Architecture and Design Document

FitGoose

Version 1.0

SE2: Software Design and Architecture
CS446/CS646/ECE452 Spring 2015

Prepared by:

YuFan Dong, yf2dong, 20466594
XiangTao Bao, xtbao, 20409248
Jiayi Wang, j469wang, 20501675
Yifan Yang, y369yang, 20543540

July 5, 2015

Course Instructor: Ed Barsalou
Project TA: Oleksii Kononenko

Table of Contents

FitGoose is an easy use gym exercises tracking application, which will allow user track and analysis all the data about their exercises.

1. Architecture Description

A variety of Architectural Patterns were used to make this app both functional and easy to use.

Architectural Patterns

Blackboard:

The main architecture of our application is the Blackboard Architecture, as all the modules are dependent on the data and not so much with each other. We made a database helper class as the blackboard, and this class is the only class that can access the SQLite database stored on the device(it's also the class handle backup/ sync database to cloud). And for other modules, they all have different ways to deal with the data, so data processing is done in the individual modules. We choose this architecture because data matters a lot in our app, as the main purpose of our app is tracking the fitness data and show them pretty and neatly in either list or graphic.

Client-Server:

Our app have two servers. First, we have a small private server that is used to make small updates in our program. It contains all the default exercises and the links to their YouTube videos. The clients would periodically sync with our update server. To do that, our app will send a request to this private server and pull the online database to their device. This way, small changes to the app (e.g. changing the url of an exercise's YouTube video) would not require an update through the Google Play Store.

The other server is the Google Drive server which we use to backup user data. Users will backup their local database to their own Google Drive account. This two-servers architecture design is good for security as user's own data only stored at their own device and/or their own Google Drive, no one else(including our developers) can access it. Besides, using Google Drive as our backup server also improves our app's scalability, as most of the workloads is on the Google Drive server instead of our own server.

Event based:
Event based architecture is an obvious inclusion in our app. All user input and application response follows the event based architecture. User inputs will also update the database in real time.

Non-Functional Properties

Security:
For the NonFunctional Property of security, we have two servers to ensure the user data safety. We only let user pull(update) system data like new sets of exercise from our own server. User will not to send data to our own server, instead, they will save their data locally. And the local data will sync with user's google drive to achieve a remote copy. Since we believe google drive have a higher security level than our own server, we believe doing such way will enhance user's data security.

Efficiency:
There are many ways in which our app improve efficiency. First, our app uses a local database to cache user data. We will store and retrieve all user data locally. In this way we avoid use the remote database, which means we save the time to communicate with remote server. (we also sync local data with google drive thus safety and security can be guaranteed) Secondly we have simple and compact interfaces that aims increase user-friendliness. Different modules also share user interfaces that has the same functionality. Our calendar and daily plan module will call a same function to bring up the same dialog. Such design allows better architecture and at the same time achieve efficiency.

Adaptability:
For the nonfunctional property of adaptability, we chose to use API 17 in order to satisfy the most Android platform without sacrificing too much functionality. From report last month, API 17 (4.2) covers more than 75% of android user worldwide. Also we use relative location setting to design our user interface, in this way we can ensure our views will scale nicely with different devices with varied resolutions.

Scalability:
For the nonfunctional property of scalability, our app have many ways to be adapted to meet new size / scope requirements. First, our app can support as many users as possible since all user data are stored locally and get backup with their own google drive. Our own server only used to provide new features. Thus no

matter how big our server is, our app can support unlimited number of users. Besides, we have a lot of new features coming available like facebook support and share training plan. which may let more people know this app and get more people involved.

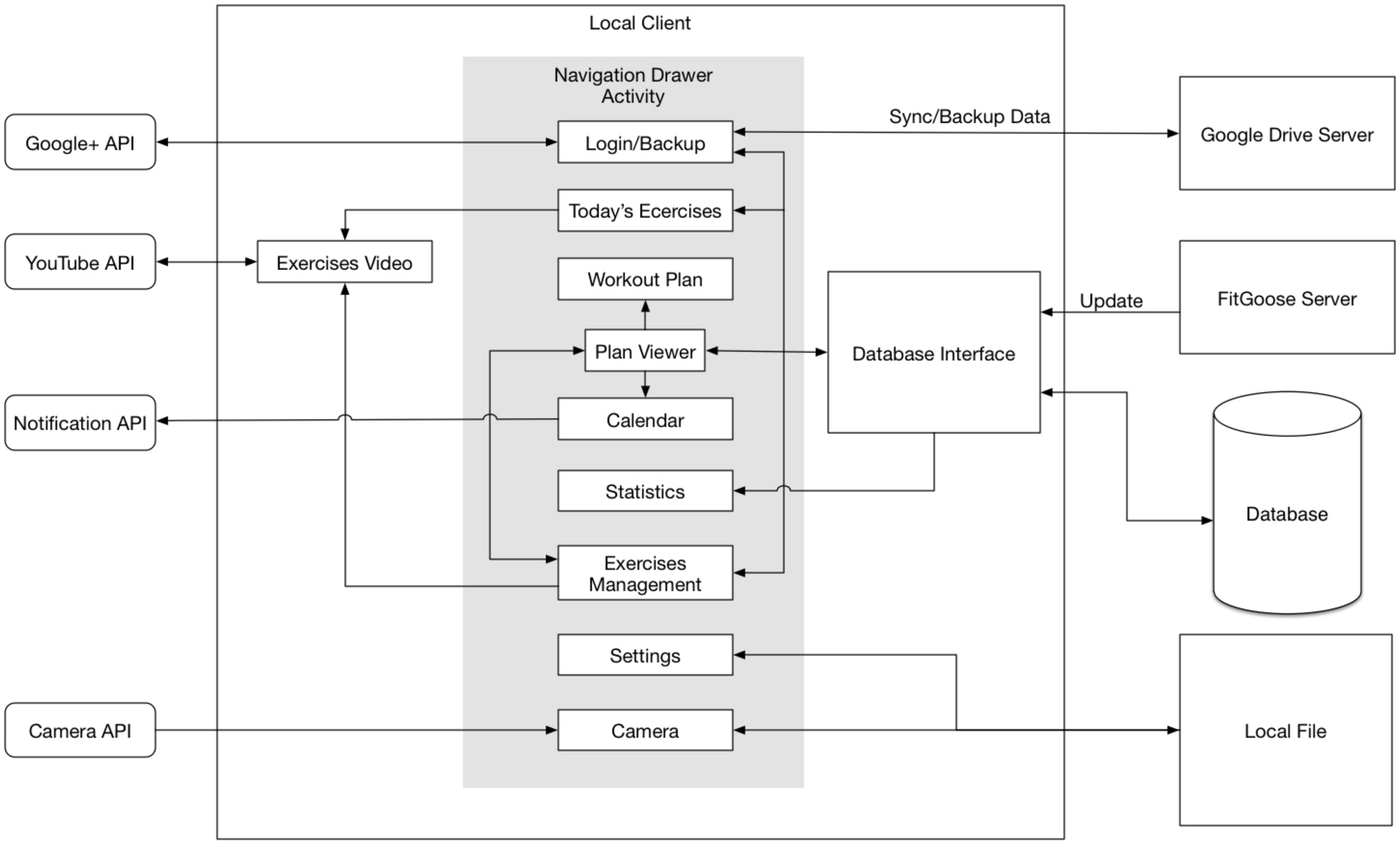
Dependability:

Our app is very dependable since we gracefully handle many different errors made by user or system compatibility issues. For all user input dialog, we have error checking and exception handling method to prevent invalid input and to ensure the app itself will not crush. Also in our design, we will try to restrict the user input so only the correct input is given. For instance, when we want the user to enter the number of sets for an exercise, we pop up a number input keyboard instead of a full size letter keyboard. Moreover, for camera or calendar and other APIs, if something goes wrong and they cannot be correctly opened, a dialog will be popped up so app itself will not terminate. Also for safety concern, we use google drive to backup user local data for higher security level.

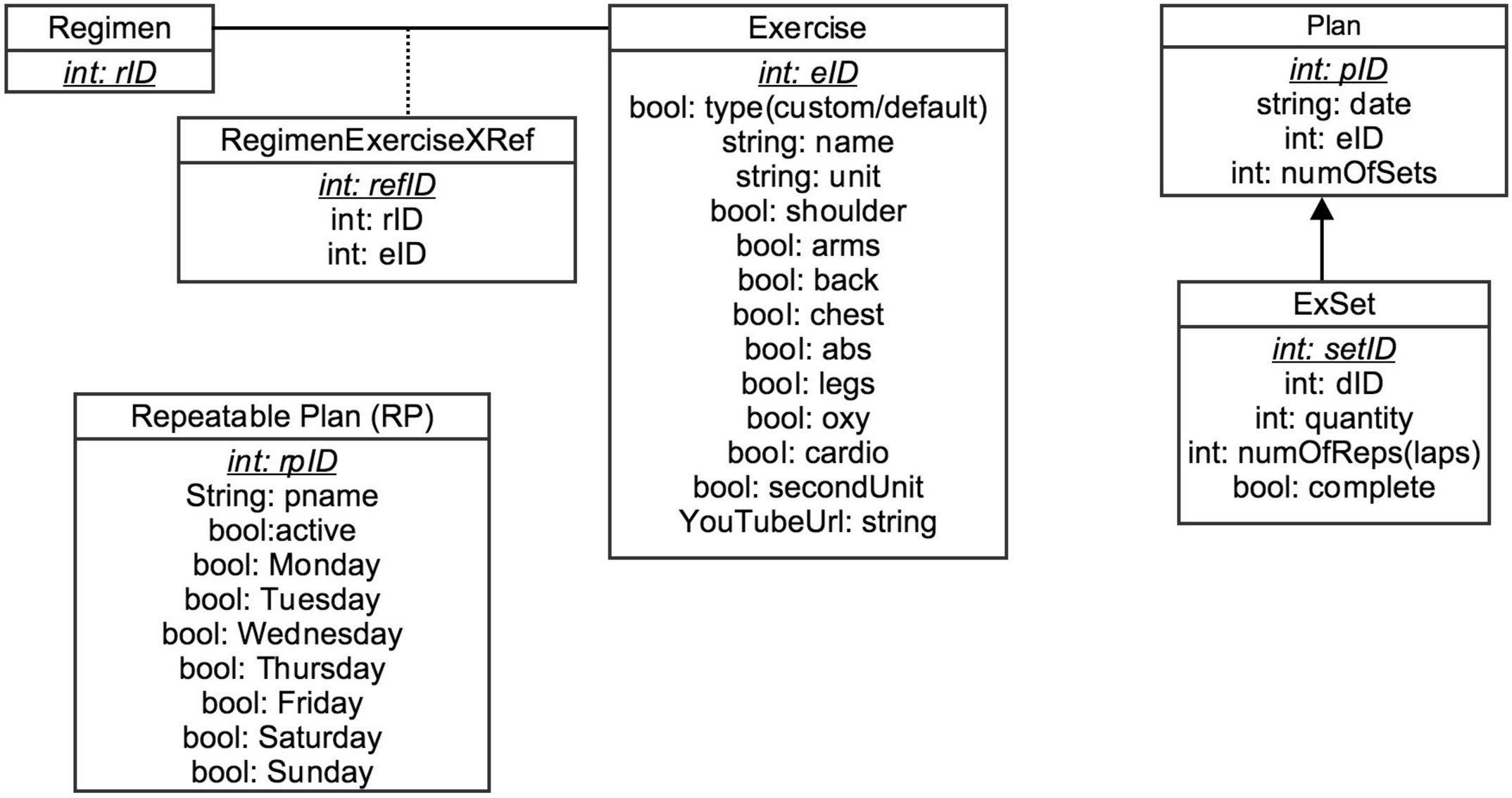
Evolvability:

For evolvability, it is easy for our app to add new features since we clearly define each module's responsibility. We have a local database and all the other modules have information communication with it. Thus if any further function or module is added, it only need to communicate with the local database, and other modules will not be touched. This way we reduce workload for testing and at the same time achieve high evolvability.

Architecture Diagram



Database Schemas Diagram



2. Design Description

The whole application was build on navigation drawer activity, by using navigation drawer, we can switch each feature by switch associate fragment. Each feature/module communicate directly with the database and do most of the data analysis on its own. The following is a brief description highlighting the design of each feature:

- *Today's Exercise - Display exercise process based on image colour, user can interact with data by dialog window.*
- *For MyPlan tab - using Continuous Scrolling design pattern to show exercises each day. User will be able to use "scroll to load" function to load more data if it is not shown.*
- *Calendar - User can use the calendar to check their workout history as well as add future workout plans.*
- *Statistics - Generate user statistics graph based on exercise data.*
- *Exercise Manager - Add/Delete exercise as require.*
- *Exercise Videos - Link to Youtube video and show each exercise's tutorial video.*

UI Design Pattern

Navigation Drawer:

Navigation Drawer is a panel with a menu hidden at left side of the screen. When an user click the button on left top or using finger slide from left edge of the screen, the list menu will open and user can click different options in the panel to switch to the feature they want to use. We are using Navigation Drawer Design Pattern to switch each feature with associate fragment.

The reason why we choose Navigation Drawer instead of tab menu or any other navigation design pattern is that most operation in our application is independent with others. User will not need to switch each views frequently. Hiding the panel will save a lot of screen spaces which is important for mobile apps which don't have large screens. The Navigation Drawer is also very intuitive and easy to use.

Continuous Scrolling:

Continuous Scrolling Design Pattern is used when user need to view a list of data but the data is too large to show in a single page, or the data has infinite set (like repeat exercises for each day). This design pattern will automatically load new set of data when user scrolled to the bottom of the page. In our program, we have

MyPlan feature, which will show each day's exercises. We would like to using Continuous Scrolling design pattern to let user to load exercises plan in the future even if it is far away from current date.

Compare to paging the data into different view, Continuous Scrolling is much more natural way to show all the data. Also for paging, user need to click to load more data, as our goal for this application is let user click as less button as possible, Continuous Scrolling design pattern will automatic load more data when scrolling to the bottom which will help us achieve this goal.

Backend Design Patterns

Facade:

We used the facade pattern to abstract database operations. The other modules in the program cannot access the database directly and they cannot send SQL commands to the database. Instead they must interact with a database manager module. Common queries are abstracted as methods and the parameters can be passed to specify the exact query. The other modules can also add/delete entries to the database via methods supplied by the database manager with some restrictions.

By providing an extra layer between the database and the rest of the program, we add a layer of security and encapsulation. We prevent things like SQL injections and avoid the program being crashed by an invalid SQL command. It also abstracts the complexity of SQL commands into simple methods.

Proxy:

We use the proxy pattern for passing information from the database to the rest of the program. Each entry from a table is represented by a class. For example, when an exercise is retrieved from the Exercise table, its information is copied into into a newly created Exercise proxy object. That object is then passed to the module that requested it. Hierarchical relationships between tables are represented through composition between the objects. For example, a workout plan contains many exercises, so the plan's exercises are turned into an arraylist and put into the workout plan class as an instance field.

By having the modules deal with Java classes which are proxies of database entries, it becomes easier to access the information. We also represent the data hierarchical structure of our data through the proxy classes to further ease the accessibility of the data.

Singleton:

We use the singleton pattern for the database management module. The class FGDataSource, which is used for creating and managing the database, is created using the singleton pattern. Meaning only one instance of FGDataSource can exist in the program, and only one instance of the database can exist.

By ensuring only one instance of the database can exist in the program, we guarantee there would be no duplicate databases. We also ensure every module is accessing the same database/database manager to to ensure consistency and reliability.

Coupling Discussion

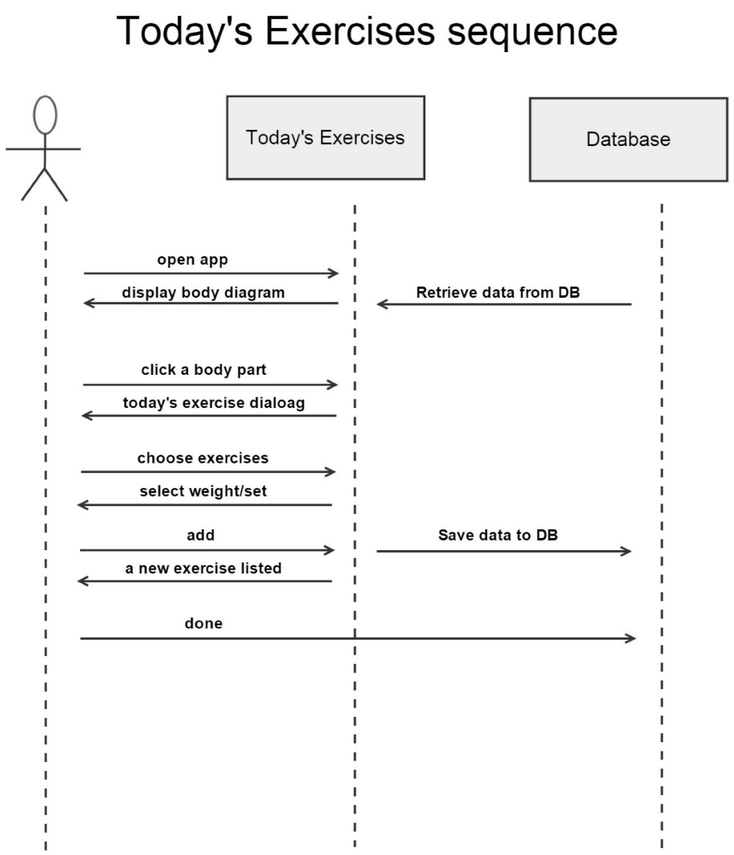
In order to reduce coupling in our design, we divided the program into several distinct modules, each corresponding to a specific feature. All the data is managed by the database management module, and by using a facade pattern we are able to reduce the coupling between the database and the other modules. All the UI-oriented modules like Today's Exercise and Calendar interacts with the database to obtain the data, but hardly communicate to each other in order to reduce the amount of coupling. When the modules communicate with each other, it is always through well defined methods. Information is passed using mostly proxy objects(Exercise, Plan etc.) in order to reduce coupling.

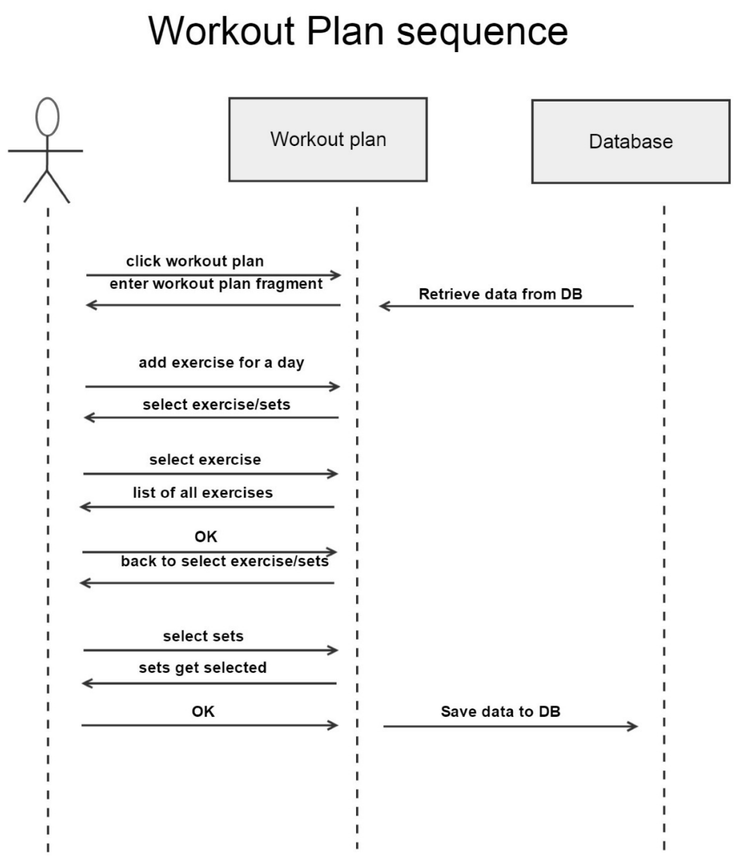
Future Expansion

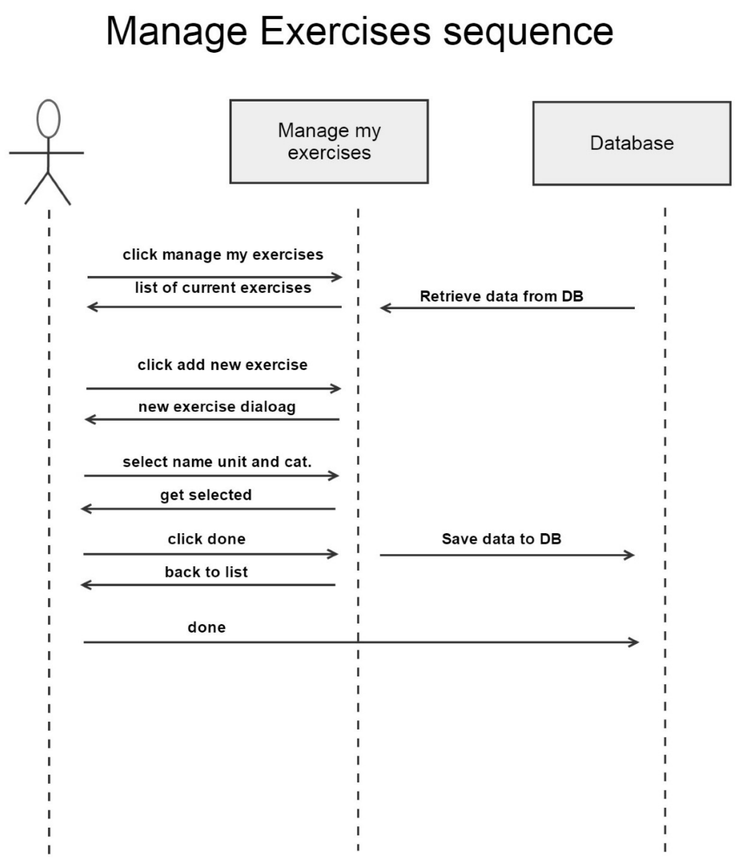
One of the possible additional feature we discussed since day one was adding a server. We did eventually add a server and google driver to our design, but we can expand that further. We can add a login system and have user data be stored on our own server instead of relying on google drive backup. The login UI can easily be inserted as a Fragment by using the Navigation Drawer pattern. Since the rest of the modules mainly only communicates with the database manager, they wouldn't be affected. Only the database manager need to be updated to handle syncing issues.

Another additional feature we thought about adding was connecting the app with social media apps such as Facebook and Twitter. The user can post their regimens on social media or post the exercises they worked on on their recent trip to the gym. They can also post their progress pics. Adding this feature would not affect other modules in the program since the social medias module would pull information directly from the database and the local files.

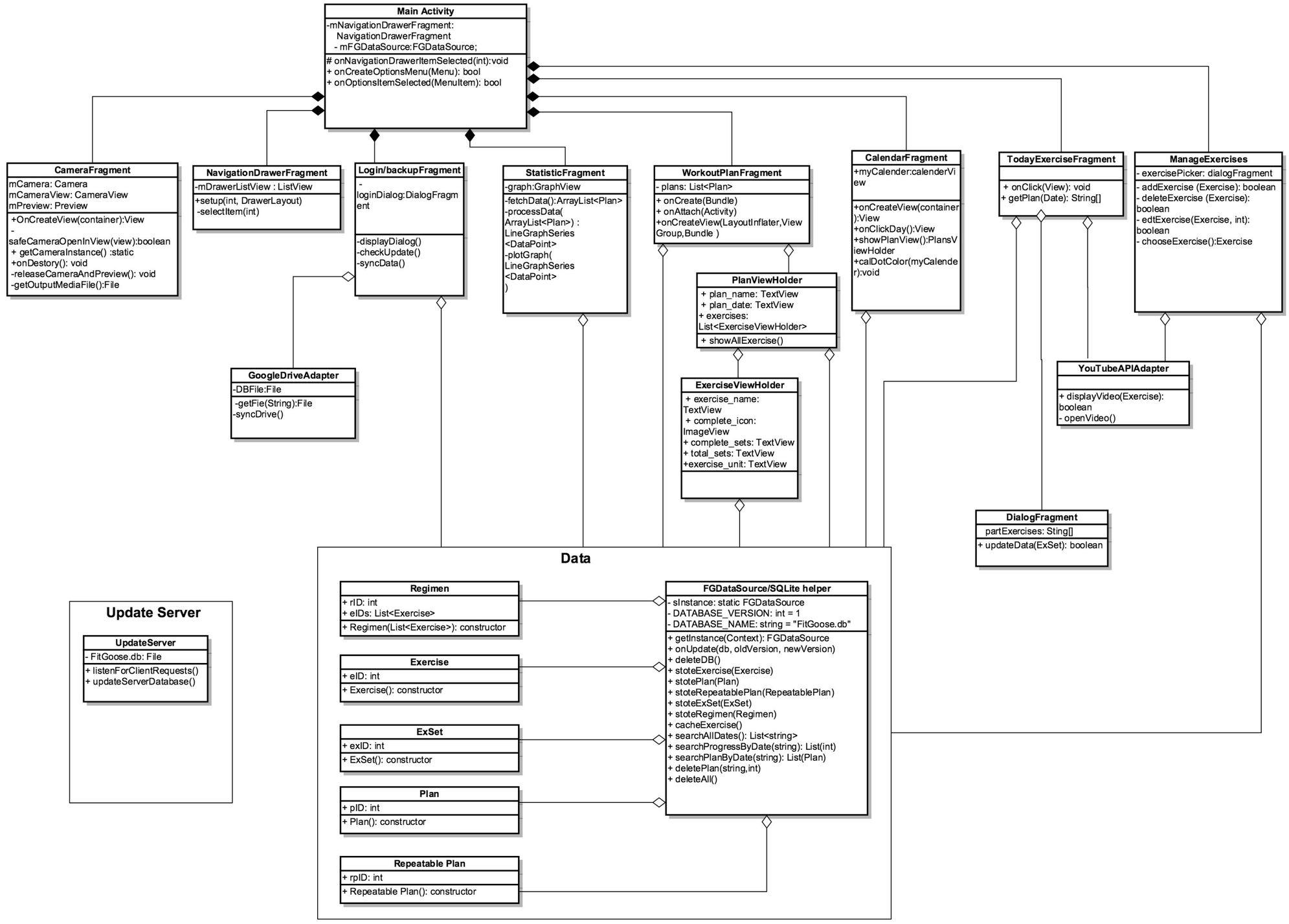
Sequence Diagram







Class Diagram



3. Contribution to Architectural Components

XiangTao Bao

- General Idea and Design
- Today's Exercise
  - Body Figure
- Exercise Videos (YouTube)

YuFan Dong:

- Navigation Drawer Menu
- Statistics
- Google Plus/Google Drive

Jiayi Wang

- Database
- Workout Plan
- Plan Viewer
- Update Server

Yifan Yang

- Event Calendar
- Notification from Calendar
- Camera API
  - Photo Gallery