Załącznik 4. Struktura plików projektu `p2p-chat`

1. Pliki w katalogu głównym

`Cargo.toml`	Główny plik konfiguracyjny projektu w ekosystemie Rust, definiujący nazwę pakietu, metadane, zależności oraz konfigurację kompilacji aplikacji `p2p-chat`.
`build.sh`	Skrypt powłoki automatyzujący proces budowy projektu: kompiluje interfejs webowy w katalogu `web-ui`, a następnie tworzy plik wykonywalny w języku Rust w trybie `release`.
`dev-web.sh`	Skrypt uruchamiający środowisko programistyczne interfejsu webowego, uruchamiając serwer Vite i przekierowując żądania API do warstwy serwerowej napisanej w Ruście.

2. Moduły źródłowe backendu w katalogu `src`

2.1. Warstwa aplikacji (`src/app`)

`src/app/mod.rs`	Główny moduł warstwy aplikacyjnej odpowiedzialny za przygotowanie środowiska uruchomieniowego, wybór trybu pracy (klient lub *mailbox*) oraz start głównych komponentów systemu.

`src/app/args.rs`	Definicja oraz obsługa argumentów linii poleceń aplikacji, oparta na bibliotece `clap`, obejmująca m.in. wybór portów i trybu działania.
`src/app/client.rs`	Logika uruchamiania aplikacji w trybie klienta: inicjalizacja warstwy sieciowej, magazynu danych, silnika synchronizacji, interfejsu tekstowego oraz serwera webowego.
`src/app/mailbox.rs`	Implementacja trybu pracy węzła *mailbox*, odpowiedzialnego za przyjmowanie, przechowywanie i udostępnianie wiadomości dla nieobecnych użytkowników.
`src/app/setup.rs`	Procedury przygotowania aplikacji, w tym inicjalizacja bazy `sled`, odczyt lub generowanie tożsamości użytkownika oraz konfiguracja szyfrowania przechowywanych danych.

2.2. Interfejs CLI (`src/cli`)

`src/cli/mod.rs`	Główny moduł interfejsu linii poleceń, integrujący definicje komend z resztą aplikacji.
`src/cli/commands.rs`	Definicje struktur komend CLI oraz głównej struktury `Node` reprezentującej rdzeń aplikacji, wykorzystywanej m.in. przez interfejs webowy do dostępu do stanu systemu.

2.3. Moduły kryptograficzne (`src/crypto`)

`src/crypto/mod.rs`	Punkt wejścia do warstwy kryptograficznej, eksportujący typy oraz funkcje używane w pozostałych częściach systemu.
`src/crypto/identity.rs`	Reprezentacja tożsamości kryptograficznej użytkownika, obejmująca klucze długoterminowe, operacje podpisu oraz szyfrowania end-to-end.
`src/crypto/hpke.rs`	Implementacja uproszczonego schematu hybrydowego szyfrowania w stylu HPKE, wykorzystywanego do bezpiecznego uzgadniania kluczy i szyfrowania wiadomości.
`src/crypto/storage.rs`	Funkcje odpowiedzialne za szyfrowanie i deszyfrowanie danych przechowywanych w lokalnej bazie, w tym generowanie i obsługę klucza szyfrującego magazyn.

2.4. Logowanie (`src/logging`)

`src/logging/mod.rs`	Główny moduł konfiguracji logowania, integrujący kolektor logów z systemem `tracing`.
`src/logging/buffer.rs`	Bufor w pamięci przechowujący ostatnie wpisy logów na potrzeby ich wyświetlania w interfejsie użytkownika.
`src/logging/collector.rs`	Implementacja kolektora logów, który odbiera zdarzenia z systemu `tracing` i przekazuje je do bufora oraz interfejsu tekstowego.

2.5. Główne punkty wejścia (`src/main.rs`, `src/mailbox.rs`)

`src/main.rs`	Główna funkcja programu uruchamiająca aplikację `p2p-chat`, inicjalizująca asynchroniczne środowisko Tokio oraz delegująca start do modułu `app`.
`src/mailbox.rs`	Dodatkowy punkt wejścia i logika wysokopoziomowa związana z obsługą węzła *mailbox* w kontekście warstwy sieciowej i synchronizacji.

2.6. Warstwa sieciowa wysokiego poziomu (`src/net`)

`src/net/mod.rs`	Moduł agregujący funkcje wysokopoziomowej warstwy `net`, odpowiedzialnej za operacje czatu i interakcję z DHT.
`src/net/chat.rs`	Funkcje związane z wysyłaniem i odbieraniem wiadomości czatu na poziomie logiki aplikacyjnej, w tym mapowanie wiadomości na komendy sieciowe.
`src/net/discovery.rs`	Mechanizmy odkrywania innych węzłów w sieci P2P przy użyciu DHT i mDNS, wykorzystywane do utrzymywania listy znanych peerów.
`src/net/mailbox.rs`	Operacje związane z komunikacją z węzłami *mailbox*: przekazywanie wiadomości do przechowania oraz ich odbieranie przez klienta.

2.7. Niskopoziomowa warstwa sieciowa `libp2p` (`src/network`)

`src/network/mod.rs`	Główny moduł warstwy `network` oparty na bibliotece `libp2p`, definiujący publiczny interfejs `NetworkLayer` oraz typy zdarzeń sieciowych.
`src/network/behaviour.rs`	Implementacja zachowania `libp2p Swarm` (*behaviour*) integrującego protokoły Noise, Kademlia, mDNS oraz request-response w spójną logikę sieciową.
`src/network/commands.rs`	Definicje komend kierowanych do warstwy sieciowej, takich jak wysyłanie wiadomości, zapytania o mailboxy czy aktualizacja informacji o peerach.
`src/network/handle.rs`	Uchwyt (*handle*) do komunikacji z warstwą sieciową z innych części aplikacji, umożliwiający asynchroniczne wysyłanie komend i odbiór odpowiedzi.
`src/network/message.rs`	Definicje typów wiadomości i odpowiedzi wymienianych pomiędzy warstwą sieciową a pozostałymi komponentami, w tym serializowanych struktur protokołu aplikacyjnego.
`src/network/handlers/mod.rs`	Moduł zbiorczy dla handlerów zdarzeń sieciowych, porządkujący obsługę różnych rodzajów zdarzeń pochodzących z `Swarm`.
`src/network/handlers/chat.rs`	Obsługa zdarzeń związanych z wiadomościami czatu na poziomie `libp2p`, w tym przyjmowanie i dekodowanie danych od innych peerów.
`src/network/handlers/discovery.rs`	Obsługa zdarzeń odkrywania peerów (m.in. z mDNS i DHT), aktualizująca lokalny widok topologii sieci.

`src/network/handlers/mailbox.rs`	Handler odpowiedzialny za komunikację z węzłami *mailbox* na poziomie protokołu sieciowego, w tym przyjmowanie zapytań oraz odpowiedzi.
`src/network/handlers/swarm.rs`	Obsługa zdarzeń generowanych przez `Swarm` (takich jak nawiązanie lub utrata połączenia), koordynująca przepływ informacji w warstwie sieciowej.
`src/network/layer/mod.rs`	Główny moduł implementujący `NetworkLayer`, łączący zachowanie `Swarm`, handlerów oraz kanały komunikacyjne z resztą systemu.
`src/network/layer/builder.rs`	Funkcje budujące instancję warstwy sieciowej, konfigurujące parametry `libp2p`, transport, protokoły i początkową topologię.
`src/network/layer/runtime.rs`	Kod zarządzający pętlą zdarzeń warstwy sieciowej, odpowiedzialny za uruchomienie i utrzymanie `Swarm` w osobnym zadaniu asynchronicznym.
`src/network/layer/state.rs`	Struktury przechowujące stan wewnętrzny warstwy sieciowej, w tym informacje o połączeniach, znanych peerach oraz trwających zapytaniach.
`src/network/layer/providers.rs`	Logika dotycząca obsługi dostawców usługi *mailbox* w DHT, w tym ranking, metryki wydajności i wybór najlepszych węzłów.

2.8. Warstwa przechowywania danych (`src/storage`)

`src/storage/mod.rs`	Moduł agregujący interfejsy oraz implementacje magazynów danych opartych na bazie `sled`.
`src/storage/friends.rs`	Magazyn przechowujący listę znajomych wraz z ich kluczami publicznymi używanymi do szyfrowania end-to-end.
`src/storage/history.rs`	Magazyn historii wiadomości, odpowiedzialny za zapisywanie, odczyt i filtrowanie konwersacji między peerami.
`src/storage/outbox.rs`	Magazyn wiadomości oczekujących na wysłanie (*outbox*), z którego korzysta silnik synchronizacji przy próbach dostarczenia wiadomości.
`src/storage/seen.rs`	Mechanizm śledzenia wiadomości oznaczonych jako przeczytane, wykorzystywany do aktualizacji statusów dostarczenia i odczytu.
`src/storage/known_mailboxes.rs`	Magazyn znanych węzłów *mailbox* wraz z metrykami jakości, służący do wyboru dostawców dla nowych wiadomości.
`src/storage/mailbox/mod.rs`	Moduł logiczny odpowiadający za lokalne przechowywanie wiadomości w roli *mailbox*, w tym struktury danych i interfejsy dostępu.
`src/storage/mailbox/operations.rs`	Implementacja operacji na lokalnym magazynie *mailbox*: zapisywanie, odczytywanie i usuwanie wiadomości przechowywanych dla innych użytkowników.

2.9. Silnik synchronizacji (`src/sync`)

`src/sync/mod.rs`	Główny moduł silnika synchronizacji odpowiedzialny za cykliczne próby dostarczenia wiadomości, odświeżanie informacji o mailboxach oraz reagowanie na zdarzenia sieciowe.
`src/sync/backoff.rs`	Implementacja strategii wycofania (*backoff*) dla ponawiania prób komunikacji z niedostępnymi węzłami, ograniczająca nadmierne obciążenie sieci.
`src/sync/retry.rs`	Funkcje wspierające ponawianie operacji sieciowych i zapisu, uwzględniające politykę błędów i limity prób.
`src/sync/engine/mod.rs`	Główny moduł silnika, definiujący strukturę `SyncEngine` oraz jego publiczny interfejs wykorzystywany przez resztę aplikacji.
`src/sync/engine/events.rs`	Definicje zdarzeń wewnętrznych silnika synchronizacji, które inicjują operacje takie jak natychmiastowe wysłanie wiadomości czy ponowne odkrywanie mailboxów.
`src/sync/engine/performance.rs`	Zbieranie i aktualizacja metryk wydajności dotyczących dostawców *mailbox*, wykorzystywanych przy wyliczaniu ich lokalnego rankingu.

2.10. Typy wspólne (`src/types.rs`)

`src/types.rs`	Definicje wspólnych typów domenowych, takich jak struktura wiadomości, statusy dostarczenia oraz aliasy typów wykorzystywane w różnych modułach systemu.

2.11. Interfejs tekstowy (TUI) (`src/ui`)

`src/ui/mod.rs`	Główny moduł interfejsu tekstowego, eksportujący funkcję uruchamiającą TUI oraz podstawowe typy reprezentujące zdarzenia i stany UI.
`src/ui/action.rs`	Definicje akcji użytkownika wykonywanych w interfejsie tekstowym, takich jak wysłanie wiadomości czy zmiana rozmówcy.
`src/ui/completers.rs`	Logika podpowiedzi (*completion*) dla wprowadzanych komend i identyfikatorów peerów, ułatwiająca obsługę z klawiatury.
`src/ui/event.rs`	Definicja zdarzeń interfejsu użytkownika, łączących wejście z klawiatury z reakcjami aplikacji.
`src/ui/log_entry.rs`	Struktury reprezentujące pojedyncze wpisy logów wyświetlane w trybie podglądu logów.
`src/ui/mode.rs`	Definicja dostępnych trybów interfejsu (m.in. tryb czatu i tryb logów) oraz logika przełączania się między nimi.
`src/ui/chat_mode/mod.rs`	Główny moduł trybu czatu, zarządzający historią wpisów użytkownika, podpowiedziami oraz bieżącym stanem pola wprowadzania.

src/ui/chat_mode/input.rs	Obsługa wejścia użytkownika w trybie czatu, w tym nawigacja po historii i akceptowanie podpowiedzi.
src/ui/chat_mode/render.rs	Renderowanie widoku czatu w terminalu, obejmujące listę wiadomości, pole wprowadzania oraz podpowiedzi.
src/ui/log_mode/mod.rs	Główny moduł trybu logów, odpowiedzialny za prezentację zebranych wpisów logów w interfejsie tekstowym.
src/ui/log_mode/render.rs	Implementacja renderowania widoku logów, w tym przewijania i formatowania wpisów.
src/ui/runner/mod.rs	Pętla główna interfejsu TUI, która spina odczyt zdarzeń, aktualizację stanu oraz renderowanie widoku.
src/ui/state/mod.rs	Główne struktury stanu interfejsu użytkownika, agregujące informacje o aktualnym widoku, zaznaczeniach i historii.
src/ui/state/chat.rs	Stan specyficzny dla trybu czatu, obejmujący m.in. aktualnie wybraną konwersację oraz bufor wprowadzanych wiadomości.
src/ui/state/input.rs	Stan pola wprowadzania tekstu, w tym bieżącą zawartość i kursor edycji.
src/ui/state/logs.rs	Stan widoku logów, obejmujący m.in. pozycję przewijania oraz filtrację wpisów.
src/ui/terminal/mod.rs	Moduł odpowiedzialny za integrację z biblioteką terminalową, inicjalizację trybu graficznego TUI oraz odtwarzanie ustawień po zakończeniu pracy.

`src/ui/terminal/events.rs`	Obsługa zdarzeń terminalowych, takich jak naciśnięcia klawiszy czy zmiana rozmiaru okna.
`src/ui/terminal/render.rs`	Funkcje renderujące główny układ interfejsu w terminalu, w tym okna czatu i panel logów.
`src/ui/terminal/controller.rs`	Warstwa kontrolera łącząca zdarzenia użytkownika z odpowiednimi akcjami i aktualizacjami stanu interfejsu.
`src/ui/terminal/lifecycle.rs`	Procedury cyklu życia interfejsu terminalowego, odpowiadające za poprawną inicjalizację oraz sprzątanie zasobów.

2.12. Warstwa webowa backendu (`src/web`)

`src/web/mod.rs`	Główny moduł serwera webowego opartego na frameworku `Axum`, odpowiedzialny za uruchomienie serwera HTTP, integrację API, WebSocketów oraz serwowanie zasobów statycznych.
`src/web/api.rs`	Implementacja REST API udostępnianego interfejsowi webowemu, obejmująca m.in. pobieranie listy znajomych, historii rozmów oraz statusu systemu.
`src/web/websocket.rs`	Obsługa połączeń WebSocket, wykorzystywana do przesyłania powiadomień o nowych wiadomościach i zmianach stanu w czasie rzeczywistym do interfejsu przeglądarkowego.

3. Pliki źródłowe interfejsu webowego (`web-ui`)

3.1. Konfiguracja i zależności

`web-ui/package.json`	Główny plik konfiguracyjny projektu frontendowego, definiujący zależności NPM, skrypty uruchomieniowe oraz metadane aplikacji.
`web-ui/package-lock.json`	Zablokowany opis drzewa zależności NPM zapewniający powtarzalność instalacji bibliotek frontendowych.
`web-ui/env.d.ts`	Deklaracje typów TypeScript dla zmiennych środowiskowych oraz rozszerzeń używanych przez narzędzia budujące projekt.
`web-ui/tsconfig.json`	Główny plik konfiguracji kompilatora TypeScript dla całego projektu frontendowego.
`web-ui/tsconfig.app.json`	Konfiguracja TypeScript specyficzna dla kodu aplikacji (bez testów i narzędzi pomocniczych).
`web-ui/tsconfig.node.json`	Konfiguracja TypeScript dla plików uruchamianych w środowisku Node.js (np. skrypty budujące aplikację).
`web-ui/vite.config.ts`	Konfiguracja bundlera Vite odpowiedzialnego za budowanie i serwowanie aplikacji Vue.

3.2. Główne moduły aplikacji

`web-ui/src/main.ts`	Główny punkt wejścia aplikacji Vue, inicjalizujący instancję aplikacji, konfigurujący router oraz magazyn stanu Pinia, a następnie montujący aplikację w drzewie DOM.
`web-ui/src/App.vue`	Komponent korzeniowy aplikacji Vue, renderujący bieżący widok na podstawie konfiguracji routera.
`web-ui/src/router/index.ts`	Konfiguracja nawigacji po stronie klienta (Vue Router), definiująca dostępne ścieżki i przypisane im widoki.
`web-ui/src/peerBranding.ts`	Stałe i funkcje związane z wyświetlaniem nazwy oraz identyfikatora lokalnego węzła w interfejsie użytkownika.

3.3. Warstwa API i komunikacja z backendem

`web-ui/src/api/types.ts`	Definicje typów TypeScript odzwierciedlających struktury danych zwracane przez backend (m.in. wiadomości, znajomi, status systemu).
`web-ui/src/api/client.ts`	Klient HTTP odpowiedzialny za wywoływanie punktów końcowych (endpointów) REST API warstwy serwerowej, w tym obsługę błędów i mapowanie odpowiedzi na typy domenowe.
`web-ui/src/api/websocket.ts`	Moduł zarządzający połączeniem WebSocket z backendem, wykorzystywany do odbierania w czasie rzeczywistym informacji o nowych wiadomościach i zmianach stanu.

3.4. Magazyny stanu (stores)

`web-ui/src/stores/identity.ts`	Magazyn stanu zawierający informacje o lokalnej tożsamości użytkownika, wykorzystywany do prezentowania danych w UI oraz wysyłania żądań do backendu.
`web-ui/src/stores/friends.ts`	Magazyn stanu listy znajomych oraz ich statusów (online/offline), synchronizowany z backendem.
`web-ui/src/stores/conversations.ts`	Magazyn stanu konwersacji i wiadomości, odpowiedzialny za lokalne buforowanie treści czatu wyświetlanej w interfejsie.

3.5. Widoki (`web-ui/src/views`)

`MessengerView.vue`	Główny widok komunikatora prezentujący układ interfejsu aplikacji: listę rozmów, okno czatu oraz panele informacyjne.

3.6. Komponenty (`web-ui/src/components`)

`DraggableWindow.vue`	Komponent prezentujący okno w stylu klasycznych interfejsów, które można przeciągać w obrębie ekranu.
`FramedAvatar.vue`	Komponent odpowiedzialny za wyświetlanie awataru użytkownika w ozdobnej ramce.

`StatusPanel.vue`	Panel prezentujący podstawowe informacje o stanie systemu i połączenia, w tym status online/offline.
`AddFriendModal.vue`	Okno modalne umożliwiające dodanie nowego znajomego na podstawie jego identyfikatora.
`InfoPanel.vue`	Panel informacyjny wyświetlający szczegóły dotyczące aktualnie wybranego rozmówcy oraz systemu.
`ChatList.vue`	Lista dostępnych konwersacji oraz skrótowych informacji o ostatnich wiadomościach.
`ChatWindow.vue`	Główne okno czatu prezentujące przebieg rozmowy oraz pole wprowadzania wiadomości.

3.7. Zasoby statyczne i style

`web-ui/src/assets/main.css`	Główny arkusz stylów aplikacji frontendowej, definiujący ogólny wygląd interfejsu.
`web-ui/src/assets/base.css`	Podstawowe style resetujące oraz wspólne reguły typograficzne wykorzystywane w całej aplikacji.
`web-ui/src/assets/logo.svg`	Wektorowa grafika logo aplikacji wykorzystywana w interfejsie.
`web-ui/src/assets/*.gif`	Zestaw animowanych grafik w formacie GIF (pliki `1.gif`–`8.gif`) używanych jako elementy dekoracyjne interfejsu oraz tła.
`web-ui/src/types/gradient-gl.d.ts`	Deklaracje typów dla niestandardowych modułów związanych z efektami graficznymi tła, używane przez część warstwy prezentacji.