**Menedżer zadań – dokumentacja**

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Państwowa Akademia Nauk Stosowanych w Krośnie

Wydział Informatyki

Przedmiot: Programowanie w języku Java

Prowadzący: dr. Marcin Skuba

Temat projektu: Menedżer zadań (Spring Boot)

Autor: Konrad Płaziak

Nr indeksu: 27631

Spis treści

[Wprowadzenie 3](#_Toc200561436)

[Wykorzystane technologie 4](#_Toc200561437)

[Struktura projektu 5](#_Toc200561438)

[ViewController.java 6](#_Toc200561439)

[Task.java 7](#_Toc200561440)

[TaskConfig.java 9](#_Toc200561441)

[TaskController.java 10](#_Toc200561442)

[TaskRepository.java 12](#_Toc200561443)

[TaskService.java 13](#_Toc200561444)

[HabitTrackerApplication.java 15](#_Toc200561445)

[calendar.css 16](#_Toc200561446)

[navbar.css 18](#_Toc200561447)

[summary.css 19](#_Toc200561448)

[task.css 21](#_Toc200561449)

[calendar.js 25](#_Toc200561450)

[summary.js 28](#_Toc200561451)

[task.js 30](#_Toc200561452)

[navbar.html 33](#_Toc200561453)

[calendar.html 34](#_Toc200561454)

[summary.html 35](#_Toc200561455)

[task.html 36](#_Toc200561456)

[Konfiguracja application.properties 37](#_Toc200561457)

Wprowadzenie

Menedżer zadań to rozbudowana aplikacja typu ToDo lista, stworzona w ramach zajęć z języka Java. Jej głównym celem jest umożliwienie użytkownikowi skutecznego zarządzania zadaniami i organizowania codziennych obowiązków. Aplikacja została zrealizowana w technologii Spring Boot, z wykorzystaniem bazy danych oraz prostego interfejsu webowego z dynamiczną obsługą zadań.

**Główne funkcjonalności aplikacji:**

* Dodawanie, edytowanie i usuwanie zadań.
* Oznaczanie zadań jako wykonane.
* Przeglądanie zadań według daty – dla konkretnego dnia lub całego miesiąca.
* Dynamiczna aktualizacja listy zadań bez konieczności przeładowywania strony.
* Trwałe przechowywanie danych w bazie danych.
* Kalendarz jako wizualna reprezentacja zadań na dany miesiąc.
* Dedykowana podstrona z podsumowaniem miesiąca zawierająca statystyki zadań (np. liczba ukończonych, aktywnych, itp.).

**Projekt powstał z myślą o praktycznym przetestowaniu znajomości:**

* Architektury aplikacji webowych REST w Spring Boot,
* Integracji backendu z prostym frontendem (HTML + JavaScript),
* Pbsługi zapytań HTTP oraz operacji CRUD,
* Pracy z bazą danych (JPA/Hibernate),
* Podstawowych zagadnień związanych z UX/UI.

Aplikacja ma charakter edukacyjny, ale została zaprojektowana w sposób umożliwiający łatwe rozszerzenie jej w przyszłości, np. o konta użytkowników, powiadomienia, poziomy priorytetów czy inne funkcje.

Wykorzystane technologie

* Java 17
* Spring Boot
* Spring Web
* Spring Data JPA
* Thymeleaf
* Maven
* H2 Database(do celów testowych)
* HTML
* CSS
* JavaScript

Struktura projektu

├───main

│ ├───java

│ │ └───org

│ │ └───example

│ │ └───habittracker

│ │ │ HabitTrackerApplication.java

│ │ │

│ │ ├───home

│ │ │ ViewController.java

│ │ │

│ │ └───task

│ │ Task.java

│ │ TaskConfig.java

│ │ TaskController.java

│ │ TaskRepository.java

│ │ TaskService.java

│ │

│ └───resources

│ │ application.properties

│ │

│ ├───static

│ │ ├───css

│ │ │ calendar.css

│ │ │ navbar.css

│ │ │ summary.css

│ │ │ task.css

│ │ │

│ │ └───script

│ │ calendar.js

│ │ summary.js

│ │ task.js

│ │

│ └───templates

│ │ calendar.html

│ │ summary.html

│ │ task.html

│ │

│ └───fragments

│ navbar.html

│

└───test

└───java

└───org

└───example

└───habittracker

HabitTrackerApplicationTests.java

ViewController.java

**Opis:** Klasa odpowiadająca za przekierowywanie na odpowiednie widoki HTML aplikacji.

package org.example.habittracker.home;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

@Controller

public class ViewController {

@RequestMapping("/task")

public String tasks() {

return "task";

}

@RequestMapping("/summary")

public String summary() {

return "summary";

}

@RequestMapping("/calendar")

public String calendar() {

return "calendar";

}

}

Task.java

**Opis:** Reprezentuje pojedyncze zadanie z polami takimi jak nazwa, data i status wykonania, mapowane na encję bazy danych.

package org.example.habittracker.task;

import jakarta.persistence.\*;

import java.time.LocalDate;

@Entity

@Table(name = "tasks")

public class Task {

@Id

@SequenceGenerator(

name = "task\_sequence",

sequenceName = "task\_sequence",

allocationSize = 1

)

@GeneratedValue(

strategy = GenerationType.SEQUENCE,

generator = "task\_sequence"

)

private Long id;

private String name;

private LocalDate date;

private boolean completed;

public Task() {

}

public Task(String name, LocalDate date, long id, boolean completed) {

this.name = name;

this.date = date;

this.id = id;

this.completed = completed;

}

public Task(String name, LocalDate date) {

this.name = name;

this.date = date;

this.completed = false;

}

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public LocalDate getDate() {

return date;

}

public void setDate(LocalDate date) {

this.date = date;

}

public boolean isCompleted() {

return completed;

}

public void setCompleted(boolean completed) {

this.completed = completed;

}

@Override

public String toString() {

return "Task{" +

"id=" + id +

", name='" + name + '\'' +

", date=" + date +

", completed=" + completed +

'}';

}

}

TaskConfig.java

**Opis:** Konfiguracja Springa, która inicjalizuje bazę danych przykładowymi zadaniami przy starcie aplikacji.

package org.example.habittracker.task;

import org.springframework.boot.CommandLineRunner;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import java.time.LocalDate;

import java.time.Month;

import java.util.List;

@Configuration

public class TaskConfig {

@Bean

CommandLineRunner runner(TaskRepository taskRepository) {

return args -> {

Task gym = new Task("Siłownia", LocalDate.now());

Task java = new Task("Nauka Javy", LocalDate.now());

Task call = new Task("Zadzwonić do mamy", LocalDate.now().plusDays(-1));

call.setCompleted(true);

Task shopping = new Task("Zrobić zakupy", LocalDate.now().plusDays(1));

Task gym2 = new Task("Siłownia", LocalDate.now().plusDays(-40));

gym2.setCompleted(true);

Task dentist = new Task("Wizyta u dentysty", LocalDate.now().plusDays(24));

Task juvenalia = new Task("Juvenalia", LocalDate.of(2025, Month.MAY, 29));

Task cv = new Task("Napisać CV", LocalDate.now().plusDays(-8));

cv.setCompleted(true);

Task cleaning = new Task("Sprzątanie", LocalDate.now().plusDays(-34));

Task graphics = new Task("Dokończyć grafikę", LocalDate.now().plusDays(-18));

Task shoes = new Task("Kupić buty", LocalDate.now().plusDays(26));

Task shopping2 = new Task("Zrobić zakupy", LocalDate.now().plusDays(-9));

shopping2.setCompleted(true);

Task mail = new Task("Przeczytać maile", LocalDate.now().plusDays(-18));

mail.setCompleted(true);

Task trash = new Task("Wynieść śmieci", LocalDate.now().plusDays(-26));

trash.setCompleted(true);

Task finals = new Task("Nauka do sesji", LocalDate.of(2025, Month.JUNE, 3));

Task birthday = new Task("Urodziny", LocalDate.of(2025, Month.MARCH, 24));

birthday.setCompleted(true);

Task spring = new Task("Nauka Spring Boota", LocalDate.of(2025, Month.MARCH, 20));

spring.setCompleted(true);

taskRepository.saveAll(List.of(gym, java, call, shopping, gym2, dentist, juvenalia, cv, cleaning, graphics, shoes, shopping2, mail, trash, finals, birthday, spring));

};

}

}

TaskController.java

**Opis:** Kontroler Springa obsługujący żądania HTTP związane z zarządzaniem zadaniami, takie jak pobieranie, dodawanie, aktualizacja i usuwanie zadań.

package org.example.habittracker.task;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.time.LocalDate;  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/v1/task")  
public class TaskController {  
  
 private final TaskService taskService;  
  
 @Autowired  
 public TaskController(TaskService taskService) {  
 this.taskService = taskService;  
 }  
  
 @GetMapping  
 public List<Task> getTasks() {  
 return taskService.getTasks();  
 }  
  
 @GetMapping(path = "{date}")  
 public ResponseEntity<List<Task>> getTasksByDate(@PathVariable LocalDate date) {  
 List<Task> tasks = taskService.getTasksByDate(date);  
 if (tasks == null || tasks.isEmpty()) {  
 return ResponseEntity.*notFound*().build();  
 }  
 return ResponseEntity.*ok*(tasks);  
 }  
  
 @PostMapping  
 public void addTask(@RequestBody Task task) {  
 taskService.addTask(task);  
 }  
  
 @DeleteMapping(path = "{taskId}")  
 public void deleteTask(@PathVariable("taskId") Long TaskId) {  
 taskService.deleteTask(TaskId);  
 }  
  
 @PutMapping(path = "{taskId}")  
 public ResponseEntity<Void> updateTask(  
 @PathVariable("taskId") Long taskId,  
 @RequestParam(required = false) String name,  
 @RequestParam(required = false) String date) {  
 LocalDate parsedDate = date != null ? LocalDate.*parse*(date) : null;  
 taskService.updateTask(taskId, name, parsedDate);  
 return ResponseEntity.*ok*().build();  
 }  
  
 @PutMapping(path = "{taskId}/complete")  
 public ResponseEntity<Void> completeTask(  
 @PathVariable ("taskId") Long taskId)  
 {  
 taskService.completeTask(taskId);  
 return ResponseEntity.*ok*().build();  
 }  
  
 @GetMapping(path = "/month/{date}")  
 public ResponseEntity<List<Task>> getTasksFromMonth(@PathVariable LocalDate date) {  
 LocalDate startOfMonth = date.withDayOfMonth(1);  
 LocalDate endOfMonth = date.withDayOfMonth(date.lengthOfMonth());  
 List<Task> tasks = taskService.getTasksByDateRange(startOfMonth, endOfMonth);  
 if (tasks == null || tasks.isEmpty()) {  
 return ResponseEntity.*notFound*().build();  
 }  
 return ResponseEntity.*ok*(tasks);  
 }  
}

TaskRepository.java

**Opis:** Interfejs repozytorium Spring Data JPA zapewniający podstawowe operacje CRUD oraz metody do wyszukiwania zadań po dacie i zakresie dat.

package org.example.habittracker.task;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
import java.time.LocalDate;  
import java.util.List;  
  
@Repository  
public interface TaskRepository extends JpaRepository<Task, Long> {  
  
 List<Task> findByDate(LocalDate date);  
  
 List<Task> findByDateBetween(LocalDate start, LocalDate end);  
}

TaskService.java

**Opis:** Warstwa serwisu zawierająca logikę biznesową do zarządzania zadaniami, w tym walidację i operacje na repozytorium.

package org.example.habittracker.task;  
  
import jakarta.transaction.Transactional;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.time.LocalDate;  
import java.util.List;  
import java.util.Objects;  
  
@Service  
public class TaskService {  
  
 private final TaskRepository taskRepository;  
  
 @Autowired  
 public TaskService(TaskRepository taskRepository) {  
 this.taskRepository = taskRepository;  
 }  
  
 public List<Task> getTasks() {  
 return taskRepository.findAll();  
 }  
  
 public void addTask(Task task) {  
 taskRepository.save(task);  
 }  
  
 public void deleteTask(Long taskId) {  
 boolean exists = taskRepository.existsById(taskId);  
 if (!exists) {  
 throw new IllegalStateException("Task does not exist");  
 }  
 taskRepository.deleteById(taskId);  
 }  
  
 @Transactional  
 public void updateTask(Long taskId, String name, LocalDate date) {  
 Task task = taskRepository.findById(taskId)  
 .orElseThrow(() -> new IllegalStateException("Task does not exist"));  
  
 if(name != null && !Objects.*equals*(task.getName(), name)) {  
 task.setName(name);  
 }  
  
 if(date != null && !Objects.*equals*(task.getDate(), date)) {  
 task.setDate(date);  
 }  
 }  
  
 @Transactional  
 public void completeTask(Long taskId) {  
 Task task = taskRepository.findById(taskId)  
 .orElseThrow(() -> new IllegalStateException("Task does not exist"));  
 task.setCompleted(!task.isCompleted());  
 }  
  
 public List<Task> getTasksByDate(LocalDate date) {  
 return taskRepository.findByDate(date);  
 }  
  
 public List<Task> getTasksByDateRange(LocalDate start, LocalDate end) {  
 return taskRepository.findByDateBetween(start, end);  
 }  
}

HabitTrackerApplication.java

**Opis:** Główna klasa aplikacji Spring Boot, która uruchamia całą aplikację do śledzenia nawyków, wykorzystując automatyczną konfigurację i wbudowany serwer.

package org.example.habittracker;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class HabitTrackerApplication {

public static void main(String[] args) {

SpringApplication.run(HabitTrackerApplication.class, args);

}

}

calendar.css

**Opis:** Arkusz stylów definiujący wygląd kalendarza zadań, włączając w to układ dni, kolorystykę dla różnych statusów zadań oraz style dla bieżącego dnia.

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background-color: #f4f4f4;

color: #333;

line-height: 1.6;

display: flex;

flex-direction: column;

align-items: center;

min-height: 100vh;

}

.calendar-container {

background-color: #fff;

padding: 30px;

border-radius: 8px;

box-shadow: 0 0 15px rgba(0,0,0,0.1);

width: 100%;

max-width: 700px;

margin-top: 20px;

}

.calendar-header {

display: flex;

justify-content: space-between;

align-items: center;

margin-bottom: 30px;

}

.calendar-header h2 {

font-size: 2em;

margin: 0;

}

.calendar-header button {

background: none;

border: 1px solid #ddd;

padding: 10px 15px;

cursor: pointer;

border-radius: 4px;

font-size: 1.8em;

}

.calendar-header button:hover {

background-color: #f0f0f0;

}

.calendar-weekdays, .calendar-days {

display: grid;

grid-template-columns: repeat(7, 1fr);

gap: 8px;

}

.calendar-weekdays div {

text-align: center;

font-weight: bold;

padding-bottom: 15px;

font-size: 1.1em;

}

.calendar-days div {

text-align: center;

padding: 15px 10px;

border: 1px solid #eee;

min-height: 70px;

display: flex;

justify-content: center;

align-items: center;

border-radius: 4px;

font-size: 1.2em;

position: relative;

}

.calendar-days div .task-count {

position: absolute;

top: 5px;

right: 5px;

background-color: rgba(0, 0, 0, 0.1);

color: #333;

font-size: 0.7em;

padding: 2px 5px;

border-radius: 50%;

line-height: 1;

min-width: 18px;

min-height: 18px;

display: flex;

justify-content: center;

align-items: center;

}

.calendar-days div.empty {

background-color: #f9f9f9;

border: 1px solid #f9f9f9;

}

.calendar-days div.tasks-pending {

background-color: #ffcccc;

border-color: #ff9999;

}

.calendar-days div.tasks-completed-all {

background-color: #ccffcc;

border-color: #99ff99;

}

.calendar-days div.today {

background-color: #007bff;

color: white;

font-weight: bold;

border-color: #007bff;

}

.calendar-days div.today .task-count {

background-color: rgba(255, 255, 255, 0.3);

color: white;

}

navbar.css

**Opis:** Arkusz stylów odpowiedzialny za wygląd górnego paska nawigacyjnego, zawierający style dla linków i kontenera nawigacji.

.nav-top-bar {  
 background-color: #166088;  
 padding: 1rem 0;  
 width: 100%;  
 margin: 0 auto 35px auto;  
}  
  
.nav-bar-container {  
 max-width: 800px;  
 margin: 0 auto;  
 display: flex;  
 gap: 1.5rem;  
 padding: 0 20px;  
}  
  
.nav-bar-link {  
 font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;  
 color: white;  
 text-decoration: none;  
 font-size: 1.1rem;  
  
}  
  
.nav-bar-link:hover {  
 text-decoration: underline;  
}

summary.css

**Opis:** Arkusz stylów dla sekcji podsumowania, obejmujący układ kart statystyk, nawigację miesięczną oraz listę zadań z wyróżnieniem ich statusów.

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background-color: #f8f9fa;

color: #333;

line-height: 1.6;

}

.summary-container {

max-width: 800px;

margin: 0 auto;

background-color: white;

border-radius: 0 0 8px 8px;

box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

overflow: hidden;

}

.summary-header {

text-align: center;

margin-bottom: 25px;

color: #166088;

}

.summary-header h1 {

margin-bottom: 15px;

}

.month-navigation {

display: flex;

justify-content: center;

align-items: center;

gap: 20px;

margin-top: 15px;

}

.month-button {

background: #166088;

color: white;

border: none;

border-radius: 50%;

width: 36px;

height: 36px;

font-size: 1.2rem;

cursor: pointer;

display: flex;

align-items: center;

justify-content: center;

transition: background 0.2s;

}

.month-button:hover {

background: #0d4b6e;

}

.month-display {

font-size: 1.2rem;

font-weight: 600;

min-width: 120px;

}

.stats-grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));

gap: 15px;

margin-bottom: 30px;

}

.stat-card {

background: #f8f9fa;

border-radius: 8px;

padding: 15px;

text-align: center;

}

.stat-card.highlight {

background: #e6f2ff;

border: 1px solid #166088;

}

.stat-card h3 {

margin-top: 0;

color: #166088;

font-size: 1rem;

}

.stat-value {

font-size: 2rem;

font-weight: bold;

margin: 10px 0;

}

.tasks-summary h2 {

color: #166088;

border-bottom: 2px solid #e9ecef;

padding-bottom: 8px;

}

.tasks-list {

margin-top: 15px;

}

.task-item {

background: #f8f9fa;

padding: 12px 15px;

margin-bottom: 8px;

border-radius: 4px;

display: flex;

align-items: center;

}

.task-name {

flex-grow: 1;

}

.task-date {

color: #6c757d;

font-size: 0.9rem;

margin-right: 15px;

}

.task-status {

font-weight: bold;

color: #28a745;

}

.task-status.pending {

color: #dc3545;

}

task.css

**Opis:** Główny arkusz stylów aplikacji, definiujący kolorystykę, układ formularza dodawania zadań, listę zadań z interaktywnymi elementami oraz style dla stanów wykonania.

:root {

--primary-color: #4a6fa5;

--secondary-color: #166088;

--background-color: #f8f9fa;

--text-color: #333;

--light-gray: #e9ecef;

--medium-gray: #ced4da;

--dark-gray: #6c757d;

--success-color: #28a745;

}

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background-color: var(--background-color);

color: var(--text-color);

line-height: 1.6;

}

.app-container {

max-width: 800px;

margin: 0 auto;

background-color: white;

border-radius: 0 0 8px 8px;

box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

overflow: hidden;

}

.app-header {

background-color: var(--primary-color);

color: white;

padding: 20px;

text-align: center;

}

.app-header h1 {

margin-bottom: 15px;

font-size: 2rem;

}

.date-picker-container {

display: flex;

justify-content: center;

}

.date-picker {

padding: 8px 12px;

border: none;

border-radius: 4px;

font-size: 1rem;

background-color: white;

}

.main-content {

padding: 20px;

}

.tasks-section h2,

.add-task-section h2 {

margin-bottom: 15px;

color: var(--secondary-color);

font-size: 1.5rem;

border-bottom: 2px solid var(--light-gray);

padding-bottom: 8px;

}

.tasks-list {

margin-bottom: 30px;

}

.task-item {

display: flex;

align-items: center;

padding: 12px 15px;

margin-bottom: 8px;

background-color: var(--light-gray);

border-radius: 4px;

transition: background-color 0.2s;

}

.task-item:hover {

background-color: var(--medium-gray);

}

.task-checkbox {

margin-right: 12px;

transform: scale(1.2);

}

.task-label {

flex-grow: 1;

cursor: pointer;

}

.task-date {

background-color: var(--dark-gray);

color: white;

padding: 4px 8px;

border-radius: 12px;

font-size: 0.8rem;

margin-left: 10px;

}

.add-task-form {

display: flex;

gap: 10px;

margin-top: 15px;

}

.task-input {

flex-grow: 1;

padding: 10px;

border: 1px solid var(--medium-gray);

border-radius: 4px;

font-size: 1rem;

}

.task-date-input {

padding: 10px;

border: 1px solid var(--medium-gray);

border-radius: 4px;

font-size: 1rem;

min-width: 140px;

}

.add-button {

padding: 10px 20px;

background-color: var(--success-color);

color: white;

border: none;

border-radius: 4px;

font-size: 1rem;

cursor: pointer;

transition: background-color 0.2s;

}

.add-button:hover {

background-color: #218838;

}

.task-item.completed {

opacity: 0.7;

}

.task-item.completed .task-label {

text-decoration: line-through;

}

.task-actions {

display: flex;

align-items: center;

gap: 5px;

margin-left: 10px;

}

.edit-btn {

background: none;

border: none;

color: #6c757d;

font-size: 1rem;

cursor: pointer;

padding: 2px 5px;

border-radius: 3px;

transition: background-color 0.2s, color 0.2s;

}

.edit-btn:hover {

background-color: #e9ecef;

color: #495057;

}

.delete-btn {

background: none;

border: none;

color: #dc3545;

font-size: 1.2rem;

cursor: pointer;

padding: 0 5px;

line-height: 1;

}

.delete-btn:hover {

color: #a71d2a;

}

.loading {

opacity: 0.6;

pointer-events: none;

}

.date-navigation {

display: flex;

align-items: center;

justify-content: center;

gap: 15px;

margin-top: 15px;

}

.nav-button {

padding: 8px 12px;

background-color: var(--secondary-color);

color: white;

border: none;

border-radius: 4px;

font-size: 0.9rem;

cursor: pointer;

transition: background-color 0.2s;

}

.nav-button:hover {

background-color: #0d4b6e;

}

.date-picker-container {

margin: 0;

}

calendar.js

**Opis:** Skrypt odpowiedzialny za renderowanie interaktywnego kalendarza zadań, pobieranie danych z API oraz wizualne oznaczanie dni z zadaniami i ich statusami.

document.addEventListener('DOMContentLoaded', () => {

const currentMonthYearDisplay = document.getElementById('currentMonthYear');

const calendarGrid = document.getElementById('calendarDays');

const prevMonthButton = document.getElementById('prevMonthBtn');

const nextMonthButton = document.getElementById('nextMonthBtn');

let displayedDate = new Date();

// api url

const API\_BASE\_URL = "http://localhost:8080/api/v1";

// wczytanie zadań na dany miesiąc

async function fetchTasksForMonth(year, monthZeroIndexed) {

if (!API\_BASE\_URL) {

console.error("API\_BASE\_URL nie jest skonfigurowany.");

return {};

}

const monthOneIndexed = monthZeroIndexed + 1;

// format daty

const apiDateParam = `${year}-${String(monthOneIndexed).padStart(2, '0')}-01`;

const apiUrl = `${API\_BASE\_URL}/task/month/${apiDateParam}`;

try {

const response = await fetch(apiUrl);

if (!response.ok) {

console.error(`Błąd API (${apiUrl}): Status ${response.status}`);

calendarGrid.innerHTML = `<div style="text-align:center; grid-column: 1 / -1; padding: 20px;">Błąd ładowania danych.</div>`;

return {};

}

const tasks = await response.json();

//

const tasksGroupedByDate = {};

if (Array.isArray(tasks)) {

tasks.forEach(task => {

const taskDateString = task.date;

if (!taskDateString || !/^\d{4}-\d{2}-\d{2}$/.test(taskDateString)) {

console.warn("Pominięto zadanie z nieprawidłową datą:", task);

return;

}

if (!tasksGroupedByDate[taskDateString]) {

tasksGroupedByDate[taskDateString] = { completedTasks: 0, totalTasks: 0 };

}

tasksGroupedByDate[taskDateString].totalTasks++;

if (task.completed === true) {

tasksGroupedByDate[taskDateString].completedTasks++;

}

});

} else {

console.error("Odpowiedź z API nie jest tablicą:", tasks);

}

return tasksGroupedByDate;

} catch (error) {

console.error(`Błąd sieci lub przetwarzania JSON z ${apiUrl}:`, error);

calendarGrid.innerHTML = `<div style="text-align:center; grid-column: 1 / -1; padding: 20px;">Błąd połączenia.</div>`;

return {};

}

}

// renderowanie kalendarza

async function renderCalendar() {

const year = displayedDate.getFullYear();

const month = displayedDate.getMonth();

// format daty

currentMonthYearDisplay.textContent = `${new Intl.DateTimeFormat('pl-PL', { month: 'long' }).format(displayedDate)} ${year}`;

calendarGrid.innerHTML = '<div style="text-align:center; grid-column: 1 / -1; padding: 20px;">Ładowanie...</div>';

// wczytanie zadań

const monthlyTasksData = await fetchTasksForMonth(year, month);

calendarGrid.innerHTML = '';

// ustawienie dni miesiąca

const firstDayOfMonthObject = new Date(year, month, 1);

const lastDayOfMonthObject = new Date(year, month + 1, 0);

const totalDaysInMonth = lastDayOfMonthObject.getDate();

// Ustalanie pierwszego dnia miesiąca

let dayOfWeekForFirstDay = firstDayOfMonthObject.getDay();

dayOfWeekForFirstDay = (dayOfWeekForFirstDay === 0) ? 6 : dayOfWeekForFirstDay - 1;

for (let i = 0; i < dayOfWeekForFirstDay; i++) {

const emptyCell = document.createElement('div');

emptyCell.classList.add('empty');

calendarGrid.appendChild(emptyCell);

}

// tworzenie dni miesiąca

for (let dayNumber = 1; dayNumber <= totalDaysInMonth; dayNumber++) {

const dayCell = document.createElement('div');

dayCell.textContent = dayNumber;

const cellDateString = `${year}-${String(month + 1).padStart(2, '0')}-${String(dayNumber).padStart(2, '0')}`;

const tasksForThisDay = monthlyTasksData[cellDateString];

if (tasksForThisDay && tasksForThisDay.totalTasks > 0) {

const taskCountSpan = document.createElement('span');

taskCountSpan.textContent = tasksForThisDay.totalTasks;

taskCountSpan.classList.add('task-count');

dayCell.appendChild(taskCountSpan);

if (tasksForThisDay.completedTasks < tasksForThisDay.totalTasks) {

dayCell.classList.add('tasks-pending');

} else {

dayCell.classList.add('tasks-completed-all');

}

}

const today = new Date();

if (dayNumber === today.getDate() && month === today.getMonth() && year === today.getFullYear()) {

dayCell.classList.remove('tasks-pending', 'tasks-completed-all');

dayCell.classList.add('today');

}

calendarGrid.appendChild(dayCell);

}

}

// przyciski do zmiany miesiąca

prevMonthButton.addEventListener('click', () => {

displayedDate.setMonth(displayedDate.getMonth() - 1);

renderCalendar();

});

nextMonthButton.addEventListener('click', () => {

displayedDate.setMonth(displayedDate.getMonth() + 1);

renderCalendar();

});

if (API\_BASE\_URL) {

renderCalendar();

} else {

console.error("API\_BASE\_URL nie jest ustawiony. Kalendarz nie zostanie zainicjowany.");

currentMonthYearDisplay.textContent = "Błąd konfiguracji";

calendarGrid.innerHTML = '<div style="text-align:center; grid-column: 1 / -1; padding: 20px;">Błąd konfiguracji API.</div>';

}

});

summary.js

**Opis:** Skrypt zarządzający podsumowaniem miesięcznym zadań, w tym statystykami ukończenia oraz listą zadań z możliwością nawigacji między miesiącami.

document.addEventListener('DOMContentLoaded', function() {

const prevMonthBtn = document.getElementById('prev-month');

const nextMonthBtn = document.getElementById('next-month');

const monthDisplay = document.getElementById('current-month');

const tasksList = document.getElementById('tasks-list');

const totalTasksEl = document.getElementById('total-tasks');

const completedTasksEl = document.getElementById('completed-tasks');

const completionRateEl = document.getElementById('completion-rate');

let currentDate = new Date();

// Ustaw początkową datę

updateMonthDisplay();

loadData();

// poprzedni i następny miesiąc

prevMonthBtn.addEventListener('click', () => {

currentDate.setMonth(currentDate.getMonth() - 1);

updateMonthDisplay();

loadData();

});

nextMonthBtn.addEventListener('click', () => {

currentDate.setMonth(currentDate.getMonth() + 1);

updateMonthDisplay();

loadData();

});

// Aktualizacja wyświetlania miesiąca

function updateMonthDisplay() {

const options = { month: 'long', year: 'numeric' };

monthDisplay.textContent = currentDate.toLocaleDateString('pl-PL', options);

}

// Załaduj dane

async function loadData() {

const year = currentDate.getFullYear();

const month = String(currentDate.getMonth() + 1).padStart(2, '0');

const apiUrl = `http://localhost:8080/api/v1/task/month/${year}-${month}-01`;

try {

const response = await fetch(apiUrl);

if (!response.ok) throw new Error('Failed to fetch data');

const tasks = await response.json();

renderTasks(tasks);

updateStats(tasks);

} catch (error) {

console.error('Error:', error);

tasksList.innerHTML = '<p class="error-message">Nic nie znaleziono</p>';

}

}

// Renderowanie zadań

function renderTasks(tasks) {

tasksList.innerHTML = '';

if (!tasks || tasks.length === 0) {

tasksList.innerHTML = '<p class="no-tasks">No tasks found for this month</p>';

return;

}

// Sortowanie

tasks.sort((a, b) => new Date(a.date) - new Date(b.date));

tasks.forEach(task => {

const taskElement = document.createElement('div');

taskElement.className = 'task-item';

taskElement.innerHTML = `

<div class="task-name">${task.name}</div>

<div class="task-date">${formatDate(task.date)}</div>

<div class="task-status ${task.completed ? '' : 'pending'}">

${task.completed ? '✓ Ukończono' : '✗ W trakcie'}

</div>

`;

tasksList.appendChild(taskElement);

});

}

// Aktualizacja statystyk

function updateStats(tasks) {

const total = tasks.length;

const completed = tasks.filter(task => task.completed).length;

const rate = total > 0 ? Math.round((completed / total) \* 100) : 0;

totalTasksEl.textContent = total;

completedTasksEl.textContent = completed;

completionRateEl.textContent = `${rate}%`;

// Kolorowanie

completionRateEl.style.color = rate >= 75 ? '#28a745' :

rate >= 50 ? '#ffc107' :

'#dc3545';

}

// Formatowanie daty

function formatDate(dateString) {

const date = new Date(dateString);

return date.toLocaleDateString('pl-PL', {

day: 'numeric',

month: 'short'

});

}

});

task.js

**Opis:** Główny skrypt aplikacji obsługujący dodawanie, edycję, usuwanie i oznaczanie zadań jako ukończone, z integracją API i nawigacją między dniami.

document.addEventListener('DOMContentLoaded', function() {

const datePicker = document.getElementById('date-picker');

const tasksList = document.querySelector('.tasks-list');

const addTaskForm = document.querySelector('.add-task-form');

const taskInput = document.querySelector('.task-input');

const taskDateInput = document.querySelector('.task-date-input');

const prevDayBtn = document.getElementById('prev-day');

const nextDayBtn = document.getElementById('next-day');

// API URL

const API\_URL = 'http://localhost:8080/api/v1/task';

// Ustaw początkową datę na dzisiaj

const today = new Date().toISOString().split('T')[0];

datePicker.value = today;

taskDateInput.value = today;

// Dodaj nasłuchiwanie zdarzeń

datePicker.addEventListener('change', loadTasksForDate);

addTaskForm.addEventListener('submit', handleAddTask);

prevDayBtn.addEventListener('click', navigateToPreviousDay);

nextDayBtn.addEventListener('click', navigateToNextDay);

// Początkowe załadowanie zadań

loadTasksForDate();

// Załadowanie zadań dla wybranej daty

async function loadTasksForDate() {

try {

const date = datePicker.value;

const response = await fetch(`${API\_URL}/${date}`);

if (!response.ok) {

if (response.status === 404) {

tasksList.innerHTML = '<p class="no-tasks">Nic nie znaleziono</p>';

return;

}

throw new Error('Failed to fetch tasks');

}

const tasks = await response.json();

if (!tasks || tasks.length === 0) {

tasksList.innerHTML = '<p class="no-tasks">Nic nie znaleziono</p>';

return;

}

renderTasks(tasks);

} catch (error) {

console.error('Error loading tasks:', error);

tasksList.innerHTML = '<p class="no-tasks">Nic nie znaleziono</p>';

}

}

// Renderowanie zadań

function renderTasks(tasks) {

tasksList.innerHTML = '';

if (!tasks || tasks.length === 0) {

tasksList.innerHTML = '<p>Nie znaleziono</p>';

return;

} tasks.forEach(task => {

const taskElement = document.createElement('div');

taskElement.className = `task-item ${task.completed ? 'completed' : ''}`;

taskElement.innerHTML = `

<input type="checkbox" id="task-${task.id}" class="task-checkbox"

${task.completed ? 'checked' : ''}>

<label for="task-${task.id}" class="task-label">${task.name}</label>

<span class="task-date">${task.date}</span>

<div class="task-actions">

<button class="edit-btn" data-task-id="${task.id}">✏️</button>

<button class="delete-btn" data-task-id="${task.id}">×</button>

</div>

`;

tasksList.appendChild(taskElement);

const checkbox = taskElement.querySelector('.task-checkbox');

checkbox.addEventListener('change', () => toggleTaskCompletion(task.id));

const editBtn = taskElement.querySelector('.edit-btn');

editBtn.addEventListener('click', () => editTask(task.id, task.name));

const deleteBtn = taskElement.querySelector('.delete-btn');

deleteBtn.addEventListener('click', () => deleteTask(task.id));

});

}

// Dodanie nowego zadania

async function handleAddTask(e) {

e.preventDefault();

const newTask = {

name: taskInput.value.trim(),

date: taskDateInput.value

};

if (!newTask.name) return;

try {

const response = await fetch(API\_URL, {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(newTask)

});

if (!response.ok) throw new Error('Failed to add task');

taskInput.value = '';

loadTasksForDate();

} catch (error) {

console.error('Error adding task:', error);

}

}

// Zmiana statusu zadania na ukończone/nieukończone

async function toggleTaskCompletion(taskId) {

try {

const response = await fetch(`${API\_URL}/${taskId}/complete`, {

method: 'PUT'

});

if (!response.ok) throw new Error('Failed to update task');

loadTasksForDate();

} catch (error) {

console.error('Error completing task:', error);

}

}

// Edycja zadania

function editTask(taskId, currentName) {

const newName = prompt('Edytuj nazwę zadania:', currentName);

if (newName === null || newName.trim() === '' || newName.trim() === currentName) {

return; // Użytkownik anulował lub nie wprowadził zmian

}

updateTask(taskId, newName.trim());

} // Aktualizacja zadania

async function updateTask(taskId, newName) {

try {

const response = await fetch(`${API\_URL}/${taskId}?name=${encodeURIComponent(newName)}`, {

method: 'PUT'

});

if (!response.ok) throw new Error('Failed to update task');

loadTasksForDate();

} catch (error) {

console.error('Error updating task:', error);

alert('Nie udało się zaktualizować zadania');

}

}

// Usunięcie zadania

async function deleteTask(taskId) {

try {

const response = await fetch(`${API\_URL}/${taskId}`, {

method: 'DELETE'

});

if (!response.ok) throw new Error('Failed to delete task');

loadTasksForDate();

} catch (error) {

console.error('Error deleting task:', error);

}

}

// Nawigacja do poprzedniego/następnego dnia

function navigateToPreviousDay() {

const currentDate = new Date(datePicker.value);

currentDate.setDate(currentDate.getDate() - 1);

datePicker.value = formatDate(currentDate);

loadTasksForDate();

}

function navigateToNextDay() {

const currentDate = new Date(datePicker.value);

currentDate.setDate(currentDate.getDate() + 1);

datePicker.value = formatDate(currentDate);

loadTasksForDate();

}

// Formatowanie daty

function formatDate(date) {

const year = date.getFullYear();

const month = String(date.getMonth() + 1).padStart(2, '0');

const day = String(date.getDate()).padStart(2, '0');

return `${year}-${month}-${day}`;

}

});

navbar.html

**Opis:** Fragment kodu HTML definiujący pasek nawigacyjny aplikacji z linkami do głównych sekcji: Zadania, Podsumowanie i Kalendarz.

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

</head>

<body>

<nav th:fragment="navbar" class="nav-top-bar">

<div class="nav-bar-container">

<a th:href="@{/task}" class="nav-bar-link">Zadania</a>

<a th:href="@{/summary}" class="nav-bar-link">Podsumowanie</a>

<a th:href="@{/calendar}" class="nav-bar-link">Kalendarz</a>

</div>

</nav>

</body>

</html>

calendar.html

**Opis:** Szablon strony kalendarza z interaktywnym widokiem miesięcznym, przyciskami nawigacji oraz obszarem na wyświetlanie dni z zadaniami.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Habit Tracker | Kalendarz</title>

<link rel="stylesheet" th:href="@{/css/navbar.css}">

<link rel="stylesheet" th:href="@{/css/calendar.css}">

<script src="/script/calendar.js"></script>

</head>

<body>

<div th:replace="~{fragments/navbar :: navbar}"></div>

<div class="calendar-container">

<div class="calendar-header">

<button id="prevMonthBtn">&lt;</button>

<h2 id="currentMonthYear"></h2>

<button id="nextMonthBtn">&gt;</button>

</div>

<div class="calendar-weekdays">

<div>Pn</div>

<div>Wt</div>

<div>Śr</div>

<div>Cz</div>

<div>Pt</div>

<div>So</div>

<div>Nd</div>

</div>

<div class="calendar-days" id="calendarDays">

</div>

</div>

</body>

</html>

summary.html

**Opis:** Strona podsumowania miesięcznego z sekcjami statystyk (liczba zadań, wykonane, stopień realizacji) oraz listą zadań.

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Habit Tracker | Podsumowanie</title>

<link rel="stylesheet" th:href="@{/css/navbar.css}">

<link rel="stylesheet" th:href="@{/css/summary.css}">

<script src="/script/summary.js"></script>

</head>

<body>

<div th:replace="~{fragments/navbar :: navbar}"></div>

<div class="summary-container">

<header class="summary-header">

<h1>Podsumowanie miesiąca</h1>

<div class="month-navigation">

<button class="month-button" id="prev-month">←</button>

<div class="month-display" id="current-month">Styczeń 2077</div>

<button class="month-button" id="next-month">→</button>

</div>

</header>

<main class="summary-content">

<div class="stats-grid">

<div class="stat-card highlight">

<h3>Łącznie zadań</h3>

<div class="stat-value" id="total-tasks">0</div>

</div>

<div class="stat-card highlight">

<h3>Wykonano</h3>

<div class="stat-value" id="completed-tasks">0</div>

</div>

<div class="stat-card highlight">

<h3>Stopień wykonanych</h3>

<div class="stat-value" id="completion-rate">0%</div>

</div>

</div>

<section class="tasks-summary">

<h2>Lista zadań</h2>

<div class="tasks-list" id="tasks-list">

</div>

</section>

</main>

</div>

</body>

</html>

task.html

**Opis:** Główny interfejs aplikacji do zarządzania zadaniami, zawierający listę zadań, formularz dodawania oraz nawigację między dniami.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Habit Tracker | Zadania</title>

<link rel="stylesheet" th:href="@{/css/navbar.css}">

<link rel="stylesheet" th:href="@{/css/task.css}">

<script src="/script/task.js"></script>

</head>

<body>

<div th:replace="~{fragments/navbar :: navbar}"></div>

<div class="app-container">

<header class="app-header">

<h1>Menedżer zadań</h1>

<div class="date-navigation">

<button class="nav-button" id="prev-day">←</button>

<div class="date-picker-container">

<input type="date" id="date-picker" class="date-picker">

</div>

<button class="nav-button" id="next-day">→</button>

</div>

</header>

<main class="main-content">

<section class="tasks-section">

<h2>Dzisiejsze zadania</h2>

<div class="tasks-list"></div>

</section>

<section class="add-task-section">

<h2>Dodaj nowe zadanie</h2>

<form class="add-task-form">

<input type="text" placeholder="Nazwa" class="task-input" required>

<input type="date" class="task-date-input">

<button type="submit" class="add-button">Dodaj</button>

</form>

</section>

</main>

</div>

</body>

</html>

Konfiguracja application.properties

spring.application.name=HabitTracker  
  
#PostgreSQL  
#spring.datasource.url=jdbc:postgresql://localhost:5432/habits  
#spring.datasource.username=user  
#spring.datasource.password=root  
#spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect  
  
#H2  
spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
  
spring.h2.console.enabled=true  
spring.h2.console.path=/h2-console  
  
spring.jpa.hibernate.ddl-auto=create-drop  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.format\_sql=true  
  
server.error.include-message=*always*