Створення програмної структури з урахуванням спеціалізованого Фреймворку

4.3.1Spring MVC:

Фреймворк Spring MVC забезпечує архітектуру паттерна Model - View - Controller (Модель - Відображення - Контролер) за допомогою слабо пов'язаних готових компонентів. Патерн MVC розділяє аспекти додатки (логіку введення, бізнес-логіку і логіку UI), забезпечуючи при цьому вільну зв'язок між ними.

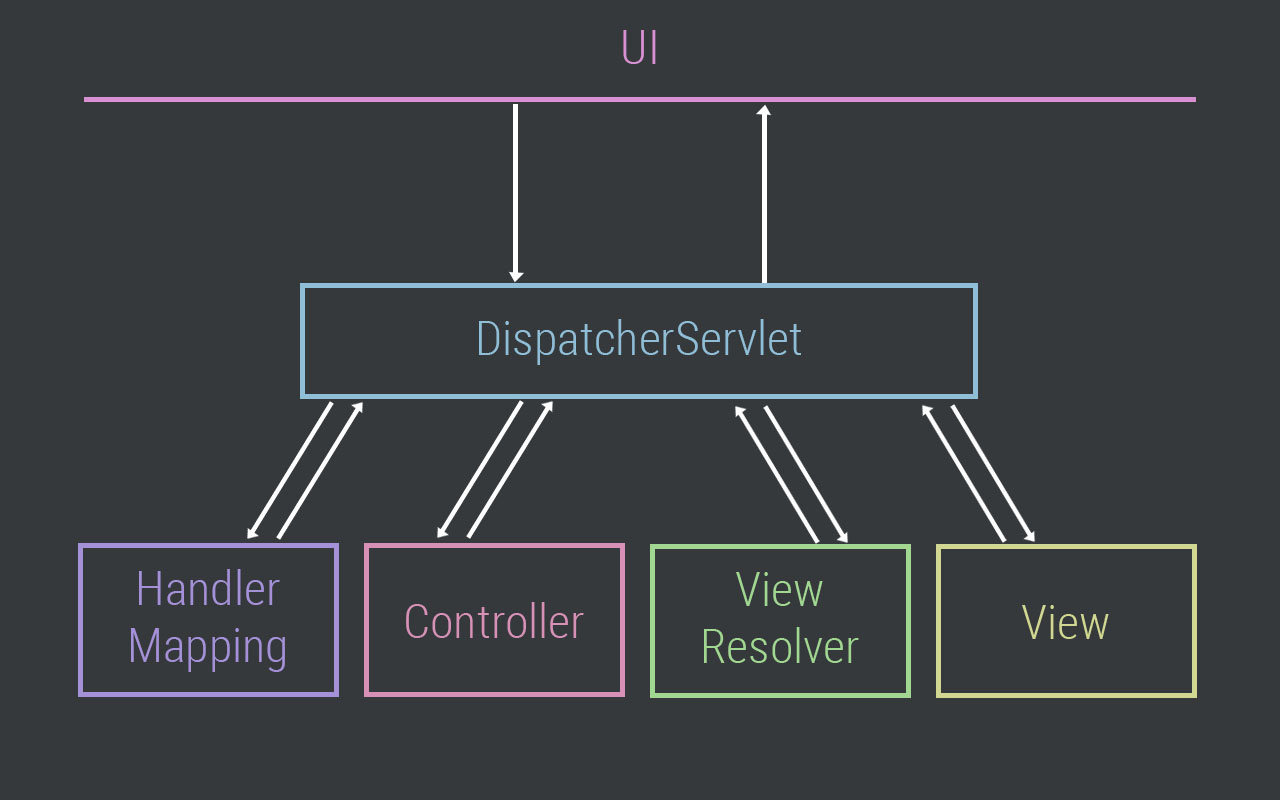
****

Рис.4.3.1.1 архітектуру паттерна MVC

1. **Model** (Модель) инкапсулирует (объединяет) данные приложения, в целом они будут состоять из POJO («Старых добрых Java-объектов», или бинов).
2. **View** (Отображение, Вид) отвечает за отображение данных Модели, — как правило, генерируя HTML, которые мы видим в своём браузере.
3. **Controller** (Контроллер) обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид.

**Структура проекту**

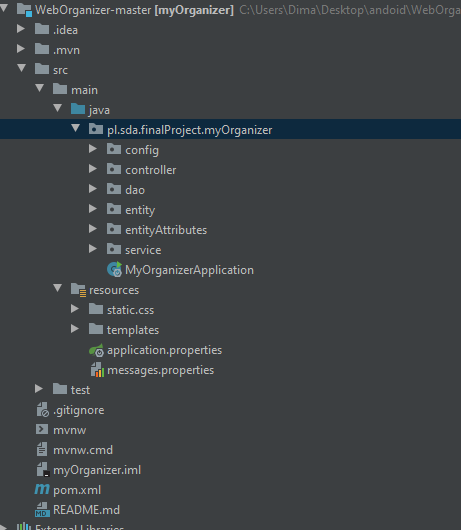
****

Рис.4.3.1.2 Структура проекту

Папка «dao», «entity» , «service» зберігає в собі класи , які потрібні для взаємодії з базою даних. Детальніше буде в наступних завданнях.

Entity – всі класи нашої моделі MVC.

Controller – всі класи контролерів MVC.

Templates – папка з html файлами (Вид MVC)

Файл application.properties - файл, де ми можемо вказати різні види властивостей (наприклад, параметри бази даних, чи параметри запуску серверу, чи параметри використання бібліотеки для авторизації та інше)

Maven автоматично завантажить бібліотеки залежностей Spring і помістить їх в локальний репозиторій Maven . Pom.xml – файл в якому, ми прописуємо потрібні бібліотеки.

Style.cc – папка з файлами css для стилізації вашої веб-сторінки.

4.3.2. Створення програмних класів

Модель event:

package pl.sda.finalProject.myOrganizer.model;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import org.hibernate.validator.constraints.NotEmpty;

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

import pl.sda.finalProject.myOrganizer.entity.Event;

import pl.sda.finalProject.myOrganizer.entity.MyUser;

import pl.sda.finalProject.myOrganizer.service.EventService;

import javax.validation.constraints.Max;

import javax.validation.constraints.Min;

import java.time.LocalDate;

import java.time.LocalTime;

import java.time.format.DateTimeFormatter;

@Data

@NoArgsConstructor

@AllArgsConstructor

public class EventModel {

private Long id;

@NotEmpty

private String name;

private LocalDate creationDate;

@NotEmpty

private String stringEventDate;

private LocalDate eventDate;

Продовження моделі event:

private String stringEventTime;

private LocalTime eventTime;

@Min(0L)

@Max(59L)

private int minutesBefore;

@Min(0L)

@Max(23L)

private int hoursBefore;

@Min(0L)

@Max(366L)

private int daysBefore;

private MyUser user;

public EventModel(Event eventEntity){

this.id = eventEntity.getId();

this.name = eventEntity.getName();

this.creationDate = eventEntity.getCreationDate();

this.eventDate = eventEntity.getEventDate();

this.eventTime = eventEntity.getEventTime();

this.minutesBefore = eventEntity.getMinutesBefore();

this.hoursBefore = eventEntity.getHoursBefore();

this.daysBefore = eventEntity.getDaysBefore();

this.user = eventEntity.getUser();

}

}

Таблиця 4.3.2.1. Опис анотації:

|  |  |
| --- | --- |
| Анотація | Опис |
| @Controller | @Controller - (Слой представления) Аннотация для маркировки java класса, как класса контроллера. Данный класс представляет собой компонент, похожий на обычный сервлет (HttpServlet) (работающий с объектами HttpServletRequest и HttpServletResponse), но с расширенными возможностями от Spring Framework. |

Продовження таблиці 4.3.2.1. Опис анотації:

|  |  |
| --- | --- |
| @Autowired | Аннотация @Autowired обеспечивает контроль над тем, где и как автосвязывание должны быть осуществлено. Мы можем использовать @Autowired как для методов, так и для конструкторов |
| @GetMapping | Аннотация @GetMapping говорит — метод list() должен быть вызван, когда кто-то вызывает метод GET на пути /notes. Имя пути, очевидно, берётся из параметра @RequestMapping на классе. |
| @PostMapping | Аннотация @PostMapping без спецификации пути, говорит нам, что метод create() обслуживает POST запросы по пути /notes. |

Зразок коду контроллер event:

@Controller

public class EventController {

@Autowired

private IEventRepository eventRepository;

@Autowired

private IUserRepository userRepository;

@Autowired

private EventService eventService;

@GetMapping("/organizer/events")

public String showEventsPage(Model model, Principal principal) {

EventModel newEvent = new EventModel();

MyUser activeUser = userRepository.findOne(principal.getName());

model.addAttribute("events", eventService.getCurrentEvents(activeUser));

model.addAttribute("newEvent", newEvent);

model.addAttribute("today", LocalDate.*now*());

model.addAttribute("eventsToRemind", eventService.getEventsToRemind(activeUser));

return "events";

}

Продовження зразка коду контролеру event:

@PostMapping(path = "organizer/events")

public String addEvent(@Valid @ModelAttribute("newEvent") EventModel newEvent, BindingResult bindingResult,

Principal principal, Model model) {

if (bindingResult.hasErrors()) {

model.addAttribute("validErrors", true);

model.addAttribute("errorMsg",

"Adding the event failed, parameters name and date required!");

return "events";

}

MyUser activeUser = userRepository.findOne(principal.getName());

eventService.parseEventDateAndTime(newEvent);

Event entity = Event.*builder*()

.creationDate(LocalDate.*now*())

.user(activeUser)

.name(newEvent.getName())

.eventDate(newEvent.getEventDate())

.eventTime(newEvent.getEventTime())

.minutesBefore(newEvent.getMinutesBefore())

.hoursBefore(newEvent.getHoursBefore())

.daysBefore(newEvent.getDaysBefore())

.build();

eventRepository.save(entity);

return "redirect:/organizer/events";

}

@GetMapping(path = "/organizer/events/edit/{id}")

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

Event editEvent = eventRepository.findOne(id);

if (editEvent == null) {

return "eventNotFound";

}

4.4. Особливості розробки алгоритмів методів програмних класів

DAO (Data Access Object) - це шар об'єктів які забезпечують доступ до даних.

Зазвичай для реалізації DAO використовується EntityManager і з його допомогою ми працюємо з нашою БД, але в нашому випадку це система не підійде, так як ми вивчаємо Spring Data нам потрібно використовувати її засоби інакше нема чого він нам.

Spring Data надає набір готових реалізацій для створення DAO але Spring вважали за краще цей шар називати не DAO, а Repository.

Приклад створення Repository:

package pl.sda.finalProject.myOrganizer.dao;

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

import org.springframework.stereotype.Repository;

import pl.sda.finalProject.myOrganizer.entity.Event;

import pl.sda.finalProject.myOrganizer.entity.MyUser;

import java.util.List;

@Repository

public interface IEventRepository extends JpaRepository<Event, Long> {

List<Event> findByUserOrderByEventDateAsc(MyUser user);

List<Event> findAllByUser(MyUser user);

void deleteAllByUser(MyUser user);

}

JpaRepository - це інтерфейс фреймворка Spring Data надає набір стандартних методів JPA для роботи з БД.

Service - це Java клас, який надає з себе основну (Бізнес-Логіку). В основному сервіс використовує готові DAO / Repositories або ж інші сервіси, для того щоб надати кінцеві дані для призначеного для користувача інтерфейсу.

Приклад створення Service:

package pl.sda.finalProject.myOrganizer.service;

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

import org.springframework.stereotype.Service;

import pl.sda.finalProject.myOrganizer.dao.IEventRepository;

import pl.sda.finalProject.myOrganizer.entity.Event;

import pl.sda.finalProject.myOrganizer.entity.MyUser;

import pl.sda.finalProject.myOrganizer.model.EventModel;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.LocalTime;

import java.time.format.DateTimeFormatter;

import java.util.Iterator;

import java.util.List;

import java.util.stream.Collectors;

@Service

public class EventService {

@Autowired

private IEventRepository eventRepository;

@Autowired

private DateTimeFormatter dateFormatter;

@Autowired

private DateTimeFormatter timeFormatter;

public void parseEventDateAndTime(EventModel eventModel) {

// eventDate required

Продовження прикладу створення Service:

eventModel.setEventDate(LocalDate.*parse*(eventModel.getStringEventDate(), dateFormatter));

// optional

if (!eventModel.getStringEventTime().isEmpty()) {

eventModel.setEventTime(LocalTime.*parse*(eventModel.getStringEventTime(), timeFormatter));

}

}

public String parseEventDate(LocalDate localDate) {

return localDate.format(dateFormatter);

}

public String parseEventTime(LocalTime localTime) {

return localTime.format(timeFormatter);

}

public List<Event> getCurrentEvents(MyUser activeUser) {

List<Event> events = eventRepository.findByUserOrderByEventDateAsc(activeUser);

for (Iterator<Event> iterator = events.iterator(); iterator.hasNext(); ) {

Event event = iterator.next();

if (event.getEventDate().isBefore(LocalDate.*now*())) {

eventRepository.delete(event);

iterator.remove();

}

}

return events;

}

public List<Event> getEventsToRemind(MyUser activeUser) {

return eventRepository.findAllByUser(activeUser).stream().filter(

event -> isEventToRemind(event)

).collect(Collectors.*toList*());

}

private boolean isEventToRemind(Event event) {

if (event.getEventTime() == null) {

event.setEventTime(LocalTime.*MIDNIGHT*);

}

if (event.getDaysBefore() > 0 && event.getEventDate().

minusDays(event.getDaysBefore()).isEqual(LocalDate.*now*())) {

return true;

}

if (event.getHoursBefore() > 0 && LocalDateTime.*now*().isAfter(LocalDateTime.*of*(event.getEventDate(),

event.getEventTime()).minusHours(event.getHoursBefore()))) {

return true;

}

if (event.getMinutesBefore() > 0 && LocalDateTime.*now*().isAfter(LocalDateTime.*of*(event.getEventDate(),

event.getEventTime()).minusMinutes(event.getMinutesBefore()))) {

return true;

} else return false;

}

}

4.5. Використання спеціалізованих програмних бібліотек

За допомогою конфігураційного файлу pom.xml встановлюються бібліотеки. Все що потрібно це вказати пакети, які безпосередньо потрібні. Можна легко знайти пакети, використовуючи Google, і пошук "maven репозиторій".

Приклад файлу pom.xml:

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

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

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

<modelVersion>4.0.0</modelVersion>

<groupId>pl.sda.finalProject</groupId>

<artifactId>myOrganizer</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>myOrganizer</name>

<description>Demo project for Spring Boot</description>

// Все приложения Spring Boot конфигурируются от **spring-boot-starter-parent**

<parent>

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

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

<version>1.5.10.RELEASE</version>

<relativePath/>

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

/\* Оскільки ми створюємо REST API, то необхідно в якості залежно використовувати spring-boot-starter-web, яка неявно визначає всі інші залежності, такі як spring-core, spring-web, spring-webmvc, servlet api, і бібліотеку jackson-databind \* /

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

Продовження прикладу файла pom.xml:

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

//Підключення бібліотеки для роботи с БД postgresql

<dependency>

<groupId>org.postgresql</groupId

<artifactId>postgresql</artifactId>

<scope>runtime</scope>

</dependency>

//Підключення бібліотеки Lombook

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

//Підключення бібліотеки Security

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-test</artifactId>

<scope>test</scope

</dependency>

//Підключення бібліотеки Hibernate

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-java8</artifactId>

<version>5.1.0.Final</version>

</dependency>

//Підключення бібліотеки Security

<dependency>

<groupId>org.thymeleaf.extras</groupId>

<artifactId>thymeleaf-extras-springsecurity4</artifactId>

<version>3.0.2.RELEASE</version>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>1.4.196</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>