22. Spring Framework (CRUD)

- Реализация create, read, update, delete методов с помощью JDBC template

- Конфигурирование БД в среде PostgreSQL

- Валидация данных

- Использование Thymeleaf

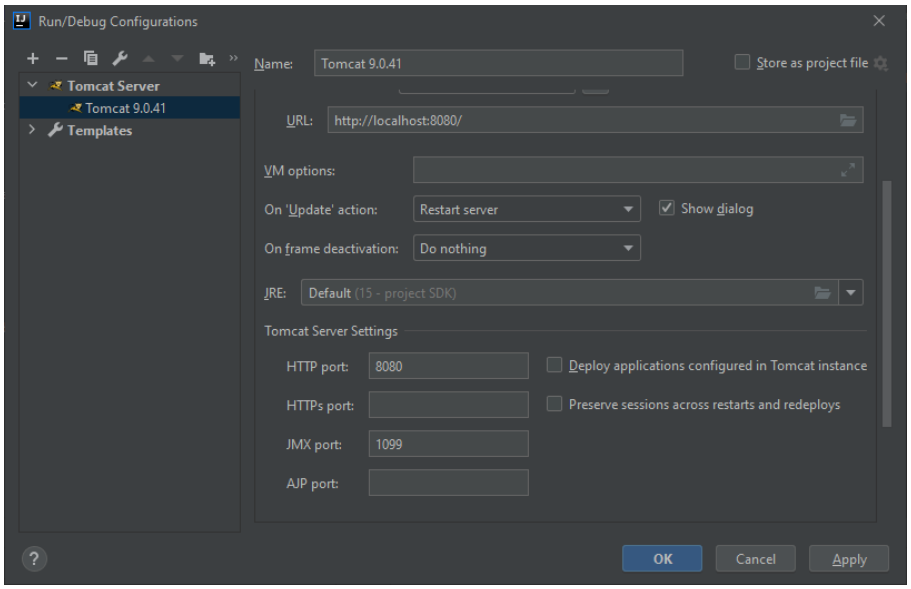
Сценарий: Создание структуры БД в среде PostgreSQL, модификация данных средствами JDBCTemplate, реализация CRUD методов, валидация данных, сборка веб-страниц шаблонизатором Thymeleaf.

Pom.xml (добавление зависимостей):

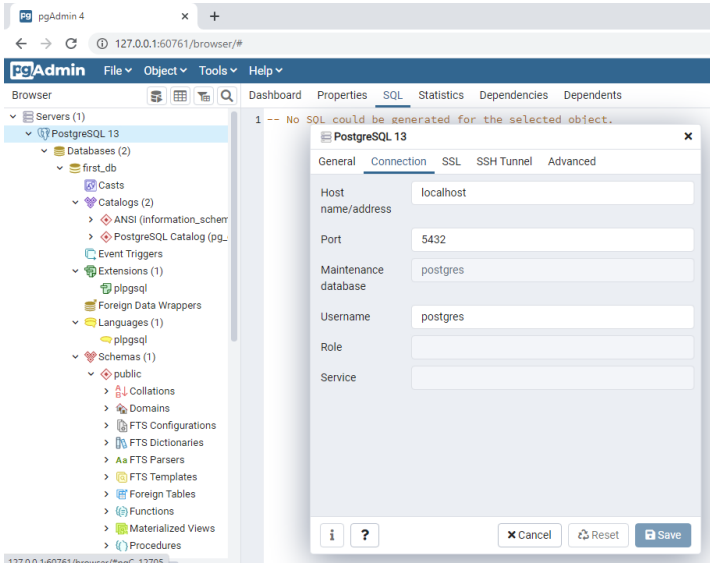
```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
```

```
        <version>${spring.version}</version>
    </dependency>

    <dependency>
        <groupId>org.thymeleaf</groupId>
        <artifactId>thymeleaf-spring5</artifactId>
        <version>3.0.11.RELEASE</version>
    </dependency>

    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
    </dependency>

    <dependency>
        <groupId>org.hibernate.validator</groupId>
        <artifactId>hibernate-validator</artifactId>
        <version>6.2.0.Final</version>
    </dependency>

    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>42.2.18</version>
    </dependency>

</dependencies>
```

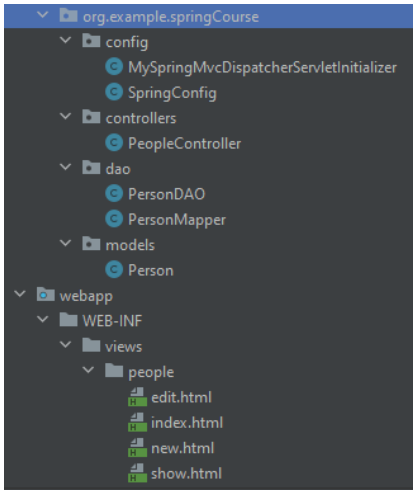
Добавление tomcat-сервера:



Создание соединения в среде PostgreSQL:



Структура проекта:



config.DispatcherServletInitializer (конфигурация DispatcherServlet):

```
public class MySpringMvcDispatcherServletInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[]{SpringConfig.class};
    }
```

```
    @Override
    protected String[] getServletMappings() {
        return new String[]{"/"};
    }

    @Override
    public void onStartup(ServletContext aServletContext) throws
ServletException {
        super.onStartup(aServletContext);
        registerHiddenFieldFilter(aServletContext);
    }

    private void registerHiddenFieldFilter(ServletContext aContext) {
        aContext.addFilter("hiddenHttpMethodFilter",
                new HiddenHttpMethodFilter()).addMappingForUrlPatterns(null,
true, "/*");
    }
}
```

config.SpringConfig (основной класс конфигурации, создание контекста, генерация бинов DataSource, JdbcTemplate, TemplateEngine, TemplateResolver, конфигурация ViewResolver):

```
@Configuration
@ComponentScan("org.example.springCourse")
@EnableWebMvc
public class SpringConfig implements WebMvcConfigurer {
    private final ApplicationContext applicationContext;

    @Autowired
    public SpringConfig(ApplicationContext applicationContext) {
        this.applicationContext = applicationContext;
    }

    @Bean
    public SpringResourceTemplateResolver templateResolver() {
        SpringResourceTemplateResolver templateResolver = new
SpringResourceTemplateResolver();
        templateResolver.setApplicationContext(applicationContext);
        templateResolver.setPrefix("/WEB-INF/views/");
        templateResolver.setSuffix(".html");
        return templateResolver;
    }

    @Bean
    public SpringTemplateEngine templateEngine() {
        SpringTemplateEngine templateEngine = new SpringTemplateEngine();
        templateEngine.setTemplateResolver(templateResolver());
        templateEngine.setEnableSpringELCompiler(true);
        return templateEngine;
    }

    @Override
    public void configureViewResolvers(ViewResolverRegistry registry) {
        ThymeleafViewResolver resolver = new ThymeleafViewResolver();
        resolver.setTemplateEngine(templateEngine());
        registry.viewResolver(resolver);
    }
```

```
    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("org.postgresql.Driver");
        dataSource.setUrl("jdbc:postgresql://localhost:5432/first_db");
        dataSource.setUsername("postgres");
        dataSource.setPassword("postgres");

        return dataSource;
    }

    @Bean
    public JdbcTemplate jdbcTemplate() {
        return new JdbcTemplate(dataSource());
    }
}
```

models.Person (связь с таблицей Person из БД, и валидация полей, @NotEmpty, @Size, @Email):

```
public class Person {
    private int id;

    @NotEmpty(message = "Name should not be empty")
    @Size(min = 2, max = 30, message = "Name should be between 2 and 3
characters")
    private String name;
    @Min(value = 0, message = "Age should be greater than 0")
    private int age;

    @NotEmpty(message = "Email should not be empty")
    @Email(message = "Email should be valid")
    private String email;
```

dao.PersonDAO (реализация методов Read, Create, Update, Delete):

```
@Component
public class PersonDAO {
    private final JdbcTemplate jdbcTemplate;

    @Autowired
    public PersonDAO(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public List<Person> index() {
        return jdbcTemplate.query("SELECT * from Person", new
BeanPropertyRowMapper<>(Person.class));
    }


    public Person show(int id) {
        return jdbcTemplate.query("SELECT * from Person WHERE id=?",
                new Object[]{id}, new BeanPropertyRowMapper<>(Person.class))
                .stream().findAny().orElse(null);
    }
```

```
    public void save(Person person) {
        jdbcTemplate.update("INSERT INTO Person VALUES (1, ?, ?, ?)",
                person.getName(),
                person.getAge(),
                person.getEmail());

    }

    public void update(int id, Person updatedPerson) {
        jdbcTemplate.update("UPDATE Person SET name=?, age=?, email=? WHERE
id=?", updatedPerson.getName(),
                updatedPerson.getAge(), updatedPerson.getEmail(), id);
    }

    public void delete(int id) {
        jdbcTemplate.update("DELETE from Person WHERE id=?", id);
    }
}
```

dao.PersonMapper (чтение полей из БД и присвоение их объекту Person, интерфейс RowMapper, ResultSet):

```
public class PersonMapper implements RowMapper<Person> {
    @Override
    public Person mapRow(ResultSet resultSet, int i) throws SQLException {
        Person person = new Person();
        person.setId(resultSet.getInt("id"));
        person.setName(resultSet.getString("name"));
        person.setAge(resultSet.getInt("age"));
        person.setEmail(resultSet.getString("email"));
        return person;
    }
}
```

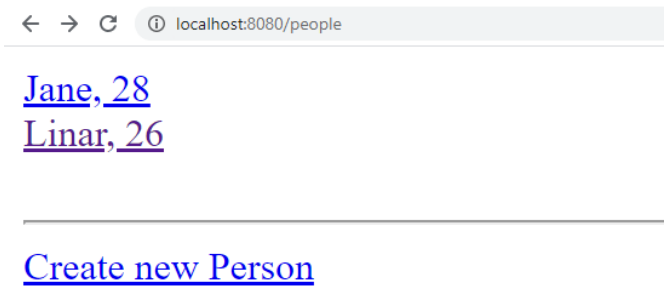
controllers.PeopleController (связывание маппингов с объектами view, @PostMapping, @GetMapping, @PatchMapping, @DeleteMapping, @PathVariable, BindingResult):

```
@Controller
@RequestMapping("/people")
public class PeopleController {

    private final PersonDAO personDAO;

    @Autowired
    public PeopleController(PersonDAO personDAO) {
        this.personDAO = personDAO;
    }

    @GetMapping()
    public String index(Model model) {
        model.addAttribute("people", personDAO.index());
        return "people/index";
    }

    @GetMapping("/{id}")
    public String show(@PathVariable("id") int id, Model model) {
```

```
        model.addAttribute("person", personDAO.show(id));
        return "people/show";

    }

    @GetMapping("/new")
    public String newPerson(@ModelAttribute("person") Person person) {
         return "people/new";
    }

    @PostMapping()
    public String create(@ModelAttribute("person") @Valid Person person,
BindingResult bindingResult) {
        if (bindingResult.hasErrors())
            return "people/new";
        personDAO.save(person);
        return "redirect:/people";
    }

    @GetMapping("/{id}/edit")
    public String edit(Model model, @PathVariable("id") int id) {
        model.addAttribute("person", personDAO.show(id));
        return "people/edit";

    }

    @PatchMapping("/{id}")
    public String update(@ModelAttribute("person") @Valid Person person,
BindingResult bindingResult, @PathVariable("id") int id) {
        if (bindingResult.hasErrors())
            return "people/edit";
        personDAO.update(id, person);
        return "redirect:/people";
    }

    @DeleteMapping("/{id}")
    public String delete(@PathVariable("id") int id) {
        personDAO.delete(id);
        return "redirect:/people";
    }
}
```

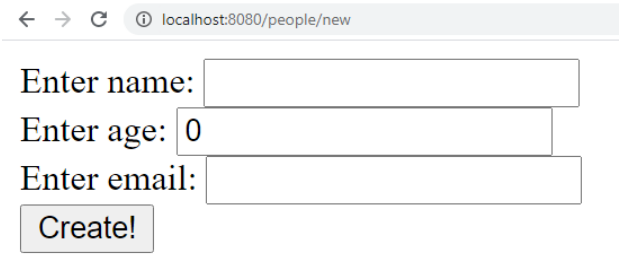
views.index.html (отображение всех пользователей):

```
<head>
    <meta charset="UTF-8">
    <title>Все люди</title>
</head>
<body>

<div th:each="person : ${people}">
    <a th:href="@{/people/{id}(id=${person.getId()})}"
       th:text="${person.getName() + ', ' + person.getAge()}">user</a>
</div>

<br/>
<hr/>
<a href="/people/new">Create new Person</a>

</body>
```



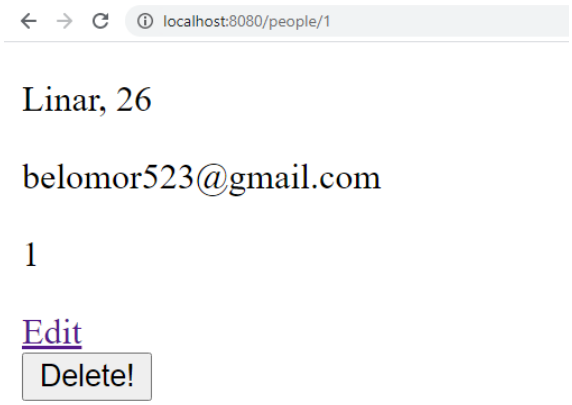
views.new.html (страница добавления пользователя):

```
<body>

<form th:method="POST" th:action="@{/people}" th:object="${person}">
    <label for="name">Enter name:</label>
    <input type="text" th:field="*{name}" id="name"/>

    <div style="color:red" th:if="${#fields.hasErrors('name')}"
th:errors="*{name}">Name error</div>

    <br/>

    <label for="age">Enter age:</label>
    <input type="text" th:field="*{age}" id="age"/>

    <div style="color:red" th:if="${#fields.hasErrors('age')}"
th:errors="*{age}">Name error</div>


    <br/>
    <label for="email">Enter email:</label>
    <input type="text" th:field="*{email}" id="email"/>
    <div style="color:red" th:if="${#fields.hasErrors('email')}"
th:errors="*{email}">Name error</div>

    <br/>

    <input type="submit" value="Create!">

</form>

</body>
```



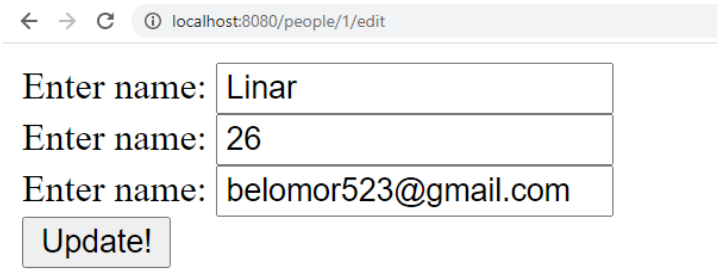
views.show.html (отображение данных о пользователе):

```
<body>

<p th:text="${person.getName() + ', ' + person.getAge()}">VALUE</p>
<p th:text="${person.getEmail()}">VALUE</p>
<p th:text="${person.getId()}">VALUE</p>

<a th:href="@{/people/{id}/edit(id=${person.getId()})}">Edit</a>

<form th:method="DELETE" th:action="@{/people/{id}(id=${person.getId()})}">
    <input type="submit" value="Delete!"/>

</form>
</body>
```



views.edit.html (форма редактирования данных):

```
<body>
<form th:method="PATCH" th:action="@{/people/{id}(id=${person.getId()})}"
th:object="${person}">
    <label for="name">Enter name:</label>
    <input type="text" th:field="*{name}" id="name"/>
    <div style="color:red" th:if="${#fields.hasErrors('name')}"
th:errors="*{name}">Name error</div>
    <br/>

    <label for="age">Enter name:</label>
    <input type="text" th:field="*{age}" id="age"/>

    <div style="color:red" th:if="${#fields.hasErrors('age')}"
th:errors="*{age}">Name error</div>

    <br/>
    <label for="email">Enter name:</label>
    <input type="text" th:field="*{email}" id="email"/>

    <div style="color:red" th:if="${#fields.hasErrors('email')}"
th:errors="*{email}">Name error</div>

    <br/>
    <input type="submit" value="Update!">


</form>
</body>
```


localhost:8080/people/1/edit
Enter name: Linar
Enter name: 26
Enter name: belomor523@gmail.com
Update!