Załącznik 5. Omówienie pliku `Cargo.toml` projektu `p2p-chat`

1. Struktura pliku `Cargo.toml`

Plik `Cargo.toml` stanowi centralny plik konfiguracyjny projektu w ekosystemie Rust. Określa podstawowe metadane pakietu (nazwa, wersja, edycja języka), a także pełną listę zależności zewnętrznych wraz z wymaganymi wersjami oraz włączonymi funkcjami (*features*). W projekcie `p2p-chat` plik ten ma charakter monolityczny – definiuje pojedynczy pakiet binarny odpowiedzialny zarówno za logikę węzła sieci P2P, jak i wbudowany serwer HTTP z interfejsem webowym.

2. Sekcja `[package]`

`name = "p2p-chat"`	Nazwa pakietu wskazuje na główny cel projektu, którym jest implementacja komunikatora typu *peer-to-peer*. Nazwa ta jest wykorzystywana m.in. przy budowaniu, dystrybucji oraz w katalogu zależności ekosystemu Rust.
`version = "0.1.0"`	Wersja pakietu w formacie semantycznym, odzwierciedlająca wczesny etap rozwoju (wersja `0.x`) oraz brak stabilnego API zewnętrznego.
`edition = "2021"`	Deklaracja korzystania z edycji Rust 2021, umożliwiającej wykorzystanie najnowszych funkcji języka i usprawnień kompilatora, w tym uproszczonej obsługi modułów i nowszych konstrukcji składniowych.

3. Zależności sieciowe i asynchroniczne

`libp2p = { version = "0.53", features = [...] }`	Biblioteka implementująca stos protokołów *peer-to-peer* wykorzystywany w projekcie. Włączone funkcje (`tcp`, `tokio`, `noise`, `yamux`, `mdns`, `kad`, `request-response`, `macros`, `serde`, `ping`) aktywują odpowiednio transport TCP, integrację z Tokio, szyfrowanie Noise, multipleksację strumieni, odkrywanie węzłów w sieci lokalnej, rozproszoną tablicę haszującą Kademlia, protokół żądanie–odpowiedź, wsparcie makr i serializacji oraz mechanizm `ping` do monitorowania żywotności połączeń.
`tokio = { version = "1.0", features = ["full"] }`	Asynchroniczne środowisko uruchomieniowe (*runtime*) wykorzystywane do obsługi współbieżnych zadań sieciowych i dyskowych. Włączenie `"full"` aktywuje pełen zestaw komponentów Tokio (timery, kanały, gniazda sieciowe, itp.), co upraszcza implementację złożonych przepływów asynchronicznych.
`futures = "0.3"`	Zestaw narzędzi do pracy z abstrakcjami *Future* i strumieniami (*Streams*), uzupełniający funkcjonalność Tokio i ułatwiający kompozycję operacji asynchronicznych.
`getrandom = "0.2"`	Biblioteka zapewniająca jednolity interfejs do systemowych generatorów liczb losowych, wykorzystywana pośrednio m.in. przez mechanizmy kryptograficzne.

4. Zależności kryptograficzne i bezpieczeństwa

`x25519-dalek = { version = "2.0", features = ["static_secrets"] }`	Implementacja operacji na krzywej eliptycznej Curve25519 wykorzystywana do realizacji wymiany kluczy X25519. Funkcja `static_secrets` włącza obsługę długoterminowych sekretów, wykorzystywanych przy konstruowaniu tożsamości kryptograficznych użytkowników.
`chacha20poly1305 = "0.10"`	Biblioteka implementująca szyfrowanie uwierzytelnione AEAD w wariancie ChaCha20-Poly1305, wykorzystywana do zapewnienia poufności i integralności treści wiadomości.
`argon2 = "0.5"`	Implementacja funkcji Argon2 służącej do bezpiecznego wyprowadzania kluczy z haseł (KDF), wykorzystywana przy ochronie materiału kluczowego zapisywanego w magazynie.
`sha2 = "0.10"`	Zestaw funkcji skrótu z rodziny SHA-2, używany do obliczania kryptograficznych skrótów danych (np. identyfikatorów, odcisków kluczy).
`rand = "0.8"`	Ogólna biblioteka generowania liczb pseudolosowych, wykorzystywana przy generowaniu identyfikatorów, nonce'ów oraz innych losowych wartości.
`rand_core = { version = "0.6", features = ["getrandom"] }`	Warstwa bazowa dla generatorów losowych, integrowana z `getrandom`; zapewnia spójne źródło entropii dla komponentów kryptograficznych.

`base64 = "0.22"`	Biblioteka kodowania i dekodowania danych w formacie Base64, używana m.in. do reprezentacji binarnych kluczy w postaci tekstowej.
`hex = "0.4"`	Obsługa reprezentacji danych w formacie szesnastkowym, przydatna do logowania i prezentacji odcisków kluczy oraz identyfikatorów.

5. Serializacja, identyfikatory i czas

`serde = { version = "1.0", features = ["derive"] }`	Główna biblioteka serializacji i deserializacji struktur danych, wykorzystywana w całym projekcie do zapisu konfiguracji, komunikatów sieciowych oraz danych w bazie. Funkcja `derive` umożliwia automatyczne generowanie implementacji `Serialize` i `Deserialize`.
`serde_json = "1.0"`	Obsługa formatu JSON, wykorzystywana m.in. do wymiany danych w interfejsie HTTP oraz przy zapisie konfiguracji.
`uuid = { version = "1.0", features = ["v4", "serde"] }`	Biblioteka generująca losowe identyfikatory UUID w wersji 4, wykorzystywane do jednoznacznego oznaczania wiadomości i innych obiektów domenowych. Włączona integracja z `serde` umożliwia wygodną serializację.
`chrono = { version = "0.4", features = ["serde"] }`	Biblioteka obsługi czasu i dat, używana do oznaczania znaczników czasowych wiadomości oraz innych zdarzeń, z możliwością bezpośredniej serializacji do formatów wymiany danych.

6. Interfejs linii poleceń, logowanie i obsługa błędów

`clap = { version = "4.0", features = ["derive"] }`	Biblioteka służąca do definiowania i parsowania argumentów linii poleceń. W projekcie wykorzystywana do konfiguracji trybu pracy aplikacji oraz parametrów takich jak porty nasłuchu. Funkcja `derive` pozwala deklaratywnie opisać strukturę argumentów.
`tracing = "0.1"`	Nowoczesny system logowania i śledzenia zdarzeń, stosowany do rejestrowania informacji diagnostycznych w sposób strukturalny.
`tracing-subscriber = { version = "0.3", features = ["env-filter"] }`	Warstwa subskrybenta dla `tracing`, odpowiedzialna za formatowanie i wypisywanie logów. Funkcja `env-filter` pozwala kontrolować poziom logowania za pomocą zmiennych środowiskowych.
`anyhow = "1.0"`	Biblioteka upraszczająca propagację błędów w kodzie aplikacyjnym poprzez ujednolicony typ błędu z obsługą kontekstu. Wykorzystywana wszędzie tam, gdzie istotniejsza jest ergonomia niż precyzyjna klasyfikacja błędów.
`thiserror = "1.0"`	Biblioteka wspierająca definiowanie własnych typów błędów w sposób deklaratywny, z czytelnymi komunikatami i integracją z systemem `std::error::Error`.
`colored = "2.0"`	Narzędzie do kolorowania tekstu w terminalu, wykorzystywane do zwiększenia czytelności komunikatów w interfejsie tekstowym i logach.

`unicode-width = "0.1"`	Biblioteka obliczająca szerokość znaków Unicode, przydatna przy wyrównywaniu tekstu w interfejsie terminalowym, zwłaszcza w obecności znaków wielobajtowych.

7. Interfejs tekstowy (TUI) i obsługa terminala

`rustyline = "14.0"`	Biblioteka dostarczająca funkcjonalność edycji linii poleceń z historią i skrótami klawiaturowymi, wykorzystywana w początkowych eksperymentach z interfejsem tekstowym.
`reedline = "0.28"`	Nowocześniejsza biblioteka do obsługi linii poleceń w terminalu, oferująca m.in. lepsze wsparcie dla podpowiedzi, historii i pracy z kolorami.
`crossterm = "0.27"`	Biblioteka do niskopoziomowej obsługi terminala w sposób przenośny między systemami, używana do realizacji interfejsu tekstowego opartego na TUI (rysowanie okien, obsługa klawiatury).

8. Warstwa webowa i serwer HTTP

`axum = { version = "0.7", features = ["ws", "macros"] }`	Lekki framework webowy wykorzystywany do implementacji serwera HTTP oraz obsługi WebSocketów. Funkcje `"ws"` i `"macros"` zapewniają wsparcie dla dwukierunkowej komunikacji w czasie rzeczywistym oraz deklaratywnego definiowania tras.
`tower = "0.4"`	Biblioteka dostarczająca abstrakcje usług (*Service*) i middleware, będąca fundamentem, na którym opiera się Axum.
`tower-http = { version = "0.5", features = ["fs", "cors"] }`	Zbiór komponentów HTTP takich jak serwowanie plików statycznych (`fs`) oraz obsługa nagłówków CORS, wykorzystywany do udostępniania zasobów interfejsu webowego i konfiguracji dostępu z przeglądarki.
`rust-embed = "8.0"`	Biblioteka umożliwiająca dołączanie plików statycznych (np. zbudowanego frontendu Vue) bezpośrednio do pliku wykonywalnego, co ułatwia dystrybucję aplikacji bez dodatkowego katalogu z zasobami.
`mime_guess = "2.0"`	Narzędzie do zgadywania typu MIME na podstawie rozszerzenia pliku, wykorzystywane przy serwowaniu zasobów statycznych (HTML, CSS, JavaScript, grafiki) z wbudowanego serwera HTTP.