Protocol: Technical Steps, Design Decisions, Failures and Solutions

1. Layered structure

- **HTTP layer:** a custom lightweight HTTP server (provided, slightly modified) built on TcpListener/TcpClient, parsing raw HTTP requests (methods, headers, body) and dispatching to endpoint handlers via an IEndpoint interface.
- **Business logic layer:** three handlers (UserHandler, ExerciseHandler, TournamentHandler) encapsulate core rules (user registration, token-based authentication, exercise validation, tournament and Elo logic).
- **Data access layer:** repository classes (UserRepo, ExerciseRepo, TournamentRepo) implement IUserRepo, IExerciseRepo, ITournamentRepo interfaces using parameterized raw SQL (no ORM) against PostgreSQL via Npgsql.
- **Models:** plain C# classes for User, Exercise, Tournament plus enums for exercise types, tournament status and rank.

2. HTTP Server Implementation

- **Custom HTTP parsing:** Modified provided server code and built HttpRequest to read request-line, headers (case-insensitive), content length, and body. Designed HttpResponse to write status line, headers and JSON body.
- **Endpoint registry & routing:** In Program.cs, registered endpoints under keys (“users”, “sessions”, “exercises”, etc.). HttpProcessor splits the path on / to select the appropriate IEndpoint, allowing future extension.
- **Logging:** Used provided StreamTracer class wrapping a StreamWriter to write both to console and an on-disk log file, making all incoming requests, responses and internal debug messages traceable.

3. Security & Authentication

- **Token generation:** Upon registration (UserRepo.RegisterUser), a simple token of form "{username}-sebToken" is generated and stored in the database. Login returns user object with Token property; clients must send Authorization: Basic <token>.
- **Token validation:** Every protected endpoint (e.g., posting exercises, running tournaments) checks for the header, extracts the token, and calls UserRepo.GetUserByToken to authenticate. Unauthorized requests immediately return 401.

4. Data Persistence

- **PostgreSQL + Npgsql:** Used parameterized SQL commands to prevent injection and to meet “no ORM” requirement. Created tables for Users, Exercises, Tournaments, and a join table for tournament participants.
- **Connection management:** Each repository method using a fresh NpgsqlConnection and disposing it promptly to avoid connection leaks.

5. Business Logic & Game Rules

- **Exercise validation:** ExerciseHandler.AddExercise throws if count or duration ≤ 0, enforcing input sanity before persisting.
- **Tournament flow:** TournamentHandler.CreateTournament initializes a new 2-minute contest. FinishTournament retrieves all exercises in window, sums counts per user, determines winner(s) or draw, updates Elo (+2 for winner, –1 for losers, +1 each on draw), and persists tournament status.

6. Unique feature

- **User ranking system:** Each user has an Elo-style Rank stored on their profile; updated after every tournament finish. Winners gain +2 points, losers get -1, and ties award +1 to each, making Rank the definitive leaderboard metric.
- **Multiple exercise types:** Originally supported push-ups only; created ExerciseType enum to include burpees (and any future types). Handlers, repositories, and database schema updated to store and query by type, enabling richer contests and future extensibility.

7. Unit Test Design

1. ExerciseHandlerTests (5 tests)

- **Valid path:** ensures AddExercise calls into the repository exactly once for each exercise type.
- **Boundary conditions:** verifies that non-positive Count or Duration throw exceptions to prevent invalid entries.
- **Type handling:** confirms that both push-ups and burpees are accepted and persisted correctly.
- **Retrieval:** tests that GetExercisesByUserId returns the expected list, including mixed types.

2. TournamentHandlerTests (6 tests)

- **Winner Elo update:** in a two-player tournament, finishing should grant +2 Elo to winner and -1 to loser.
- **Multi-participant draw:** tests draw logic (+1 Elo each) with mixed-type submissions.
- **Single-participant edge case:** ensures the handler gracefully processes tournaments with one entry.
- **Repository interactions:** mocks verify that tournament status and Elo updates are persisted correctly.

3. UserHandlerTests (8 tests)

- **Registration:** registering a new user inserts exactly once and throws if the username already exists.
- **Login:** valid credentials return a User; invalid ones return null.
- **Rank initialization:** new users start with default Rank (Newbie).
- **Profile update:** updating Name, Bio, Image calls the repo's UpdateUser.

- **Exercise library update:** ensures UpdateUserExercises replaces the user's exercise list.

4. Reason for choosing these tests

- All core game rules, authentication, and data checks are in the BusinessLogic layer. A fault here could allow malformed data, break tournament scoring, or expose security holes.