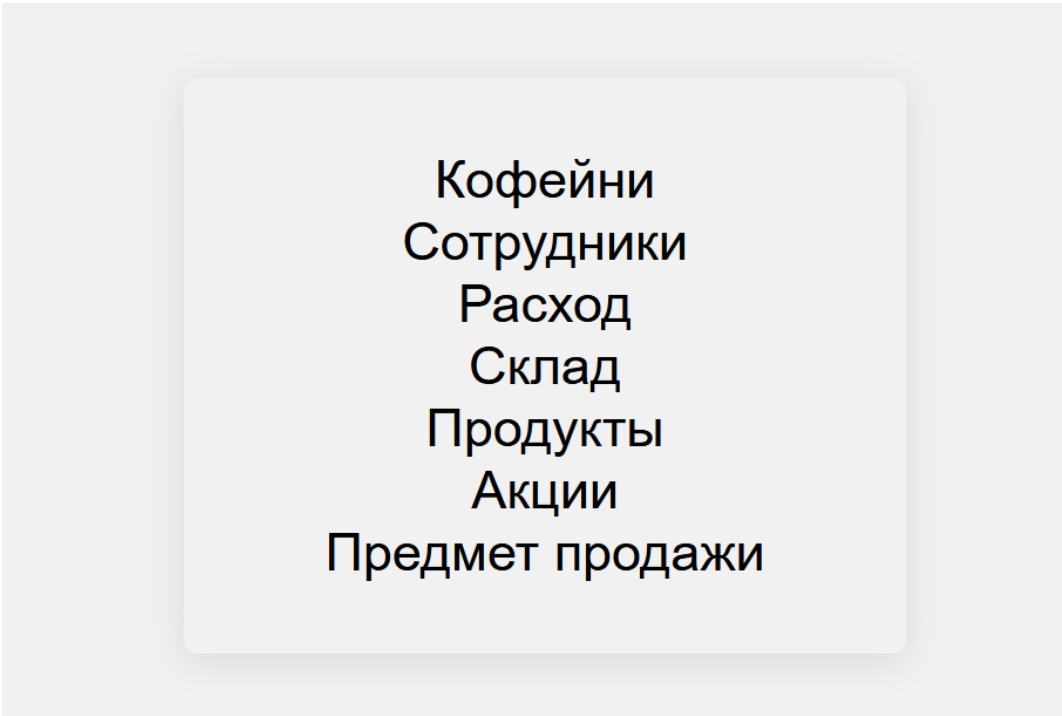


Рисунок 116 - главная

Coffee Shops					
Id	Name	Location	Phone	Email	Actions
1	Coffee Haven	123 Brew St, CoffeeTown	123-456-7890	contact@coffeehaven.com	View Update Delete
2	The Daily Grind	456 Espresso Ave, BeanCity	234-567-8901	hello@dailygrind.com	View Update Delete
3	Java Junction	789 Mug Blvd, CuppaLand	345-678-9012	info@javacoffee.com	View Update Delete
4	Brewed Awakenings	321 Latte Ln, SipCity	456-789-0123	support@brewedawak.com	View Update Delete
5	Bean There	654 Coffee Rd, MochaTown	567-890-1234	sales@beanthere.com	View Update Delete

Create New Shop

Рисунок 117 – все данные

localhost:8081/coffeeShops/1

Coffee Shop

Name:

Location:

Phone:

Email:

Рисунок 118 - уточнение

localhost:8081/coffeeShops/create

Create Coffee Shop

Name:

Location:

Phone:

Email:

Рисунок 119 - добавление

Coffee Shops					
Id	Name	Location	Phone	Email	Actions
1	Coffee Haven	123 Brew St, CoffeeTown	123-456-7890	contact@coffeehaven.com	View Update Delete
2	The Daily Grind	456 Espresso Ave, BeanCity	234-567-8901	hello@dailygrind.com	View Update Delete
3	Java Junction	789 Mug Blvd, CuppaLand	345-678-9012	info@javacoffee.com	View Update Delete
4	Brewed Awakenings	321 Latte Ln, SipCity	456-789-0123	support@brewedawak.com	View Update Delete
5	Bean There	654 Coffee Rd, MochaTown	567-890-1234	sales@beanthere.com	View Update Delete
6	ПриАрбатский	Арбат	88005553535	spp@gmail.com	View Update Delete

Create New Shop

Рисунок 120 - добавление

Вывод: Проект "Бухгалтерия кофейни" является незаменимым инструментом для эффективного управления финансовыми процессами бизнеса. Автоматизация учета позволит сократить время на рутинные

операции, улучшить финансовую прозрачность и способствовать росту прибыльности кофейни.

А так же, в ходе учебной практики по разработке приложения на основе Spring Boot я получил ценный опыт работы с современными инструментами и технологиями для создания веб-приложений. Практика позволила мне:

1. Углубить знания о Spring Framework: Я научился использовать Spring Boot для разработки RESTful приложений, что значительно упростило процесс настройки и конфигурации проекта.
2. Изучить архитектуру приложений: Понимание концепции MVC (Model-View-Controller) и как она реализуется в Spring Boot помогло мне более эффективно организовать код и выделять ответственность между компонентами приложения.
3. Работа с базами данных: Я освоил JPA и Hibernate для работы с реляционными базами данных, что позволило не только создать структуры данных, но и выполнять сложные запросы, обеспечивая высокую производительность и надежность приложения.
4. Построение пользовательского интерфейса: Используя Thymeleaf и Frontend-технологии, я получил опыт разработки интерактивного интерфейса, который улучшает взаимодействие пользователя с приложением.
5. Управление безопасностью: Я изучил основы Spring Security, что дало мне понимание важности защиты данных и пользовательских сессий, а также реализации различных уровней доступа.
6. Анализ и отладка: Работа с инструментами для тестирования и отладки приложений, такими как Postman, позволила мне наладить процесс тестирования, что повысило качество приложения.