Создание web приложения с использованием фреймворка Vaadin. Подключение базы данных через hibernate.

Vaadin — свободно распространяемый фреймворк для создания RIA-веб-приложений, разрабатываемый одноимённой финской компанией. В отличие от библиотек на Javascript и специфических плагинов для браузеров, Vaadin предлагает сервер-ориентированную архитектуру, базирующуюся на Java Enterprise Edition. Использование JEE позволяет выполнять основную часть логики приложения на стороне сервера, тогда как технология AJAX, используемая на стороне браузера, позволяет взаимодействовать с пользователем с интерактивностью, близкой к таковой по эргономике и возможностям настольных приложений. Для отображения элементов пользовательского интерфейса и взаимодействия с сервером на стороне клиента Vaadin использует собственный набор веб-компонентов или JavaScript-библиотеки Vue, React и Angular.

1. В начале создадим проект используя фреймворк Maven:

```
[denis@denis-hpelitebookfolio9470m ~]$ mkdir vaadin-hibernate-maven
[denis@denis-hpelitebookfolio9470m ~]$ cd vaadin-hibernate-maven
[denis@denis-hpelitebookfolio9470m vaadin-hibernate-maven]$ mvn -B archetype:generate -DarchetypeGroupId=com.vaadin -DarchetypeArtifactId=vaadin-archetype-application -DarchetypeVersion=LATEST -DgroupId=org.company -DartifactId=chapter-0.1 -Dversion=1.0-SNAPSHOT
```

Archetype — это набор инструментов для создания шаблонов проектов Maven. Архетип определяется как исходный образец или модель, из которой сделаны все другие вещи того же типа .

Запрашиваем создание архетипа com.vaadin с артефактом vaadin-archetype-application и версией LATEST — последней версией.

Указываем данные своего проекта:
DgroupId=org.company -DartifactId=chapter-0.1 -Dversion=1.0-SNAPSHOT

Следующие строки добавляем в файл pom.xml:

```
<name>My Application</name>
<packaging>war</packaging>
```

Должно быть так:

```
<properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <failOnMissingWebXml>false</failOnMissingWebXml>
    <vaadin.version>23.0.4</vaadin.version>
    <name>My Application</name>
    <packaging>war</packaging>
</properties>
```

Изменяем следующие строки:

```
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
```

на:

```
<maven.compiler.source>17</maven.compiler.source>
<maven.compiler.target>17</maven.compiler.target>
```

Следующие строки вносим между тегами:
<dependencies> </dependencies>

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.0.0.CR1</version>
</dependency>
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.3.3</version>
</dependency>
```

Заменяем следующую зависимость:

```
<plugin>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-maven-plugin</artifactId>
    <version>9.4.36.v20210114</version>
</plugin>
```

На:

```
<plugin>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-maven-plugin</artifactId>
    <version>9.4.43.v20210629</version>
    <configuration>
        <scanIntervalSeconds>2</scanIntervalSeconds>
    </configuration>
</plugin>
```

2. Далее устанавливаем соединение с базой данных через hibernate:
Создаём папку «resources» и маркируем её соответстующим названием.
После, в папке создаём конфигурационный файл hibernate.cfg.xml:

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-configuration xmlns="http://www.hibernate.org/xsd/orm/cfg">
    <session-factory>
        <property name="connection.url">jdbc:postgresql://192.168.43.184/BankDB</property>
        <property name="connection.driver_class">org.postgresql.Driver</property>
        <property name="connection.username">postgres</property>
        <property name="connection.password">1961</property>
        <property name="dialect">org.hibernate.dialect.PostgreSQL91Dialect</property>
        <property name="show_sql">true</property>

        <mapping resource="org/company/database/Employee.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

Подробную информацию о работе с hibernate можно найти в прошлых моих работах с одноимёнными названиями.

3. Создаём таблицу в базе данных «Employee»:
У меня она уже создана:

```
BankDB=# select * from employee;
 id_emp |    name     |   surname    | salary |             email
--------+-------------+--------------+--------+-------------------------------
      1 | Igor        | Fresca       |   5400 | dsharvell0@webeden.co.uk
      3 | Maxim       | Nistru       |   5000 | wgriggs1@amazon.co.uk
      4 | Tudor       | Eonescu      |   5300 | qbisley2@weather.com
      5 | Andrei      | Bejenar      |   7400 | jtremontana3@amazon.co.uk
      6 | Mihaela     | Sudzuker     |   4900 | ncalvert4@bandcamp.com
      8 | Ioana       | Dulce        |   6400 | vkosel6@ask.com
```

id_emp	smallint
name	varchar(20)
surname	varchar(20)
salary	integer,
email	varchar(50)

4. Создаём класс Employee:
Для этого создадим новую папку «database» и в ней класс Employee

```java
package org.company.database;

public class Employee {

    private int idEmp;
    private String name;
    private String surname;
    private String email;
    private int salary;
    @Override
    public String toString() {
        return "Client{" +
                "idEmp=" + idEmp +
                ", name='" + name + '\'' +
                ", surname='" + surname + '\'' +
                ", dateBirth=" + email +
                ", salary=" + salary +
                '}';
    }
    public int getIdEmp() {
        return idEmp;
    }
    public String getName() {
        return name;
    }
    public String getSurname() {
        return surname;
    }
    public String getEmail() {
        return email;
    }
    public int getSalary() {
        return salary;
    }
    public void setIdEmp(int idClient) {
        this.idEmp = idClient;
    }
    public void setName(String name) {
```

```
        this.name = name;
    }
    public void setSurname(String surname) {
        this.surname = surname;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public void setSalary(int telephone) {
        this.salary = telephone;
    }
}
```

5.Создадим файл «Employee.hbm.xml», конфигурационный файл hibernate связывающий таблицу базы данных с классом.
Повторяем полный путь до класса:
Ниже я покажу как распологаются директории в нашем проекте

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping xmlns="http://www.hibernate.org/xsd/hibernate-mapping">

    <class name="org.company.database.Employee" table="employee">
        <id name="idEmp" column="id_emp"/>
        <property name="name" column="name"/>
        <property name="surname" column="surname"/>
        <property name="email" column="email"/>
        <property name="salary" column="salary"/>
    </class>
</hibernate-mapping>
```

6.Для работы с классом Employee через hibernate создадим два класса в новой директории «com.company.DAO»:
«DAO»:

```
package org.company.DAO;

public interface DAO<Entity, Key> {
    void create(Entity entity);
    Entity read(Key key);
    void update(Entity entity);
    void delete(Entity entity);
}
```

«Employee DAO»:

```
package org.company.DAO;

import com.company.database.Employee;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.query.Query;

import java.util.List;

public class EmployeeDAO implements DAO<Employee, Integer>{
    private final SessionFactory factory;
```

```
    public EmployeeDAO(final SessionFactory factory) {
        this.factory = factory;
    }
    @Override
    public void create(final Employee employee) {
        try (final Session session = factory.openSession()) {
            session.beginTransaction();
            session.save(employee);
            session.getTransaction().commit();
        }
    }
    @Override
    public Employee read(final Integer idEmp) {
        try (final Session session = factory.openSession()) {
            return session.get(Employee.class, idEmp);
        }
    }
    @Override
    public void update(final Employee employee) {
        try (Session session = factory.openSession()) {
            session.beginTransaction();
            session.update(employee);
            session.getTransaction().commit();
        }
    }
    @Override
    public void delete(final Employee employee) {
        try (Session session = factory.openSession()) {
            session.beginTransaction();
            session.delete(employee);
            session.getTransaction().commit();
        }
    }
    public List getList() {
        List<Employee> list = null;
        try(Session session = factory.openSession()) {
            session.beginTransaction();
            list = session.createQuery("from Employee").list();
            session.getTransaction().commit();
        }
        return list;
    }
    public List getList(String search) {
        List<Employee> list = null;
        try(Session session = factory.openSession()) {
            session.beginTransaction();
            Query query = session.createQuery("from Employee e " +
                    "where concat(e.surname, ' ', e.name)" +
                    "   like concat('%', '" + search + "', '%') ");
            list =  query.list();
            session.getTransaction().commit();
        }
        return list;
    }
}
```

Метод getList() возвращает все строки в таблице Employee. Сохраняет их в динамическую структуру данных List и возвращает его.

Метод getList(String search) получает строку и вставляет её в запрос к базе данных:

```
            Query query = session.createQuery("from Employee e " +
                    "where concat(e.surname, ' ', e.name)" +
                    "   like concat('%', '" + search + "', '%') ");
```

следующие строки сохраняются в List и возвращаются.

Таким образом мы сможем получить только тех работников, имя или фамилия которых были введены нами в специальное поле.

6. Создадим класс «EmployeeEditor» в директории «components»:

```
package org.company.components;

import com.vaadin.flow.component.Key;
import com.vaadin.flow.component.KeyNotifier;
import com.vaadin.flow.component.button.Button;
import com.vaadin.flow.component.icon.VaadinIcon;
import com.vaadin.flow.component.orderedlayout.HorizontalLayout;
import com.vaadin.flow.component.orderedlayout.VerticalLayout;
import com.vaadin.flow.component.textfield.TextField;
import com.vaadin.flow.data.binder.Binder;
import com.vaadin.flow.data.converter.StringToIntegerConverter;
import org.company.DAO.EmployeeDAO;
import org.company.database.Employee;

public class EmployeeEditor extends VerticalLayout implements KeyNotifier {
    private EmployeeDAO employeeDAO = null;

    private Employee employee;

    private  final TextField IdEmp = new TextField("Id emp");
    private final TextField firstName = new TextField("First name");
    private final TextField lastName = new TextField("Last name");
    private final TextField email = new TextField("Email");
    private final TextField salary = new TextField("Salary");

    private final Button save = new Button("Save", VaadinIcon.CHECK.create());
    private final Button delete = new Button("Delete", VaadinIcon.TRASH.create());
    private final HorizontalLayout actions = new HorizontalLayout(save, delete);

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

    private ChangerHandler changerHandler;

    public void setChangerHandler(ChangerHandler changerHandler) {
        this.changerHandler = changerHandler;
    }

    public interface ChangerHandler {
        void onChange();
    }
    public EmployeeEditor(EmployeeDAO employeeDAO) {
        this.employeeDAO = employeeDAO;

        add(IdEmp, lastName, firstName, email, salary, actions);

        binder.forField(IdEmp).withConverter(
                new StringToIntegerConverter("Please enter a number")
        ).bind("idEmp");
        binder.bind(firstName, Employee::getName, Employee::setName);
        binder.bind(lastName, Employee::getSurname, Employee::setSurname);
        binder.bind(email, Employee::getEmail, Employee::setEmail);
```

```
        binder.forField(salary).withConverter(
                new StringToIntegerConverter("Please enter a number")
        ).bind("salary");

       //binder.bindInstanceFields(this);

        setSpacing(true);

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

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

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

        setVisible(false);
    }
    public void save() {
        employeeDAO.create(employee);
        changerHandler.onChange();
    }
    public void delete() {
        employeeDAO.delete(employee);
        changerHandler.onChange();
    }
    public  void editEmployee(Employee emp) {
        if(emp == null) {
            setVisible(false);
            return;
        }
        if(emp.getIdEmp() != 0) {
            employee = employeeDAO.read(emp.getIdEmp());
            if(this.employee == null) {
                employeeDAO.create(emp);
                employee = employeeDAO.read(emp.getIdEmp());
            }
        }
        else {
            employee = emp;
        }
        binder.setBean(employee);
        setVisible(true);
        lastName.focus();
    }
}
```

VerticalLayout — выставляет данные в виде строки.

```
public class EmployeeEditor extends VerticalLayout implements KeyNotifier
```

и интерфейс

```
 implements KeyNotifier
```

который позволит нам отслеживать нажатие клавиш.

EmployeeDAO — для работы с классом Employee.

```
EmployeeDAO employeeDAO = null;
```

Наш объект Employee

```
private Employee employee;
```

Пять текстовых полей для ввода нового рабочего или изминения существующего

```
    private final TextField IdEmp = new TextField("Id emp");
    private final TextField firstName = new TextField("First name");
    private final TextField lastName = new TextField("Last name");
    private final TextField email = new TextField("Email");
    private final TextField salary = new TextField("Salary");
```

Две кнопки:

1. Сохранить
2. Удалить

```
    private final Button save = new Button("Save", VaadinIcon.CHECK.create());
    private final Button delete = new Button("Delete", VaadinIcon.TRASH.create());
```

Horizontal Layoult

```
    private final HorizontalLayout actions = new HorizontalLayout(save, delete);
```

Для отображение кнопок горизонтально.

С помощью baiden привязываю текущию форму к нашему пользователю, так, пользователь будет привязан к текстовым полям.

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

Vaadin будет перекладывать все изменения из класса Employee в текстовые поля, и на оборот.

Добавляю все текстовые поля и кнопки на страницу

```
add(IdEmp, lastName, firstName, email, salary, actions);
```

Связваю текстовые поля с полями класса Employee

```
binder.forField(IdEmp).withConverter(
                new StringToIntegerConverter("Please enter a number")
        ).bind("idEmp");
        binder.bind(firstName, Employee::getName, Employee::setName);
        binder.bind(lastName, Employee::getSurname, Employee::setSurname);
        binder.bind(email, Employee::getEmail, Employee::setEmail);
        binder.forField(salary).withConverter(
                new StringToIntegerConverter("Please enter a number")
        ).bind("salary");
```

По умолчанию текстовое поле возвращает объект String, но так как в классе Employee есть поля разных типов. Я явно приважу их к сответстующим типам:

```
binder.forField(salary).withConverter(
                new StringToIntegerConverter("Please enter a number")
        ).bind("salary");
```

Добавляет интервалы между текстовыми полями:

```
setSpacing(true);
```

Беру кнопки(getElement()), получаю их тему (getThemeList()) и добаляю маркеры (add(«primary») и add(«error»))

```
 save.getElement().getThemeList().add("primaty");
 delete.getElement().getThemeList().add("error");
```

Это изменит их внешний вид.

Устанавливаем Listener, с помощью которого интерфейс KeyNotifier может слушать события происходящие в этом элементе.

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

Указываем, что хотим слушать нажатие клавиши Enter и отпровлять данные в метод save().

Обработчики на нажатие кнопок:

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

Делает форму невидимой

```
setVisible(false);
```

Через класс управления классом Employee сохраняем объект:

```
public void save() {
        employeeDAO.create(employee);
        changerHandler.onChange();
    }
```

Удаляем объект:

```
public void delete() {
        employeeDAO.delete(employee);
        changerHandler.onChange();
    }
```

Редактируем работника:

```
public  void editEmployee(Employee emp) {
        if(emp == null) {
            setVisible(false);
            return;
        }
        if(emp.getIdEmp() != 0) {
            employee = employeeDAO.read(emp.getIdEmp()); // проверяем если ли уже такой
работник
            if(this.employee == null) {
                employeeDAO.create(emp);      // если нет, мы его создаём
                employee = employeeDAO.read(emp.getIdEmp()); // и запрашиваем
            }
        }
        else {
            employee = emp;      // иначе передаём его объкту в классе
        }
        binder.setBean(employee); // передаём данные полей объекта в тестовые поля на
странице
        setVisible(true);
        lastName.focus(); // установка фокуса в первое поле
    }
```

7. Создадим страницу, которая будет отображать всех наших работников. Для этого создадим файл «MainView»:

```
package org.company;

import com.vaadin.flow.component.button.Button;
import com.vaadin.flow.component.grid.Grid;
import com.vaadin.flow.component.orderedlayout.HorizontalLayout;
import com.vaadin.flow.component.orderedlayout.VerticalLayout;
import com.vaadin.flow.component.textfield.TextField;
import com.vaadin.flow.data.value.ValueChangeMode;
import com.vaadin.flow.router.Route;
import org.company.DAO.EmployeeDAO;
import org.company.components.EmployeeEditor;
import org.company.database.Employee;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

/**
 * The main view contains a button and a click listener.
 */
@Route
public class MainView extends VerticalLayout {

    private final Grid<Employee> grid = new Grid<>(Employee.class, false);

    SessionFactory factory = new Configuration().configure().buildSessionFactory();

    EmployeeDAO employeeDAO = new EmployeeDAO(factory);

    private final TextField filder = new TextField("", "seach");
    private final Button addNewBtn = new Button("Add new");
    private final EmployeeEditor editor  = new EmployeeEditor(employeeDAO);
    private final HorizontalLayout toolbar = new HorizontalLayout(filder, addNewBtn);
    public MainView() {

        grid.addColumn(Employee::getIdEmp).setHeader("Id");
        grid.addColumn(Employee::getName).setHeader("First name");
        grid.addColumn(Employee::getSurname).setHeader("Last name");
        grid.addColumn(Employee::getEmail).setHeader("Date emp");
        grid.addColumn(Employee::getSalary).setHeader("Salary");

        add(toolbar, grid, editor);

        filder.setValueChangeMode(ValueChangeMode.EAGER);
        filder.addValueChangeListener(e -> printEmployee(e.getValue()));

        grid.asSingleSelect().addValueChangeListener(e -> {
            editor.editEmployee(e.getValue());
        });

        addNewBtn.addClickListener((e -> editor.editEmployee(new Employee())));

        editor.setChangerHandler(() -> {
            editor.setVisible(false);
            printEmployee(filder.getValue());
        });

        printEmployee("");
    }
```

```
    private void printEmployee(String search) {
        if(search .isEmpty()) {
            grid.setItems(employeeDAO.getList());
        }
        else {
            grid.setItems(employeeDAO.getList(search));
        }
    }
}
```

В Vaadin все объекты являются визуальными компонентами, мы можем их отображать как страницы, просто навесив на них аннотацию Route

VerticalLayout — выставляет данные в виде строки.
```
public class MainView extends VerticalLayout
```

Grid — представляет данные в виде таблицы, необходимо указать, что будет хранить таблица и как она будет это отображать.
```
private final Grid<Employee> grid = new Grid<>(Employee.class, false);
```

SessionFactory — для подключение через hibernate к базе данных.
```
SessionFactory factory = new Configuration().configure().buildSessionFactory();
```

EmployeeDAO — для работы с классом Employee, принимает объект factory.
```
EmployeeDAO employeeDAO = new EmployeeDAO(factory);
```

Текстовое поле выступает фильтром данных.
```
private final TextField filder = new TextField("", "seach");
```

Кнопка для добавления нового пользователя.
```
private final Button addNewBtn = new Button("Add new");
```

Класс для взаимодействия с объектом Employee на странице браузера.
```
private final EmployeeEditor editor  = new EmployeeEditor(employeeDAO);
```

Объединяет поле и кнопку в одну форму.
```
private final HorizontalLayout toolbar = new HorizontalLayout(filder, addNewBtn);
```

Указываем, какие столбцы мы будет отображать в таблице и откуда мы их берём.
```
        grid.addColumn(Employee::getIdEmp).setHeader("Id");
        grid.addColumn(Employee::getName).setHeader("First name");
        grid.addColumn(Employee::getSurname).setHeader("Last name");
        grid.addColumn(Employee::getEmail).setHeader("Date emp");
        grid.addColumn(Employee::getSalary).setHeader("Salary");
```

Добавляем наши формы на страницу.
```
 add(toolbar, grid, editor);
```

Обработчии событий. После ввода каждого символа (*ValueChangeMode.EAGER)*
```
filder.setValueChangeMode(ValueChangeMode.EAGER);
```
фильтр будет отрабатывать событие printEmployee
```
filder.addValueChangeListener(e -> printEmployee(e.getValue()));
```

grid.asSingleSelect().addValueChangeListener — выбираем одну строку и вызываем редактор на редактирование выбранной строчки.

```
grid.asSingleSelect().addValueChangeListener(e -> {
            editor.editEmployee(e.getValue());
        });
```

Передаём нового Employee в объект editEmployee для редактирования

```
addNewBtn.addClickListener((e -> editor.editEmployee(new Employee())));
```

editor.setChangerHandler что у нас будет отрабатывать, когда будем в займодействовать с кнопками из класса EmployeeEditor
editor.setVisible(false); - прятать editor
вызывать метод printEmployee с значением фильтра.

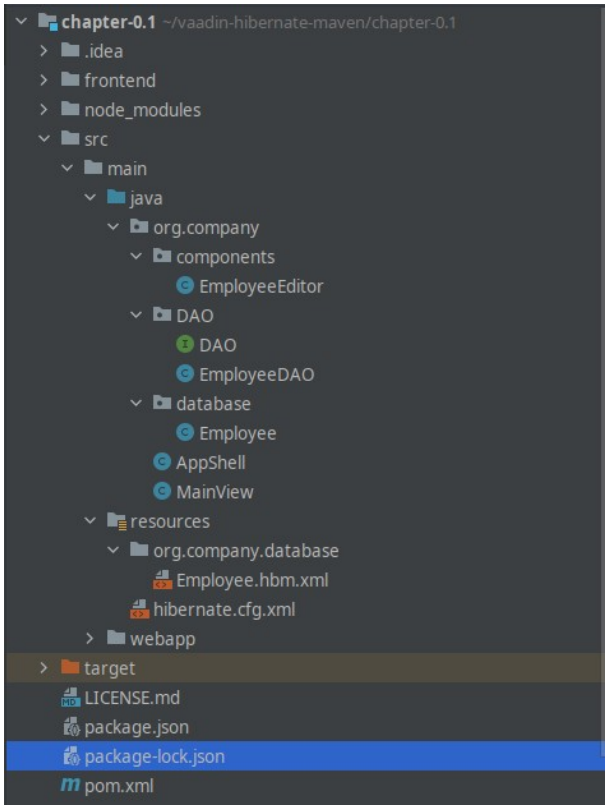
```
editor.setChangerHandler(() -> {
            editor.setVisible(false);
            printEmployee(filder.getValue());
        });
```

Отображение списка

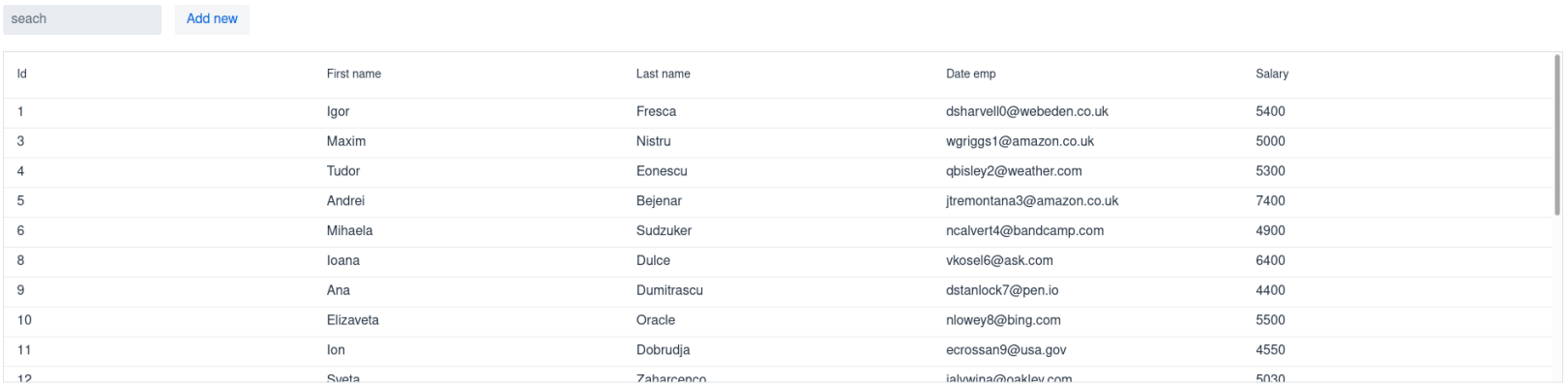
```
private void printEmployee(String search) {
        if(search .isEmpty()) {
            grid.setItems(employeeDAO.getList());
        }
        else {
            grid.setItems(employeeDAO.getList(search));
        }
    }
```

Grid.setItems — добавляет данные в таблицу.

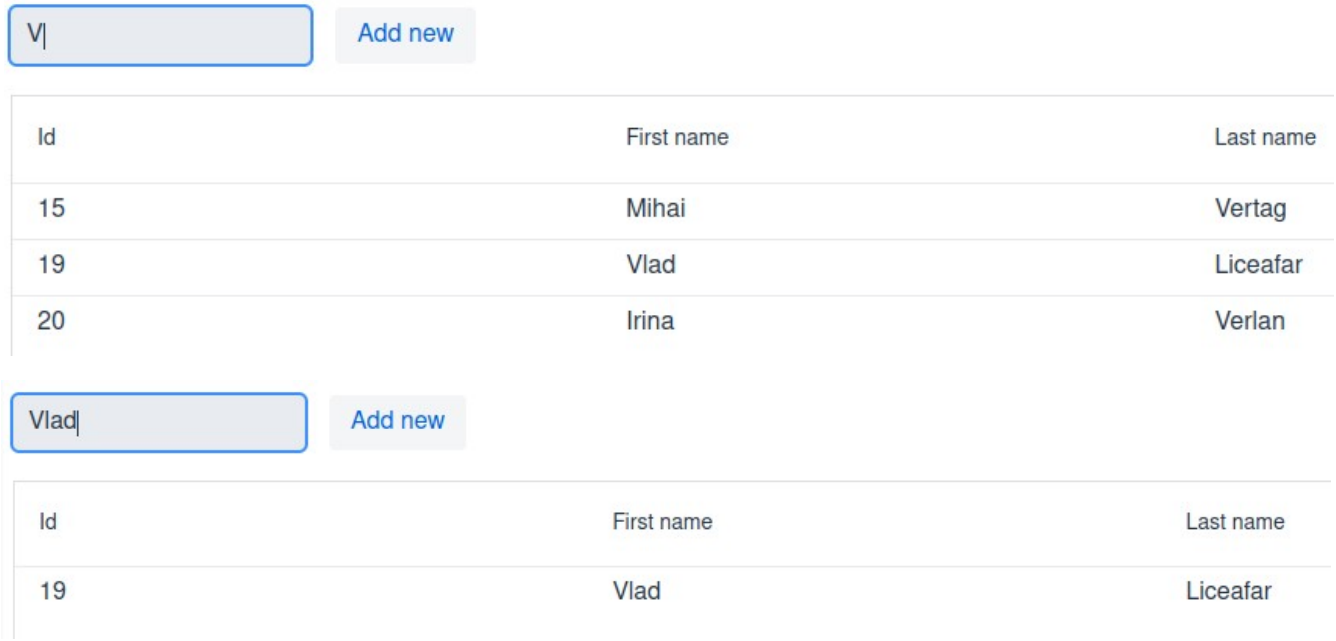
8. Структура файловой системы проекта:



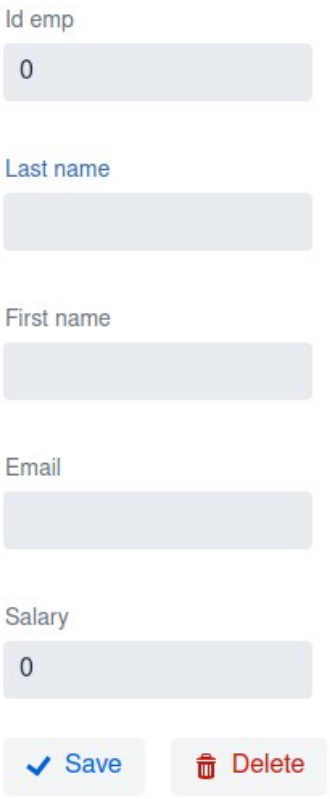
8. Запускаем приложение.
Для этого открываем терминал Idea в нижнем меню и прописываем команду:
mvn jetty:run
Открываем браузер и прописываем порт:
http://localhost:8080/



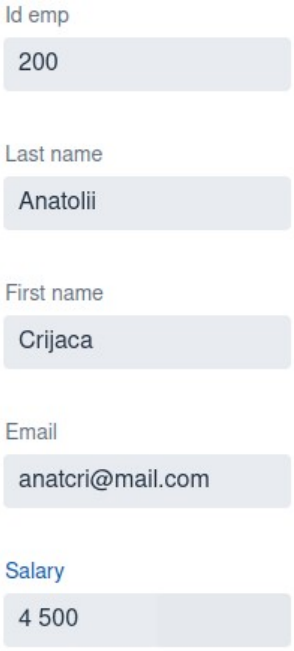
Попробуем найти сотрудника с именем «Vlad Liceafar»:
После ввода каждой буквы данные сортируются.



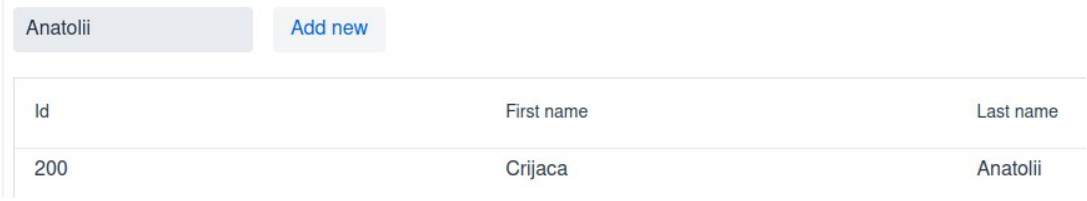
Нажмём на кнопку «Add new»



Под таблицей появилась форма с текстовыми полями.
Введём в неё новые данные:

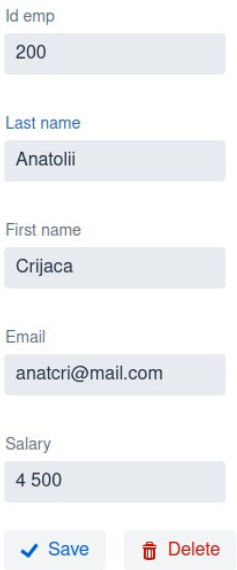


Сохраним данные и найдём их в таблице:

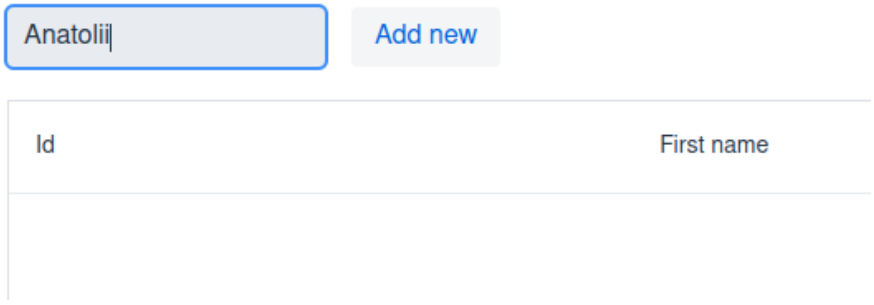


Теперь удалим введённые ранее данные:

Для этого мы нажимаем на строку с работником и его данные переходят в форму под таблицей:



Нажимаем «Delete» и пытаемся найти сотрудника:



Данные о таком сотруднике отсутствуют.