Exvi Fitness

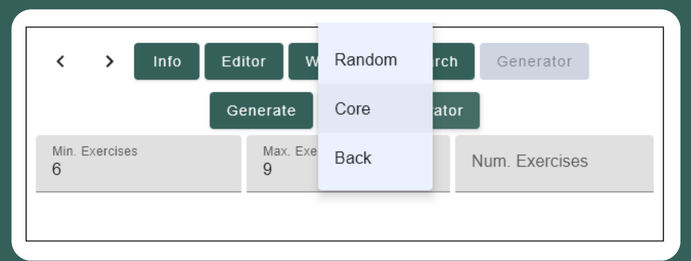
CS 30 AP Capstone Project - Callum Mackenzie

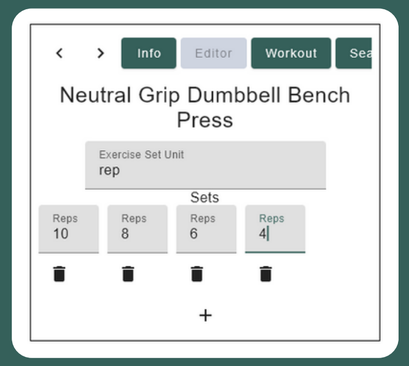
Problem

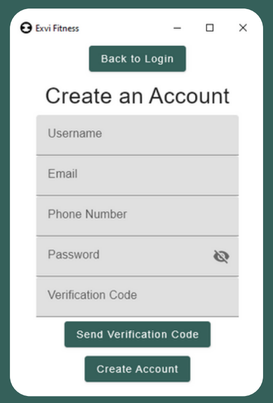
Large collections of exercises are hard to come by, and even harder to search through and compile into a workout program. And even creating said workouts is challenging in itself; a high level of physiological knowledge is required to create an efficient workout. How can an individual **create**, **track**, **share**, and **progress** through workouts **digitally** with ease?

Idea

Exvi Fitness provides a collection of exercises to search through and compile into unique workout programs. Furthermore, it allows you to **generate** unique **workouts** from preset generators with custom length. These workouts can be **synced across devices** and made public to be **shared** with others.







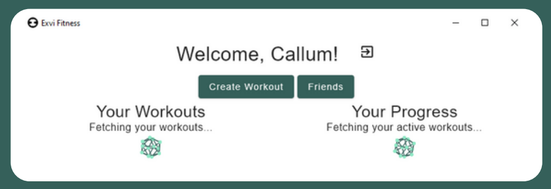
Design

Exvi Fitness is grouped into **three subprojects** to allow code to be shared between the client and server without each muddying the other with its own code. Hence, the subprojects are the **core**, the **client**, and the **server**. This design allows for encapsulation of common models without having to repeat them for the client and server. As well, this tri-project structure means client and server code can be completely separated and that each codebase can be tested and deployed independently. The separation of client and server codebases also means more security for client data, as the client can never know how the server internally interacts with data. The client and core are written in the **Kotlin** programming language, whereas the server is written in **Java**, which uses the JVM build of the Kotlin core as a dependency. This is to allow the client to be built on **multiple platforms**, which is not needed on the server as the platform is static. See page 2 for each module's design.

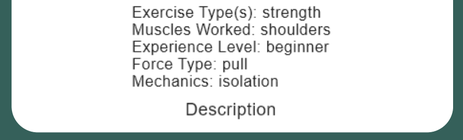
End Result

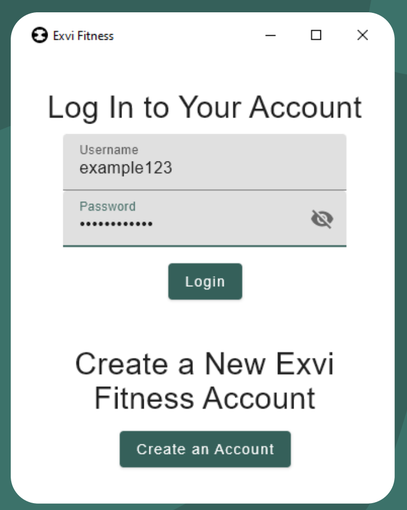
Exvi Fitness works well on **Windows**, **OSX**, **Linux**, and **Android**, however, it runs on the **JVM** on each of these platforms. Java 11 is required to run the application on desktop, which hinders the ease of use for those who do not have it installed. The core features are fleshed out fairly well, namely the **account creation**, **log ins**, **workout management**, and **friend system**. However, the **workout generation** leaves more to be desired as there was not time to implement a GUI for custom generators. Yet the groundwork in the code is present to improve the generation. Furthermore, tracking **past active workouts** is not what I had imagined in my design, and I had to completely omit the **nutrition recommendations** because of inadequate data. Overall, I am satisfied with my result as it does allow individuals to create, track, and share workouts across their devices which was the core of my initial concept.

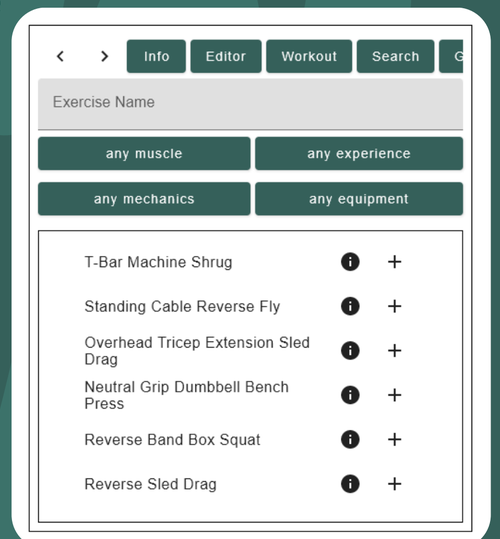
Features



- **Account system**
  - Users can create and log in to accounts
  - Verification with a mobile phone number
- **Data synchronization**
  - Users can access their account and all its data on any supported device
- **Workout creation**
  - Create new workouts, or edit your current ones
  - Add exercises to your workout
  - Modify the target reps/time of exercises in your workout
  - Generate workouts from preset generators
- **Exercise collection**
  - Exvi Fitness contains a large collection of exercises with several pieces of data including
    - Titles
    - Descriptions
    - Muscles worked
    - Equipment required
    - Video demonstrations
    - .. and more
  - These exercises can be queried from the workout creation page
- **Friend system**
  - Friend other users, or accept an incoming request to see their public workouts
  - Modify the visibility of your workouts from the workout creation page
- **Workout progression**
  - Progress through your workouts on your device
  - Input rep amounts or accept the target value
  - Save to your account to return to later







Possible Improvements

There are several improvements and additional features that could be added:

- Saving workouts to file (DOCX, PDF, etc) to print and use
- Better tracking of workout history with a calendar
- Nutrition recommendations post-workout
- Custom workout generators
- Workout generators which take into account previous user workout data
- Track body statistics, such as height, weight, body fat percentage, etc
- General health recommendations page
- Add custom exercises to a workout

Exvi Core Design

The core is the **shared model** for the client and the server; it is responsible for representing data such as exercises, workouts, and muscles in a congruent fashion to allow the client and server to communicate effectively.

There are a few main tasks that the core handles:

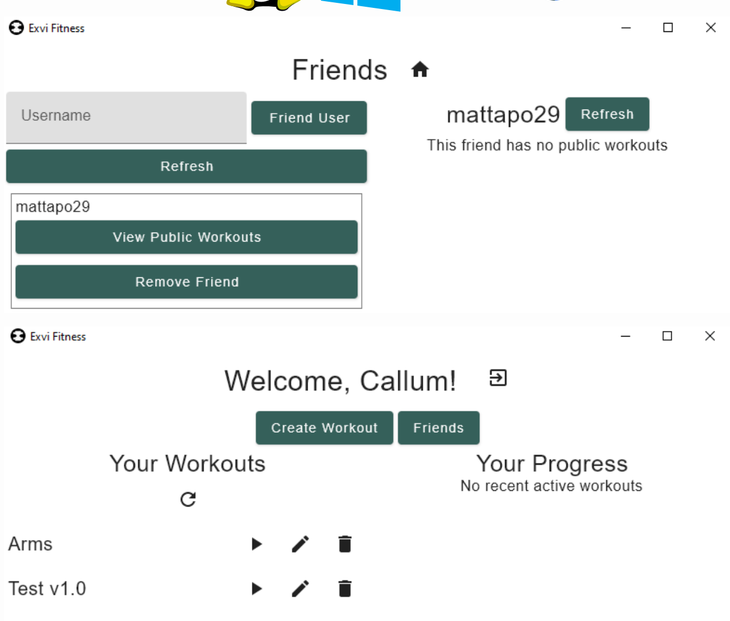
- **Data models**
  - Naturally, the core describes several types for data, including workouts, exercises, muscles, and unit values
- **Serialization**
  - The core manages polymorphic serialization of common types, meaning types can be serialized / deserialized in reference to their parent types
- **Encryption**
  - The client and server must know how data is encrypted, which is why the client provides encryption methods for them to communicate with
- **API requests / responses**
  - The job of the core is to be the intermediary between the client and server, so it describes an interface for communication with various request and response types
- **Core design pros**
  - Polymorphic serialization allows the server to very easily choose a response action based on the request type, allowing for the use of only one server endpoint and AWS lambda function
  - Interfaces for data types allows differing concrete implementations to be used
- **Core design cons**
  - It is more difficult to add new request types as they must be registered to the serializer
  - When core changes are to be used, it must be rebuilt online and imported in the client and server which can take a couple minutes

Exvi Client Design

The client is composed of the **user interface** and **client model**; the user interface interacts with the client model and core model, which in turn interact with the server to display proper information.

The client manages a few things:

- **Client data models**
  - Local data models such as client accounts and workout generation classes are managed
- **User interface**
  - The entire purpose of the client is to receive, transform, and display server information
  - The client uses Jetpack Compose, a declarative UI framework for creating multiplatform applications written in Kotlin
- **Platform differences**
  - There are many differences between platforms that must be accommodated for in code, primarily in the user interface



Exvi Server Design

The server manages AWS (Amazon Web Services) resources, primarily the document **database** and **SMS** service/ As well, the server responds to requests from the client.

The server has a few main jobs:

- **Server data models**
  - The server must manage the data structure of the document database
  - There are a couple primary root object types which are stored in the database: user data and user login info
- **Request actions**
  - Based on the request type, the server will take different actions
  - Said actions include updating workouts, deleting workouts, validating the app version, etc
- **Database actions**
  - Uses DynamoDB
  - The server interacts with the database API to update, sort, and retrieve user data
- **Managing other AWS services**
  - The server manages AWS SNS (Simple Notification Service), DynamoDB, Cloudwatch logs, and SES (Simple Email Service)

The server uses a 'serverless' design; it is accessed through an AWS API Gateway which maps to an AWS Lambda function which handles responses. This means that servers do not need to be maintained to run instances of the server application, but there is a startup time associated with the AWS Lambda function for a server to be allocated. Each AWS resource has proper permissions so it can only access the resources it requires such as SES, etc.