Dokumentacja projektowa

Wirtualne Przedszkole

Projekt inżynierski

Cel projektu

Wirtualne przedszkole zostało stworzone specjalnie dla Przedszkola nr 25. Nasza aplikacja zapewnia wszystko w jednym miejscu, dzięki czemu nauczyciele i rodzice w przedszkolu mogą łatwo uzyskać dostęp do ważnych informacji, udostępniać pliki i komunikować się ze sobą w jednej zintegrowanej platformie. Rozumiemy, że korzystanie z różnych platform do różnych celów może prowadzić do komplikacji i nieporozumień, zwłaszcza dla starszych nauczycieli, którzy mogą być mniej zaznajomieni z nowymi technologiami. Nasza aplikacja ma na celu uproszczenie i poprawienie tego procesu, łącząc wszystko w jednym miejscu i umożliwiając łatwe pozostanie informowanym i połączenie się ze sobą wszystkich zaangażowanych stron w Przedszkolu nr 25. Mamy nadzieję, że nasza aplikacja ułatwi pracę i życie nauczycielom oraz rodzicom w Przedszkolu nr 25.

Lista elementów projektu

Elementy programistyczne:

- aplikacja internetowa kompatybilna z urządzeniami mobilnymi
- serwer bazy danych MySQL
- backend java, framework Spring boot
- frontend javascript, react
- nginx

Elementy nie programistyczne:

- dokumentacja projektowa
- raporty z testów

Lista funkcjonalności

Lista wymagań funkcjonalnych przedstawiona za pomocą User Stories:

1. Jako nauczyciel/dyrektor/rodzic chcę się zalogować, żeby uzyskać dostęp do konta.
2. Jako nauczyciel/dyrektor placówki chcę umieścić ogłoszenie w panelu rodzica, żeby poinformować rodziców o ważnym wydarzeniu z życia przedszkola.
3. Jako nauczyciel/dyrektor chce umieścić filmy/zdjęcia dla określonych grup, żeby rodzice mieli do nich łatwy dostęp.
4. Jako nauczyciel chcę móc przejrzeć profil rodzica, którego dzieckiem się opiekuję, żeby móc się z nim skontaktować w razie potrzeby
5. Jako administrator/dyrektor chcę dodać nowego rodzica do bazy danych, aby rodzic miał możliwość korzystania z systemu

6. Jako administrator/dyrektor chcę usunąć rodzica z bazy danych, żeby nie zajmował w niej miejsca rodzic, którego dziecko przestaje uczęszczać do placówki
7. Jako dyrektor chcę utworzyć grupę zajęciową
8. Jako dyrektor chce przydzielić podopiecznego do grupy zajęciowej, żeby jego rodzic miał dostęp do multimediów i informacji udostępnionym konkretnej grupie
9. Jako dyrektor chcę móc przejrzeć profile nauczycieli/rodziców, żeby uzyskać dane kontaktowe, albo usunąć nieaktywne konta
10. Jako rodzic chcę uzyskać podstawowe informacje o kadrze nauczycielskiej, żeby wiedzieć kto zajmuje się moim dzieckiem.
11. Jako rodzic chcę pobrać dokumenty o przedszkolu np. statut przedszkola, koncepcja przedszkola, regulamin rady rodziców itp., żeby móc zapoznać się z nimi zapoznać
12. Jako rodzic chce zmienić zdjęcie profilowe, żeby pracownicy placówki wiedzieli jak wyglądam.
13. Jako rodzic chcę edytować informacje osobiste w profilu użytkownika, żeby w razie zmiany np. nr telefonu lub adresu zamieszkania przedszkole miało aktualne dane.
14. Jako rodzic chce przejrzeć zdjęcia z przedszkolnych wydarzeń, żeby znaleźć na nich swoje dziecko
15. Jako użytkownik chce zmienić hasło, żeby zadbać o bezpieczeństwo swojego konta.
16. Jako rodzic chcę otrzymać i odczytać wiadomość od dyrektora/nauczycieli, żeby być na bieżąco z ważnymi informacjami.
17. Jako rodzic chce potwierdzić odczytanie otrzymanej wiadomości, aby potwierdzić, że zapoznałem się z informacją.
18. Jako rodzic chce mieć dostęp do bazy linków z materiałami edukacyjnymi, aby w szybki sposób uzyskać materiały do domowej nauki mojego dziecka
19. Jako dyrektor/nauczyciel chcę wysłać wiadomość do jednego lub wielu rodziców (np. rodziców z konkretnej klasy)
20. Jako dyrektor chcę widzieć czy wiadomość została odczytana

Backend

W naszym projekcie backend napisany został za pomocą języka Java w popularnym frameworku Spring Boot.

Spring Boot to narzędzie znacznie skracające i ułatwiające proces tworzenia aplikacji. Dostarcza programiście mechanizmy sprawiające, że czas produkcji oprogramowania jest krótszy ale zachowane są przy tym dobre praktyki programistyczne i standardy.
Spring Boot zapewnia konfigurację, dzięki której pisany kod jest jeszcze bardziej zwięzły. Zapewnia też kontener aplikacji pozwalający na proste uruchamianie systemu.

Wykorzystaliśmy Spring Data JPA aby zapewnić abstrakcję dostępu do danych. Eliminuje on znacząco potrzebę pisania kodu dostępowego, czy też zapytań do bazy danych.
Jest to framework, który bazuje na JPA (Java Persistance Api), którego dostawcą jest domyślnie Hibernate. Jest on kolejną warstwą abstrakcji, nabudowaną na tych właśnie dwóch narzędziach.

W celu zapewnienia bezpieczeństwa w dostępie do danych przetwarzanych przez nasz system wykorzystaliśmy Spring Boot Security. Jest to framework, który zapewnia różne funkcje bezpieczeństwa, takie jak: uwierzytelnianie, autoryzacja do tworzenia bezpiecznych aplikacji.

Aby umożliwić komunikację pomiędzy frontendem a backendem zastosowaliśmy REST API.

Architektura backendu:

Controller - Service - Repository

Controller służy do obsługi żądań Http
Service - tu zamieszczona jest logika i przetwarzane są dane
Repository służy do komunikacją z bazą danych

Frontend

Nasz frontend został zbudowany przy pomocy biblioteki React JS.

React jest tak zwaną biblioteką języka programowania JavaScript. Wykorzystuje się ją podczas tworzenia interfejsów użytkownika dla różnego typu aplikacji. Dostępna jest dla szerokiego grona odbiorców na zasadzie open source. Dzięki niej możesz więc stworzyć bardzo złożony interfejs, składający się z małych, oddzielonych od siebie elementów.

Biblioteka React wykorzystuje nowoczesny sposób renderowania stron internetowych. Dzięki temu stają się one bardziej dynamiczne. Sprawia to, że framework ten stanowi niezwykle innowacyjne podejście w programowaniu. Jest wygodny zarówno dla samego twórcy – programisty, jak i dla końcowego użytkownika, czyli odbiorcy aplikacji.

Baza Danych

Baza danych została postawiona na MySQL server. MySQL to wolnodostępny, otwartoźródłowy system zarządzania relacyjnymi bazami danych.

Testowanie

W projekcie skupiliśmy się na testowaniu zarówno części backendowej, jak i frontendowej aplikacji. Głównym celem było upewnienie się, że poszczególne części aplikacji działają poprawnie i przynoszą oczekiwane rezultaty.

Do testowania frontendu wykorzystaliśmy Jest jako naszą platformę testową. Jest to popularna biblioteka testowa JavaScript, która zapewnia różnorodne funkcje i narzędzia testowe ułatwiające pisanie i uruchamianie testów. Aby przetestować różne punkty końcowe interfejsu API, wykorzystaliśmy fałszywą bibliotekę axios do symulacji wywołań do zaplecza i kontrolowania zwracanych wyników.

Wykorzystaliśmy również React Testing Library jako naszą platformę testową. React Testing Library to popularna biblioteka do testowania aplikacji React, która zapewnia różnorodne funkcje do testowania różnych komponentów i elementów aplikacji.

Jednym z kluczowych wyzwań podczas testowania frontendu było upewnienie się, że komponenty są prawidłowo połączone z backendem. Aby sprostać temu wyzwaniu, wykorzystaliśmy pozorowaną bibliotekę axios do symulacji wywołań interfejsu API do zaplecza i kontrolowania zwracanych wyników. To pozwoliło nam odizolować testy frontendu od backendu i przetestować różne komponenty w kontrolowanym środowisku.

Oprócz testowania komponentów przetestowaliśmy również różne strony w aplikacji. Polegało to na przetestowaniu różnych elementów na stronie, takich jak nagłówek, stopka i główna treść, aby upewnić się, że wyświetlają się poprawnie i przynoszą oczekiwane wyniki.

Ogólnie rzecz biorąc, testy w tym projekcie były kluczową częścią upewnienia się, że aplikacja działa poprawnie i zapewnia oczekiwane wyniki. Połączenie Jest i React Testing Library pozwoliło nam napisać i uruchomić testy dla frontendowych części aplikacji, dając nam pewność co do funkcjonalności aplikacji i pomagając nam wychwycić i naprawić wszelkie problemy, zanim zostaną wdrożone do produkcji.

Do testowania backendu wykorzystano Junit i Mocks do testowania usług backendu. Junit to szeroko stosowany framework do pisania i uruchamiania testów aplikacji Java. Zapewnia prosty i zorganizowany sposób pisania przypadków testowych dla różnych części aplikacji Java, w tym klas i metod. Ramy umożliwiają programistom uruchamianie automatycznych testów i sprawdzanie poprawności wyników testów, aby upewnić się, że kod działa zgodnie z oczekiwaniami.

Z drugiej strony mocki służyły do izolowania testów jednostkowych od zewnętrznych zależności. W naszym projekcie powstały makiety dla usług, które nie były jeszcze dostępne lub dla usług, które były zbyt drogie, aby wywołać je podczas testów. Dzięki temu przypadki testowe były szybsze i bardziej niezawodne, ponieważ nie były zależne od zachowania systemów zewnętrznych. Makiety zostały stworzone przy użyciu biblioteki mockito, która zapewnia potężne i elastyczne API do tworzenia i weryfikowania makiet.

Wykorzystanie Junit i Mocks w projekcie sprawiło, że proces deweloperski stał się płynniejszy i wydajniejszy. Junit zapewnił prosty i zorganizowany sposób pisania i uruchamiania testów, podczas gdy makiety pozwoliły nam wyizolować przypadki testowe i skrócić czas i zasoby potrzebne do ich uruchomienia. Dzięki temu programiści mogli skupić się na pisaniu i ulepszaniu kodu, zamiast martwić się o testowanie i sprawdzanie poprawności wyników.

W kodzie wykorzystaliśmy kilka różnych części i komponentów do przeprowadzenia testów.

Junit: Junit to framework Java używany do pisania i uruchamiania testów. Zapewnia adnotacje i metody asercji, które pomagają w strukturyzacji i sprawdzeniu poprawności testów. Wykorzystaliśmy Junit w naszym projekcie Spring Boot do przetestowania komponentów zaplecza.

Mocki: Mocki to obiekty, które symulują zachowanie rzeczywistych obiektów. W naszym projekcie Spring Boot użyliśmy makiet, aby wyizolować testowane komponenty i kontrolować ich zachowanie. To pozwoliło nam przetestować komponenty w kontrolowanym środowisku, bez polegania na rzeczywistym zachowaniu innych komponentów.

Spring Boot Test: Spring Boot Test to platforma testowa udostępniana przez Spring Boot do testowania komponentów aplikacji Spring Boot. Zapewnia szereg narzędzi i funkcji, które ułatwiają pisanie testów, takich jak obsługa JUnit, makiety obiektów i wbudowane serwery.

@MockBean: Adnotacja @MockBean służy do tworzenia próbnych obiektów w testach Spring Boot. Można go użyć do zastąpienia rzeczywistych obiektów w kontekście testowym obiektami pozorowanymi.

@Autowired: adnotacja @Autowired służy do określenia, że pole lub parametr konstruktora powinien zostać wstrzyknięty z instancją określonego typu. Jest to przydatne w testowaniu, ponieważ pozwala nam wstrzykiwać pozorowane obiekty do testowanych komponentów.

Asercje: Asercje to stwierdzenia, które weryfikują zachowanie komponentu. W naszych testach używaliśmy asercji, aby sprawdzić, czy testowane komponenty zachowują się zgodnie z oczekiwaniami. Na przykład możemy użyć asercji, aby sprawdzić, czy próbny obiekt został wywołany określoną liczbę razy lub czy odpowiedź z interfejsu API była poprawna.

@Test: Adnotacja @Test służy do wskazania, że metoda powinna zostać uruchomiona jako test. Gdy Junit uruchomi testy, wykona wszystkie metody z adnotacją @Test.

Ogólnie rzecz biorąc, wykorzystanie Junit i Mocks było kluczowym czynnikiem zapewniającym jakość i niezawodność usług backendowych w projekcie. Testy były przeprowadzane regularnie, a wszelkie awarie były natychmiast identyfikowane i usuwane, co pozwalało zespołowi deweloperskiemu wyłapywać i naprawiać wszelkie błędy na wczesnym etapie procesu tworzenia. Dzięki temu kod był bardziej solidny i niezawodny, a także zapewniał, że usługi zaplecza będą działać zgodnie z oczekiwaniami w środowisku produkcyjnym.

Polecenia dla testowania frontend
npm install --save-dev ts-node -f
npm test

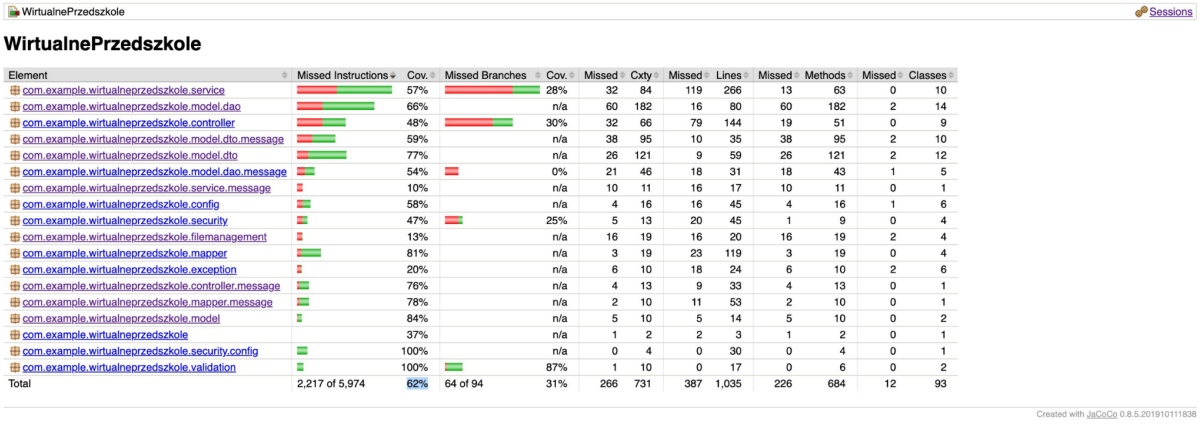
Polecenia dla testowania backend
mvn clean verify

target folder -> site -> html file

Frontend raport:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	42.96	17.15	43.54	43.39	
src	18.18	25	20	18.18	
App.js	40	25	33.33	40	161-166
index.js	0	100	0	0	9-21
src/components/navbar	100	100	100	100	
navbar.js	100	100	100	100	
src/components/sidebar	79.06	40	75	78.04	
sidebar.jsx	76.47	40	72.72	75.75	35-36,43,72,203,225-228
useWindowDimensions.js	88.88	100	80	87.5	18
src/pages	0	100	0	0	
ScrollToTop.js	0	100	0	0	5-11
src/pages/Children	73.07	0	70	75	
Child.js	100	100	100	100	
ChildNavi.js	100	100	100	100	
ChildrenComponent.js	45.45	0	42.85	50	7-10,33-38,76
ChildrenNavi.js	50	100	50	50	11-21
ChildrenService.js	100	100	100	100	
src/pages/CreateChild	97.05	75	100	96.96	
AddChild.js	100	100	100	100	
CreateChild.js	96.87	75	100	96.77	54
src/pages/CreateGroup	47.82	0	22.22	47.82	
AddGroup.js	50	100	0	50	8
CreateGroup.js	47.61	0	25	47.61	21-32,38,42-47,57-58
src/pages/CreateUser	22.9	9.09	9.75	23.8	

Backend raport:



WirtualnePrzedszkole

Sessions

WirtualnePrzedszkole

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.example.wirtualneprzedszkole.service		57%		28%	32	84	119	266	13	63	0	10
com.example.wirtualneprzedszkole.model.dao		66%		n/a	60	182	16	80	60	182	2	14
com.example.wirtualneprzedszkole.controller		48%		30%	32	66	79	144	19	51	0	9
com.example.wirtualneprzedszkole.model.dto.message		59%		n/a	38	95	10	35	38	95	2	10
com.example.wirtualneprzedszkole.model.dto		77%		n/a	26	121	9	59	26	121	2	12
com.example.wirtualneprzedszkole.model.dao.message		54%		0%	21	46	18	31	18	43	1	5
com.example.wirtualneprzedszkole.service.message		10%		n/a	10	11	16	17	10	11	0	1
com.example.wirtualneprzedszkole.config		58%		n/a	4	16	16	45	4	16	1	6
com.example.wirtualneprzedszkole.security		47%		25%	5	13	20	45	1	9	0	4
com.example.wirtualneprzedszkole.filemanagement		13%		n/a	16	19	16	20	16	19	2	4
com.example.wirtualneprzedszkole.mapper		81%		n/a	3	19	23	119	3	19	0	4
com.example.wirtualneprzedszkole.exception		20%		n/a	6	10	18	24	6	10	2	6
com.example.wirtualneprzedszkole.controller.message		76%		n/a	4	13	9	33	4	13	0	1
com.example.wirtualneprzedszkole.mapper.message		78%		n/a	2	10	11	53	2	10	0	1
com.example.wirtualneprzedszkole.model		84%		n/a	5	10	5	14	5	10	0	2
com.example.wirtualneprzedszkole		37%		n/a	1	2	2	3	1	2	0	1
com.example.wirtualneprzedszkole.security.config		100%		n/a	0	4	0	30	0	4	0	1
com.example.wirtualneprzedszkole.validation		100%		87%	1	10	0	17	0	6	0	2
Total	2,217 of 5,974	62%	64 of 94	31%	266	731	387	1,035	226	684	12	93

Created with JaCoCo 0.8.5.201910111838

Acceptance tests:

```java
no usages  new *
@Test
@WithMockUser(roles = "ADMIN")
void addChild_ShouldReturnBadRequest_WhenChildDtoIsInvalid() throws Exception {
    ChildDto childDto = ChildDto.builder().name("").lastName("").classId(1L).parents(null).build();
    String childDtoJson = objectMapper.writeValueAsString(childDto);

    mockMvc.perform(post( urlTemplate: "/api/child")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(childDtoJson))
            .andExpect(status().isBadRequest());

    verify(childService, times( wantedNumberOfInvocations: 0)).addChild(ArgumentMatchers.any());
}
```

Functional tests:

```java
@Test
public void getCurrentUser_ShouldReturnUser() {
    User user = new User();
    Authentication authentication = mock(Authentication.class);
    SecurityContext securityContext = mock(SecurityContext.class);
    when(securityContext.getAuthentication()).thenReturn(authentication);
    SecurityContextHolder.setContext(securityContext);
    when(userRepo.findByEmail(any())).thenReturn(Optional.of(user));

    User result = userService.getCurrentUser();

    verify(userRepo, times( wantedNumberOfInvocations: 1)).findByEmail(any());
    assertEquals(user, result);
}
```

Unit tests:

```java
@Test
public void testValidPassword() {
    PasswordConstraintValidator validator = new PasswordConstraintValidator();
    ConstraintValidatorContext context = mock(ConstraintValidatorContext.class);
    when(context.buildConstraintViolationWithTemplate(anyString()))
            .thenReturn(mock(ConstraintValidatorContext.ConstraintViolationBuilder.class));

    boolean result = validator.isValid( password: "ValidPassword1", context);

    assertTrue(result);
    verify(context, never()).disableDefaultConstraintViolation();
    verify(context, never()).buildConstraintViolationWithTemplate(anyString());
}
```

```java
no usages   new *
@Test
@WithMockUser(roles = "ADMIN")
public void updateUser_ShouldReturnBadRequest_WhenUserDtoIsInvalid() throws Exception {
    UserDto userDto = new UserDto();
    userDto.setId(1L);
    userDto.setName("firstName");
    userDto.setLastName("lastName");

    mockMvc.perform(put( urlTemplate: "/api/users")
                    .content(objectMapper.writeValueAsString(userDto))
                    .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isBadRequest());
}
```