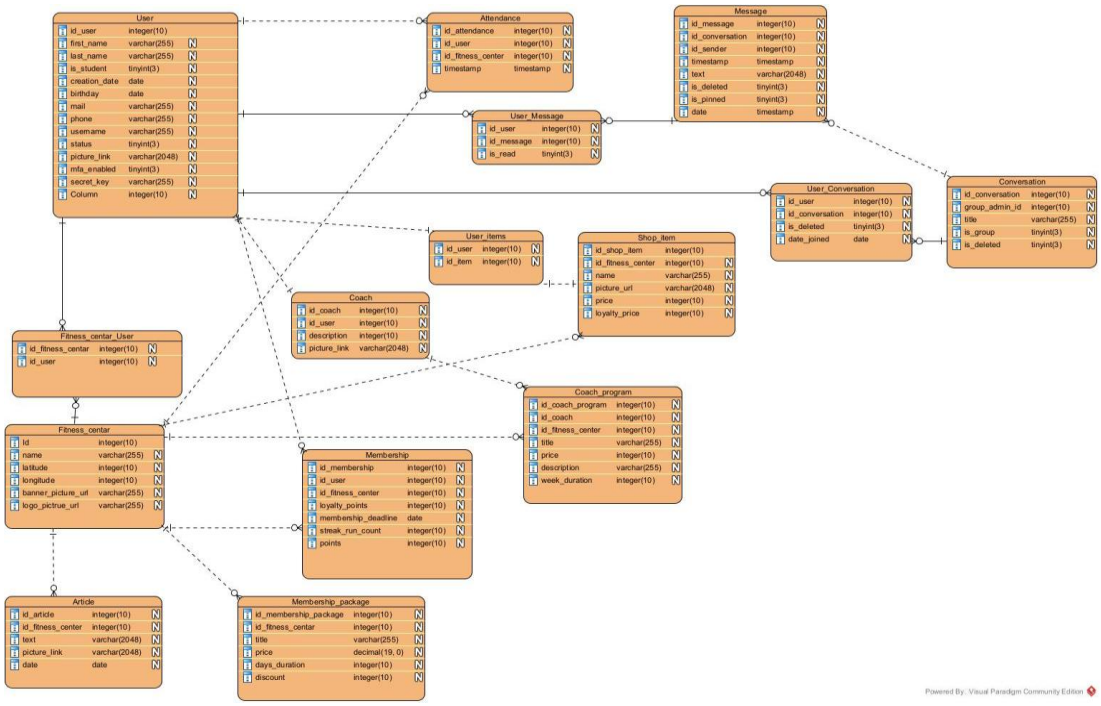
Fitness Centar

.NET Backend + Android Kotlin application

Lovro Lešić

April – July 2025.

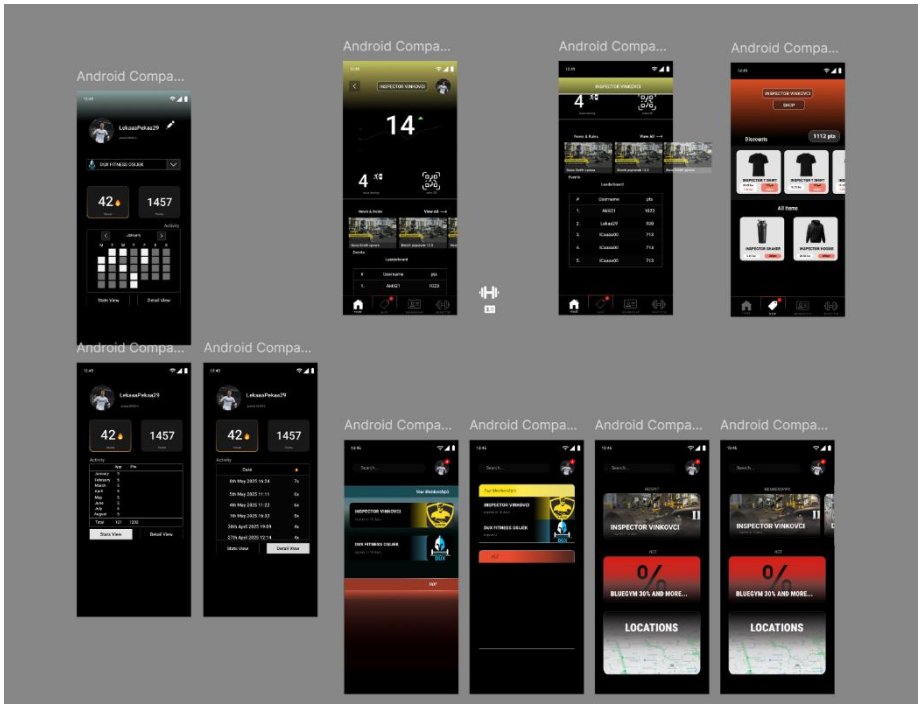


Functional Requirements for Fitness Center Application:

- **User Authentication and Registration: Users can create a new user account, log in, and the system generates and stores an authentication token with the user's email address as reference.**
- **Authentication Token Renewal: The token can be renewed to continue authentication.**
- **Adding Fitness Center: Enables entry of a new fitness center into the system.**
- **Adding Fitness Center Subscription Packages: Users can add, delete, and update subscription packages, and discounts can be added or removed from them.**
- **Updating User Membership: Users can update their membership, track expiration dates, collect points and track the number of consecutive visits, with packages enabling membership extension.**

- **Adding Trainer: Trainers can add their profile with information about expertise, knowledge, and training packages.**
- **Adding Training Program Packages: Trainers can add and update training packages with goals, prices, and duration.**
- **Adding Article: Fitness centers can add articles with text, images, and dates for user notifications.**
- **Adding Attendance: With each visit to the center, users automatically record their attendance in the system.**
- **Displaying User Attendance: Users can view their attendance on a monthly calendar and visualized attendance list.**
- **Displaying Fitness Center Attendance: Fitness centers display the number of visits from their users through graphical representation, showing the center's popularity.**
- **Attendance Points Table: Each fitness center tracks user points based on attendance, with a display of top users on the main screen.**
- **Fitness Center Store: Each center has a sales offering with items for sale.**
- **Adding Items to Store: Centers can add items to their store with prices in euros and points.**
- **Displaying User's Purchased Items: Users can view the history of purchased items from all centers.**
- **Chat Between Users: Enables sending messages between users or in user groups.**
- **Adding Message: Users can send messages to groups or direct chat.**
- **Marking Message as Read: Messages in the system are marked as read after being opened.**
- **Fitness Center Map: Displays all center locations in the system and enables navigation to the selected center.**
- **Displaying Nearby Centers: The main screen shows the 3 nearest fitness centers with distance data and a button to display on the map.**
- **Displaying Fitness Centers with Package Discounts: The main screen shows gyms that offer discounts on their packages.**
- **Displaying Current Center Occupancy: The center's main screen shows the number of users who have entered the center in the last 1.5 hours.**
- **Displaying Number of Users Finishing Workout: Shows the number of users finishing their workout (attendance age between 1h and 1.5h).**

- **Subscription Packages Screen: The center screen displays all available subscription packages with purchase option.**
- **Profile Screen: User profile displays user information and statistics related to fitness center visits. Ability to change profile picture.**



Development Comments and Notes

- Architecture
- Async
- ClaimTypes
- IConfiguration
- DTO
- Bearer
- JWT Token
- Retrofit
- RetrofitBuilder
- Screen Design
- Repositories
- ViewModel

- UIState
- Cloudflare
- User Message Exchange
- Database
- Application Design Order

Fitness Center Application Development Notes

Architecture

The backend is built on the Controller-Repository-Service pattern principle, which enables clean separation of logic and easier code maintenance. The Android application uses a modern structure: Retrofit for network communication, Repository pattern, ViewModel, UIState management, and composite screens.

Async Approach

The backend uses asynchronous functions so the database can make asynchronous calls. This is particularly important because in production, latency to a server located in a foreign country can significantly impact performance and cost us.

ClaimTypes

In the controller, through ClaimTypes from the JWT token, we can extract selected stored data. I store the email as an indicator of which user is sending the request, and also the token that indicates whether the user is authenticated.

IConfiguration

We use the built-in interface for reading the appsettings.json file where we store the admin email to which we only allow certain operations. For this MVP version, this serves as a substitute for the role-based approach that would otherwise be added in ASP.NET Identity roles and create multiple different users by permissions.

DTO (Data Transfer Objects)

An object that can be serialized because it's made of simple data or other DTO objects following the same rule. In the main rule, it must support data for adding the model it represents and also data we'll use for presentation (which are mostly not different). It becomes very useful when we need unexpected data for presentation or when we combine multiple ones because it simplifies the logic of GET requests in screen logic.

Bearer Token

Bearer token is an authentication method where the client sends a token in the HTTP header (Authorization: Bearer <token>). The server can then verify the user's identity based on that token.

JWT Token

A token that the user and database store and thus recognize the user's security access. Tokens optionally have their expiration date after which the database no longer accepts the user as logged in. A user who is already logged in should have automatic token renewal.

Retrofit

Retrofit is an HTTP client for Android that enables elegant REST API calls. It's important that Retrofit has signatures of database endpoints and that we ensure our models on the Android application and those on the backend are supported for communication in every endpoint - it's important to watch data types, naming, and nullability.

RetrofitBuilder

Builds a Retrofit instance and we add an interceptor that signs every call to the database with a Bearer token. This ensures token exchange between the backend and application, thus solving the database's security question for requests and gaining access to endpoints.

Screen Design

Navbar, possible screens, and data are preloaded by not using NavController, which gives the impression of faster loading and transitions between screens that would be present while performing calls and drawing complex screens. I didn't use strings.xml for storing colors and strings because in this MVP version I prioritized fulfilling Figma requirements and deadlines over clean code principles and the "single point of truth" principle. The backgrounds of the two main screens are made through animated linear gradient where the first color in the gradient is animated and develops through selected shades and strength in a repeating cycle of a selected interval.

Repository

They handle communication with Retrofit endpoints. For GET operations, mostly simple implementation, while sometimes you need to use multiple Android repositories to fulfill one request. Also using Flow to automate repetition of some GET requests every selected time interval. In POST and PUT operations, repositories must fulfill and prepare the object being sent. For sending messages, I used Flow instead of WebSockets due to project time constraints.

ViewModel

We use suspend functions and launchModel coroutines so the application can send requests to the database through other processor threads and thus not lose application responsiveness while fulfilling requests. Due to Kotlin development, I think coroutine design is simplified and there's less danger of memory leaks and "callback hell" application crashes.

UiState

Screen data containing different object lists are all stored in the screen's separate UiState which is refreshed by ViewModel on call or automatically. Through lifecycle remember on

UiState data in the screen, we enable listening to changes on the entire UiState, which refreshes the screen with every change, and thus in a simple way our screen responsively communicates with the database.

Cloudflare Images

For application images, I used Cloudflare Images which enables storing images in URL link form. On the mobile application, specifically for profile pictures, we send a request to Cloudflare along with the image, API key, and signature, and we get back information whether it was successfully stored and the image's URL link, which we then send to the backend so the database can enter the new image URL. I manually added other application images to Cloudflare and recorded URLs in the database.

User Message Exchange

Due to the decision that there's a possibility of messaging in groups, the logic of the model and thus the backend becomes more demanding because multiple messages are no longer just communication between sender and receiver. In this case, messages only store the sender and are connected to some conversation, which logically represents the receiver or more precisely the message destination. Members must also be connected to the conversation to be counted among users who see messages. An additional problem is the decision to have the possibility of marking messages as read, which on top of everything puts another abstraction - the message has a separate connection to each conversation user so we can individually manage the read status data for each user, because otherwise any user could switch that flag for everyone.

Database

The database is made in SQLite through DBeaver, which then enabled automatic model pulling into .NET database (Database First approach). I added newer models directly to DataContext and created .NET migrations.

Application Design Order

- **ERA MODEL, INITIAL FUNCTIONAL REQUIREMENTS** – early April
- **FIGMA DESIGN** – April
- **SECURITY .NET** – April
- **SQLITE AND .NET DATACONTEXT** – April
- **BACKEND EXECUTED CONTROLLERS** – April-May
- **ANDROID ENDPOINT APP** – May-June
- **ANDROID UX APP** – June
- **UX APP INTEGRATION INTO ENDPOINT APP** – end of June

FitnessCenter **GET/api/FitnessCenter** **GET/api/FitnessCenter/{fitnessCenterId}** **GET/api/FitnessCenter/ClosestFitnessCentars** **GET/api/FitnessCenter/coaches/{fitnessCentarId}** **GET/api/FitnessCenter/PromoFitnessCentars** **POST/api/FitnessCenter/AddFitnessCenter** **PUT/api/FitnessCenter/UpdateFitnessCentar/{fitnessCentarId}** **DELETE/api/FitnessCenter/DeleteFitnessCentar/{fitnessCentarId}**
Membership **GET/api/Membership/user** **GET/api/Membership/user/{fitnessCenterId}** **GET/api/Membership/FitnessCenter/{fitnessCenterId}** **GET/api/Membership/FitnessCenter/Leaderboard/{fitnessCenterId}** **POST/api/Membership/AddMembership** **POST/api/Membership/AddMembershipPackage** **POST/api/Membership/UpdateMembership** **GET/api/Membership/MembershipPackage/{membershipPackageId}** **GET/api/Membership/MembershipPackages/{fitnessCentarId}** **PUT/api/Membership/UpdateMembership/{membershipId}** **DELETE/api/Membership/DeleteMembership/{membershipId}**
Coach **GET/{coachId}** **GET/coachProgram/{coachProgramId}** **GET/coach/programs/{coachId}** **POST/api/Coach/AddCoach** **POST/api/Coach/AddCoachProgram** **PUT/api/Coach/UpdateCoach/{coachId}** **DELETE/api/Coach/DeleteCoach/{coachId}**
Attendance **GET/api/Attendance/users** **GET/api/Attendance/users/{fitnessCenterId}** **GET/api/Attendance/fitnesscenters/{fitnessCenterId}** **GET/api/Attendance/fitnesscenters/recent/{fitnessCenterId}** **GET/api/Attendance/fitnesscenters/leaving/{fitnessCenterId}** **POST/api/Attendance/AddAttendance** **PUT/api/Attendance/UpdateAttendance/{attendanceId}**

DELETE/api/Attendance/DeleteAttendance/{attendanceId}
Article **GET/api/Article/articles/{fitnessCenterId}** **POST/api/Article/AddArticle** **PUT/api/Article/UpdateArticle/{articleId}** **DELETE/api/Article/DeleteArticle/{articleId}**
Account **GET/api/Account/{userId}** **POST/api/Account/Register** **POST/api/Account/login** **POST/api/Account/UpdateUser**
Message **GET/api/Conversation/{conversationId}** **GET/api/Conversation/chats** **GET/api/Conversation/{conversationId}/participants** **GET/api/Conversation/{conversationId}/search** **POST/api/Conversation/message/send/{recipientId}** **POST/api/Conversation/message/markAsRead** **DELETE/api/Conversation/message/remove/{messageId}** **DELETE/api/Conversation/remove/{conversationId}** **PUT/api/Conversation/message/update/{messageId}** **GET/api/Conversation/{conversationId}/message/unreadCount** **GET/api/Conversation/unreadCount** **POST/api/Conversation/createGroup** **POST/api/Conversation/{conversationId}/addParticipant** **POST/api/Conversation/{conversationId}/removeParticipant** **POST/api/Conversation/{conversationId}/leaveGroup** **POST/api/Conversation/{messageId}/pinMessage**
Shop **GET/api/Shop/items/{fitnessCenterId}** **GET/api/Shop/items/users** **GET/api/Shop/item/{shopItemId}** **POST/api/Shop/BuyShopItem/{shopItemId}** **POST/api/Shop/AddShopItem** **PUT/api/Shop/UpdateShopItem/{shopItemId}** **DELETE/api/Shop/DeleteShopItem/{shopItemId}**