**Zaktualizowane instrukcje dla poszczególnych modułów**

**# Instrukcje dla poszczególnych modułów**

Poniżej znajdują się szczegółowe instrukcje, które możesz wykorzystać jako pierwszą wiadomość w poszczególnych czatach modułowych.

**## Standardy kodowania dla wszystkich modułów**

W każdym czacie modułowym dodaj na początku następującą sekcję dotyczącą standardów kodowania:

```

Proszę przestrzegać następujących standardów kodowania dla tego modułu:

1. Formatowanie:

- Używaj 2 spacji jako wcięcie (nie tabulatory)

- Maksymalna długość linii: 100 znaków

- Używaj średników na końcu każdej instrukcji

2. Nazewnictwo:

- Zmienne i funkcje: camelCase (np. getUserData)

- Klasy i komponenty Vue: PascalCase (np. UserProfile)

- Pliki komponentów Vue: PascalCase (np. UserProfile.vue)

- Pliki pomocnicze i usługi: camelCase (np. authService.ts)

- Stałe: UPPER\_SNAKE\_CASE (np. MAX\_FILE\_SIZE)

3. Struktura plików:

- Na początku każdego pliku dodaj komentarz ze ścieżką pliku w projekcie

- Używaj struktury zgodnej z dokumentacją projektu

- Grupuj importy: najpierw biblioteki zewnętrzne, potem importy absolutne (@/), na końcu importy względne

4. Kod:

- We frontendzie używaj Composition API z <script setup>

- W backendzie używaj dekoratorów NestJS zgodnie ze standardami

- Dokumentuj funkcje i metody używając JSDoc/TSDoc

- Używaj typów TypeScript zamiast 'any'

Zapoznaj się również z pełnym dokumentem Style Guide dostępnym w projekcie.

```

**## Struktura projektu dla każdego pliku**

W każdym czacie upewnij się, że Claude tworzy pliki zgodnie ze strukturą projektu i dodaje komentarz ze ścieżką na początku każdego pliku, np.:

```javascript

// frontend/src/components/projects/ProjectCard.vue

```

lub

```typescript

// backend/src/modules/users/users.service.ts

```

Dzięki tym komentarzom, skrypty organizacji plików będą mogły automatycznie umieścić każdy plik we właściwym katalogu.

**## Moduł Frontend (Vue.js + Tailwind CSS)**

```

Proszę o skupienie się na module frontendowym aplikacji zarządzania projektami studenckimi.

Proszę przestrzegać następujących standardów kodowania dla tego modułu:

1. Formatowanie:

- Używaj 2 spacji jako wcięcie (nie tabulatory)

- Maksymalna długość linii: 100 znaków

- Używaj średników na końcu każdej instrukcji

2. Nazewnictwo:

- Zmienne i funkcje: camelCase (np. getUserData)

- Klasy i komponenty Vue: PascalCase (np. UserProfile)

- Pliki komponentów Vue: PascalCase (np. UserProfile.vue)

- Pliki pomocnicze i usługi: camelCase (np. authService.ts)

- Stałe: UPPER\_SNAKE\_CASE (np. MAX\_FILE\_SIZE)

3. Struktura plików:

- Na początku każdego pliku dodaj komentarz ze ścieżką pliku w projekcie

- Używaj struktury zgodnej z dokumentacją projektu

- Grupuj importy: najpierw biblioteki zewnętrzne, potem importy absolutne (@/), na końcu importy względne

4. Komponenty Vue:

- Używaj Composition API z <script setup>

- Kolejność sekcji: <script>, <template>, <style>

- Używaj TypeScript dla lepszego typowania

Zapoznaj się również z pełnym dokumentem Style Guide dostępnym w projekcie.

Dla tego modułu potrzebuję następujących plików:

1. Konfiguracja projektu:

- package.json - z potrzebnymi zależnościami (Vue 3, Tailwind CSS, Pinia, Vue Router)

- vite.config.js - konfiguracja Vite

- tailwind.config.js - konfiguracja Tailwind CSS

2. Struktura aplikacji:

- main.js - punkt wejścia aplikacji

- App.vue - główny komponent

- router/index.js - konfiguracja routingu

- store/index.js - konfiguracja Pinia

3. Komponenty:

- components/layout/Navbar.vue - górny pasek nawigacyjny

- components/projects/ProjectList.vue - komponent listy projektów

- components/projects/ProjectCard.vue - karta pojedynczego projektu

- components/projects/ProjectForm.vue - formularz projektu

4. Widoki:

- views/HomeView.vue - strona główna

- views/ProjectsView.vue - lista projektów

- views/ProjectDetailsView.vue - szczegóły projektu

- views/UserDashboardView.vue - panel użytkownika

5. Serwisy:

- services/api.js - główny serwis API

- services/projectService.js - serwis projektów

- services/authService.js - serwis autoryzacji

Proszę:

1. Używać rzeczywistych nazw plików jako nazw artefaktów (np. "Navbar.vue" zamiast pełnej ścieżki)

2. Na początku każdego pliku dodać komentarz ze ścieżką (np. "// frontend/src/components/layout/Navbar.vue")

3. Na końcu dostarczyć listę referencyjną wszystkich wygenerowanych plików z ich pełnymi ścieżkami

Dzięki temu po pobraniu plików będę mógł/mogła użyć skryptu do automatycznego umieszczenia ich we właściwych katalogach.

```

**## Moduł Backend (NestJS)**

```

Proszę o skupienie się na module backendowym aplikacji zarządzania projektami studenckimi.

Proszę przestrzegać następujących standardów kodowania dla tego modułu:

1. Formatowanie:

- Używaj 2 spacji jako wcięcie (nie tabulatory)

- Maksymalna długość linii: 100 znaków

- Używaj średników na końcu każdej instrukcji

2. Nazewnictwo:

- Zmienne i funkcje: camelCase (np. getUserData)

- Klasy i interfejsy: PascalCase (np. UserService)

- Pliki: kebab-case.type.ts (np. user.service.ts, create-user.dto.ts)

- Stałe: UPPER\_SNAKE\_CASE (np. MAX\_FILE\_SIZE)

3. Struktura plików:

- Na początku każdego pliku dodaj komentarz ze ścieżką pliku w projekcie

- Używaj struktury zgodnej z dokumentacją projektu

- Przestrzegaj konwencji NestJS dla struktury modułów

4. NestJS:

- Używaj dekoratorów zgodnie ze standardami NestJS

- Implementuj logikę biznesową w serwisach, nie w kontrolerach

- Implementuj DTO jako klasy (nie interfejsy) dla walidacji

- Używaj typów TypeScript zamiast 'any'

Zapoznaj się również z pełnym dokumentem Style Guide dostępnym w projekcie.

Dla tego modułu potrzebuję następujących plików:

1. Konfiguracja projektu:

- package.json - z potrzebnymi zależnościami (NestJS, Prisma, JWT)

- nest-cli.json - konfiguracja NestJS

- tsconfig.json - konfiguracja TypeScript

2. Punkt wejścia:

- main.ts - główny plik aplikacji

- app.module.ts - główny moduł aplikacji

3. Moduł użytkowników:

- users.module.ts - moduł użytkowników

- users.controller.ts - kontroler użytkowników

- users.service.ts - serwis użytkowników

- dto/create-user.dto.ts - DTO dla tworzenia użytkownika

- entities/user.entity.ts - encja użytkownika

4. Moduł projektów:

- projects.module.ts - moduł projektów

- projects.controller.ts - kontroler projektów

- projects.service.ts - serwis projektów

- dto/create-project.dto.ts - DTO dla tworzenia projektu

- dto/update-project.dto.ts - DTO dla aktualizacji projektu

- entities/project.entity.ts - encja projektu

5. Moduł kategorii:

- categories.module.ts - moduł kategorii

- categories.controller.ts - kontroler kategorii

- categories.service.ts - serwis kategorii

- entities/category.entity.ts - encja kategorii

6. Konfiguracja autoryzacji:

- jwt-auth.guard.ts - guard JWT

- roles.guard.ts - guard ról

- roles.decorator.ts - dekorator ról

7. Konfiguracja Prisma:

- schema.prisma - schemat bazy danych

- prisma.service.ts - serwis Prisma

Proszę:

1. Używać rzeczywistych nazw plików jako nazw artefaktów (np. "users.service.ts" zamiast pełnej ścieżki)

2. Na początku każdego pliku dodać komentarz ze ścieżką (np. "// backend/src/modules/users/users.service.ts")

3. Na końcu dostarczyć listę referencyjną wszystkich wygenerowanych plików z ich pełnymi ścieżkami

Dzięki temu po pobraniu plików będę mógł/mogła użyć skryptu do automatycznego umieszczenia ich we właściwych katalogach.

```

**## Moduł Bazy Danych (PostgreSQL + Prisma)**

```

Proszę o skupienie się na module bazy danych aplikacji zarządzania projektami studenckimi.

Proszę przestrzegać następujących standardów kodowania:

1. Formatowanie:

- Używaj 2 spacji jako wcięcie (nie tabulatory)

- Używaj spójnego stylu w schemacie Prisma

- Dbaj o czytelność i organizację kodu

2. Nazewnictwo:

- Modele Prisma: PascalCase w liczbie pojedynczej (np. User, Project)

- Pola: camelCase (np. firstName, createdAt)

- Relacje: opisowe nazwy w camelCase (np. userProjects)

- Enumeracje: PascalCase (np. UserRole, ProjectStatus)

3. Struktura plików:

- Na początku każdego pliku dodaj komentarz ze ścieżką pliku w projekcie

- Używaj struktury zgodnej z dokumentacją projektu

- Organizuj schemat Prisma w logiczne sekcje z komentarzami

4. Dobre praktyki:

- Używaj odpowiednich typów danych

- Dodawaj indeksy dla pól używanych w zapytaniach wyszukiwania

- Implementuj relacje zgodnie z modelem danych

- Używaj komentarzy do opisania złożonych relacji lub ograniczeń

Zapoznaj się również z pełnym dokumentem Style Guide dostępnym w projekcie.

Dla tego modułu potrzebuję następujących plików:

1. Schemat Prisma:

- schema.prisma - pełny schemat bazy danych zawierający wszystkie modele

- migration\_script.sql - przykładowy skrypt migracji

2. Inicjalizacja danych:

- seed.ts - skrypt inicjalizacji danych testowych

- init.sql - skrypt inicjalizacyjny dla PostgreSQL

3. Obsługa bazy danych:

- prisma.module.ts - moduł Prisma

- prisma.service.ts - serwis Prisma

4. Przykłady operacji na bazie danych:

- database-operations.md - przykłady typowych operacji na bazie danych

Proszę:

1. Używać rzeczywistych nazw plików jako nazw artefaktów (np. "schema.prisma" zamiast pełnej ścieżki)

2. Na początku każdego pliku dodać komentarz ze ścieżką (np. "// backend/prisma/schema.prisma")

3. Na końcu dostarczyć listę referencyjną wszystkich wygenerowanych plików z ich pełnymi ścieżkami

Dzięki temu po pobraniu plików będę mógł/mogła użyć skryptu do automatycznego umieszczenia ich we właściwych katalogach.

```

**## Moduł Docker i Infrastruktura**

```

Proszę o skupienie się na module Docker i infrastruktury aplikacji zarządzania projektami studenckimi.

Proszę przestrzegać następujących standardów:

1. Formatowanie:

- Używaj 2 spacji jako wcięcie (nie tabulatory)

- Dbaj o czytelność i organizację plików konfiguracyjnych

- Używaj komentarzy do wyjaśnienia złożonych konfiguracji

2. Nazewnictwo:

- Serwisy: opisowe nazwy w kebab-case (np. frontend-app, api-server)

- Zmienne środowiskowe: UPPER\_SNAKE\_CASE

- Wolumeny i sieci: opisowe nazwy w kebab-case

3. Struktura plików:

- Na początku każdego pliku dodaj komentarz ze ścieżką pliku w projekcie

- Używaj struktury zgodnej z dokumentacją projektu

- Grupuj powiązane konfiguracje w odpowiednich katalogach

4. Dobre praktyki:

- Używaj wersji dla obrazów bazowych

- Minimalizuj rozmiar obrazów (wieloetapowe budowanie)

- Zabezpieczaj wrażliwe dane (używaj zmiennych środowiskowych)

- Konfiguruj odpowiednie warunki zdrowia i restartu kontenerów

Zapoznaj się również z pełnym dokumentem Style Guide dostępnym w projekcie.

Dla tego modułu potrzebuję następujących plików:

1. Konfiguracja Docker:

- docker-compose.yml - główny plik compose

- docker-compose.dev.yml - konfiguracja dla środowiska deweloperskiego

- docker-compose.prod.yml - konfiguracja dla środowiska produkcyjnego

2. Dockerfile dla frontendu:

- Dockerfile - obraz dla aplikacji Vue.js

- nginx.conf - konfiguracja Nginx dla frontendu

3. Dockerfile dla backendu:

- Dockerfile - obraz dla aplikacji NestJS

4. Konfiguracja bazy danych:

- Dockerfile - obraz dla PostgreSQL

- init.sql - skrypt inicjalizacyjny

5. Konfiguracja proxy:

- Dockerfile - obraz dla reverse proxy

- nginx.conf - konfiguracja Nginx jako proxy

- ssl-setup.md - instrukcja konfiguracji SSL

6. Skrypty pomocnicze:

- backup.sh - skrypt do tworzenia kopii zapasowych

- deploy.sh - skrypt wdrożeniowy

- setup-dev.sh - skrypt konfiguracji środowiska deweloperskiego

Proszę:

1. Używać rzeczywistych nazw plików jako nazw artefaktów (np. "docker-compose.yml" zamiast pełnej ścieżki)

2. Na początku każdego pliku dodać komentarz ze ścieżką (np. "// docker/docker-compose.yml")

3. Na końcu dostarczyć listę referencyjną wszystkich wygenerowanych plików z ich pełnymi ścieżkami

Dzięki temu po pobraniu plików będę mógł/mogła użyć skryptu do automatycznego umieszczenia ich we właściwych katalogach.

```

**## Moduł Dokumentacji**

```

Proszę o skupienie się na module dokumentacji aplikacji zarządzania projektami studenckimi.

Proszę przestrzegać następujących standardów:

1. Formatowanie:

- Używaj poprawnego formatowania Markdown

- Dbaj o spójny styl w całej dokumentacji

- Używaj nagłówków, list i kodów dla zwiększenia czytelności

2. Struktura:

- Używaj logicznej hierarchii nagłówków (h1, h2, h3, itd.)

- Grupuj powiązane informacje

- Dodawaj spisy treści do dłuższych dokumentów

- Używaj struktury zgodnej z dokumentacją projektu

3. Zawartość:

- Pisz jasno i zwięźle

- Używaj przykładów gdzie to możliwe

- Dodawaj diagramy lub zrzuty ekranu w razie potrzeby

- Uwzględniaj zarówno perspektywę techniczną jak i użytkownika

4. Dobre praktyki:

- Aktualizuj datę modyfikacji na górze dokumentu

- Dodawaj odnośniki do powiązanych dokumentów

- Używaj komentarzy do wyjaśnienia złożonych aspektów

Zapoznaj się również z pełnym dokumentem Style Guide dostępnym w projekcie.

Dla tego modułu potrzebuję następujących plików:

1. Dokumentacja główna:

- README.md - główny plik dokumentacji

- architecture.md - opis architektury systemu

- project-structure.md - struktura projektu

2. Dokumentacja API:

- README.md - wprowadzenie do API

- users.md - dokumentacja API użytkowników

- projects.md - dokumentacja API projektów

- categories.md - dokumentacja API kategorii

3. Instrukcje dla użytkowników:

- README.md - wprowadzenie dla użytkowników

- student.md - instrukcja dla studentów

- teacher.md - instrukcja dla nauczycieli

- admin.md - instrukcja dla administratorów

4. Instrukcje dla developerów:

- setup.md - konfiguracja środowiska

- coding-standards.md - standardy kodowania

- git-workflow.md - workflow Git

5. Instrukcje wdrożenia:

- local.md - wdrożenie lokalne

- production.md - wdrożenie produkcyjne

- ssl-setup.md - konfiguracja SSL

Proszę:

1. Używać rzeczywistych nazw plików jako nazw artefaktów (np. "README.md" zamiast pełnej ścieżki)

2. Na początku każdego pliku dodać komentarz ze ścieżką (np. "<!-- docs/README.md -->")

3. Na końcu dostarczyć listę referencyjną wszystkich wygenerowanych plików z ich pełnymi ścieżkami

Dzięki temu po pobraniu plików będę mógł/mogła użyć skryptu do automatycznego umieszczenia ich we właściwych katalogach.

```