Bilkent University
Department of Computer Engineering

Senior Design Project

Gymtor

Low-Level Design Report

Team Members : Emre Tolga Ayan, Umur Göğebakan, Cemal Arda Kızılkaya, Ömer Faruk Kürklü, Akın Parkan

Supervisor: Selim Aksoy

Innovation Expert: Veysi İşler

Feb 8, 2021

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

1. Introduction

Exercising regularly is one of the key ingredients of a healthy lifestyle. However, due to various reasons such as not having enough time and motivation due to heavy work loads or not having enough money to spend money on fitness clubs, most people do not exercise at all. These reasons suggest that the accessibility aspect of exercising poses a significant problem, which led to the main idea of our project.

In order to make exercising accessible to more people, our team came up with the idea of a sport assistant, which is powered by cutting edge deep learning techniques to analyze the body pose of a person and provide relevant real-time feedback to the user about what can be improved about a certain exercise in order to help the users in making the most out of their workouts.

In this report, we will be presenting our low level design report for Gymtor. It majorly covers the detailed explanation of packages we use, and the classes we implement.

1.1. Object design trade-offs

1.1.1. Speed vs Complexity: Gymtor is an application powered by artificial intelligence and machine learning. While training our data, we can create much complex architectures for better performance. However, as the model sizes increase, the time necessary to process the real time data and generate an output increases. Thus, as we increase the complexity, the feedback speed of the application reduces.

1.1.2. Performance vs Speed: For better performance, we may use cloud computing services like Amazon AWS or Google Cloud. However, sending real time visual data to the server and waiting for an answer causes high delay. Thus, our options are to use cloud computing with immense computing power but a delayed service, or local computing with limited computational power but with a low delay.

1.1.3. Functionality vs Usability: We may add many features such as focusing on a body area by touching to the screen, or adding the option to customize the feedback we generate, which provides more functionality to the user. However, as we increase the features, the application becomes more difficult to use which reduces usability.

1.2. Interface documentation guidelines

In this report, we will follow the following style for demonstrating the classes:

class ClassName
Class definition
Attributes
-attribute1: attribute_type -attribute2: attribute_type
Methods
+method_name(method_param: param_type) : return_type +method_name(method_param: param_type) : return_type

"-" means that attribute or method is private, and "+" means public.

1.3. Engineering standards (e.g., UML and IEEE)

We have used UML standards to create our figures. We have used IEEE referencing standards for our references [1] and IEEE 730-2014 quality standards [2] for our production.

1.4. Definitions, acronyms, and abbreviations

Abbreviations and some technical terms that have been used in the report are listed here with their brief explicit definition for a better understanding of the latter sections:

1.4.1. Cloud Storage

HTTP based request responder to retrieve information. Online and can be reached through the network.

1.4.2. Client

HTTP based request sender. Sends online requests to the backend to retrieve information as a response.

1.4.3. Movement Ground Truth

List<Float> of 3D point cloud sparse matrices to represent the accurate movement pattern of an exercise.

1.4.4. Machine Learning Model

Trained models to produce certain outputs from a given output according to the learnt/estimated parameters during the training phase.

1.4.5. JSON

JavaScript Object Notation as a lightweight communication format to store and transfer data.

1.4.6. AWS

Amazon Web Services, a cloud platform to provide computational resources as well as remote machine instances.

2. Packages

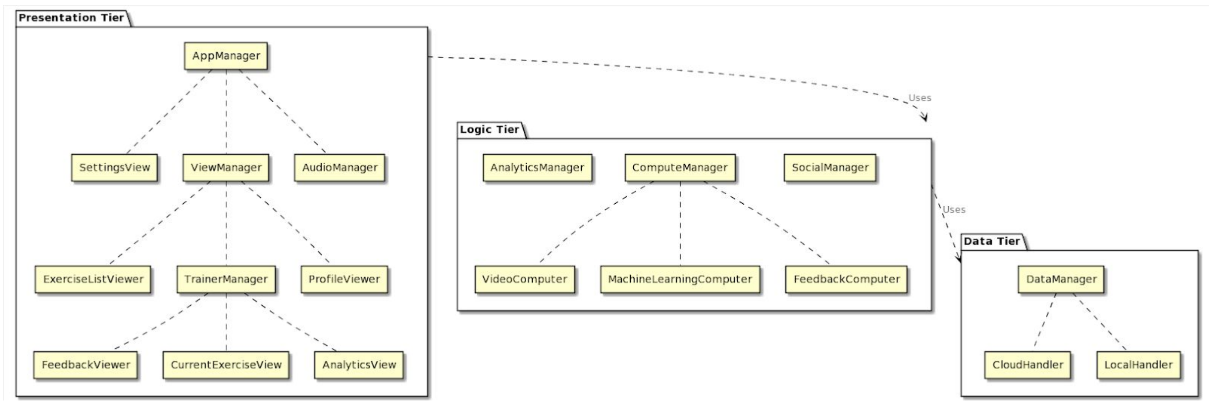

Figure 2: Packages Overview

In our Application, we have designed a three-layer architecture for modularity and scalability. Since our application makes real-time visual data analysis and real-time feedback generation, we have tried to keep our model as simple as possible to reduce the complexity of the modules and achieve a better and a faster performance.

We have presentation tier, logic tier and data tier as three layers. Presentation tier is responsible for displaying all the visuals and management of them. Logic tier is responsible for computational operations including machine learning usage. Data tier is responsible for the communication of the application with the cloud data or local data.

2.1. Presentation Tier

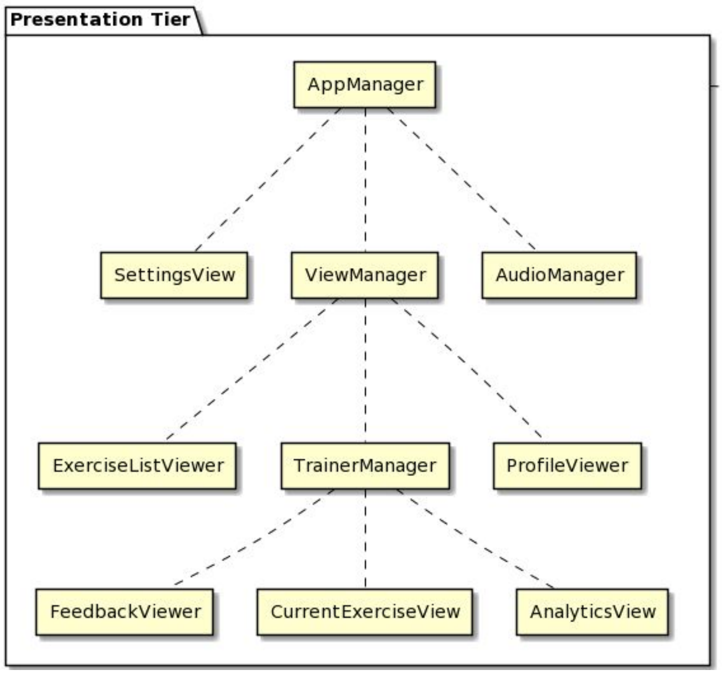

Figure 3: Presentation tier Overview

This tier is responsible for the management of visual and auditory material that is presented to the user. It includes the main package which is AppManager that handles the interaction between other packages. ViewManager is responsible for displaying the front-end content of the application. CurrentExerciseView package is responsible for displaying the video screen to the user with the generated feedback. It also captures the inputs of the user in this stage, and they are sent to the logic tier. ExerciseListViewer is responsible for retrieving the exercise plans of the user, displaying them, or scheduling them. ProfileView is responsible for retrieving and displaying the user-specific data. Other non-specific visual content is handled in the ViewManager package. AudioManager package is responsible for playing audio assets. These audio assets include sound effects such as sliding a menu, clicking on a button, etc. They also include auditory instructions in an exercise session. The SettingsView package manages the user's UI related application settings. This might be the theme of the application, colors, some UI specific personalizations, etc.

2.2. Logic Tier

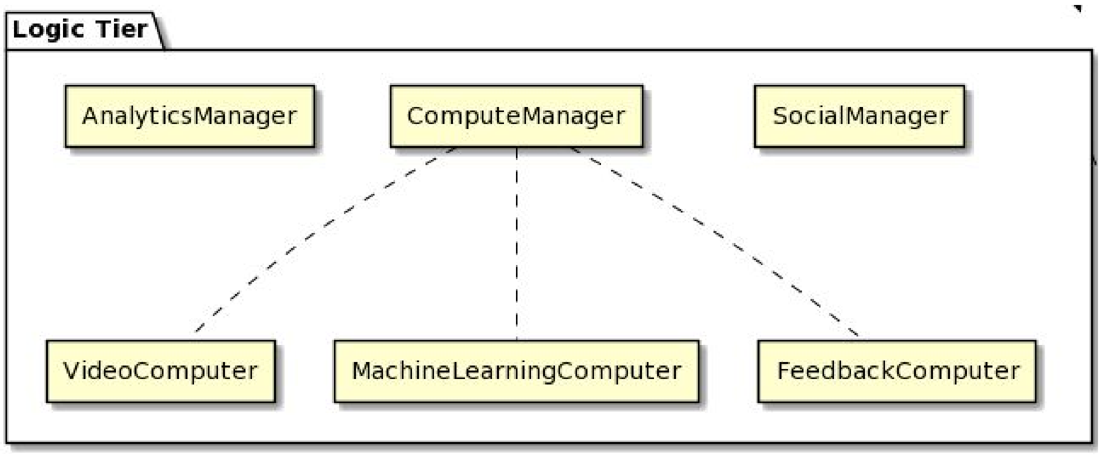

Figure 4: Logic Tier Overview

Logic tier is responsible for management of all logical operations behind the scenes. It includes a main ComputeManager which handles all the computations and also is in communication with the presentation tier. SocialManager is in control of the social activities of Gymtor such as adding personalized exercise lists, looking at friends activities etc. AnalyticManager computes the personalised feedbacks, calorie usage etc. ComputerManager handles the machine learning application of ours.

2.3. Data Tier

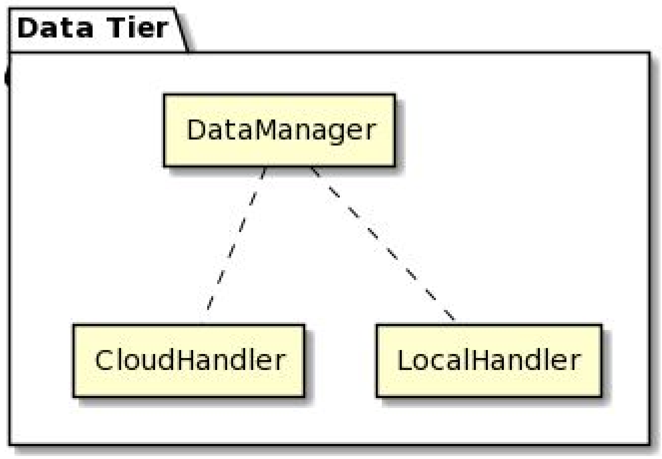

Figure 5: Data Tier Overview

Data tier handles our cloud handler and runs on a server unlike the rest of our tiers. It contains a main DataHandler which is in contact with our logic tier and handles the database appropriately. CloudHandler handles the cloud specific packages and LocalHandler handles any logical work done in the server machine.

3. Class Interfaces

3.1. Presentation Tier

3.1.1. ProfileViewer

class ProfileViewer
This class is responsible for handling the changes to the user's account.
Attributes
-dbc: DBConnector -userID: String
Methods
+changePassword(oldPassword: String, newPassword: String): boolean +updateProfilePhoto(image: Image): void +setName(name: String): void +setHeight(height: int): void +setWeight(weight: int): void +deleteAccount(password: String): void

- changePassword(oldPassword: String, newPassword: String): Updates the password of the user.
- updateProfilePhoto(image: Image): Updates the profile photo of the user.
- setName(name: String): Updates the name of the user.
- setHeight(height: int): Updates the height of the user.
- setWeight(weight: int): Updates the weight of the user.
- deleteAccount(password: String): Deletes the account of the user.

3.1.2. ViewManager

class ViewManager
This class is responsible for displaying the requested view.
Attributes
None
Methods
+setCurrentView(viewName: String): void

- setCurrentView(viewName: String): Displays the requested view on the screen.

3.1.3. ExerciseListManager

class ExerciseListManager
This class is responsible for displaying the list of exercises belonging to a user.
Attributes
-dbc: DBConnector -userID: String
Methods
+getExercises(userID: String) : List<Exercise> +addExerciseToList(ex: Exercise, userID: String): boolean +removeExerciseFromList(ex: Exercise, userID: String): boolean +deleteExerciseList(userID: String): boolean

- getExercises(userID: String): Gets the exercise list of the user.
- addExerciseToList(ex: Exercise, userID: String): Adds the exercise to the user's exercise list.
- removeExerciseFromList(ex: Exercise, userID: String): Removes the exercise from the user's exercise list.
- deleteExerciseList(userID: StringD): Deletes the exercise list of the user.

3.1.4. TrainerManager

class TrainerManager
This class is responsible for displaying the feedback to the user's current movement.
Attributes
-dbc: DBConnector -userID: String -exerciseInfo: List<Exercise>
Methods
+detectMovement(frame: Image, movement: Movement): Feedback

- detectMovement(frame: Image, movement: Movement): Examines the movement of the user with the help of the trained model and returns a feedback containing the status of the examination and the constructed body model with colors.

3.1.5. StatisticsViewer

class StatisticsViewer
This class is responsible for displaying the statistics at the end of an exercise session.
Attributes
-dbc: DBConnector -userID: String
Methods
+getFeedbacks(userId: String): List<String>

- getFeedbacks(userId: String): This method returns the statistics related to the user's exercises.

3.1.6. CurrentExerciseView

class CurrentExerciseView
This class is responsible for displaying the user's current exercise.
Attributes
-dbc: DBConnector -exerciseID: String
Methods
+getExerciseView(exerciseID: String): View

- getExerciseView(exerciseID: String): This method is used to fetch the exercise information from the database using the exerciseID and to display it on the screen.

3.1.7. AnalyticsView

class AnalyticsView
This class is responsible for displaying the user’s past exercises along with useful statistics.
Attributes
-dbc: DBConnector, -userID: String -history: List<Exercise>
Methods
+getAnalyticts(userID: String): List<Exercise> +deleteAnalytics(userID: String, analyticsID: String): boolean

- getAnalyticts(userID: String): This method is used to fetch the users analytics from the database and return these analytics as a list of Exercise objects.
- deleteAnalytics(userID: String, analyticsID: String) : This method is used to delete specific analytics of the user from the database. It requires userID and analyticsID. If the requested behaviour is successfully done, then this method returns true. Otherwise, it returns false.

3.1.8. LogInView

class LogInView
This class is responsible for providing an interface for viewing the login page of the application.
Attributes
-dbc: DBConnector
Methods
+handleLogin(email: String, password: String): boolean +handleGoogleLogin(email: String): boolean

- handleLogin(email: String, password: String): This method is provided to users to login to the application. It simply checks if the email exists on the database and if it exists this method checks if the password is valid or not. If the login request is valid, then this method leads the user to their profile page.
- handleGoogleLogin(email: String): This method is provided to users to login to the application. It simply checks if the email exists on the database. If the login request is valid, then this method leads the user to their profile page.

3.1.9. SignUpView

class SignUpView
This class is responsible for providing an interface for viewing the signup page of the application.
Attributes
-dbc: DBConnector
Methods
+handleSignUp(name: String, email: String, password: String, repeatPassword: String): boolean +handleGoogleSignUp(email: String): boolean

- handleSignUp(name: String, email: String, password: String, repeatPassword: String): This method is provided to users to sign up using their own email and the password that they specify. This method simply checks if the password and repeatPassword are the same and if the given email already exists or not.

- handleGoogleSignUp(email: String): This method is provided to users to sign up using their gmail. This method simply checks if the given email already exists or not.

3.1.10. TodayView

class TodayView
This class is responsible for providing an interface for viewing today's exercise information.
Attributes
-dbc: DBConnector, -userID: String, -view:View
Methods
+getTodayView(userID: String, today: TimeStamp): View

- getTodayView(userID: String, today: TimeStamp): This method is used to fetch user's exercise information from the database and put it into the view object.

3.1.11. HistoryView

class HistoryView
This class is responsible for providing an interface for viewing the previous exercise information.
Attributes
-dbc: DBConnector,
-userID: String,
-history: List<View>
Methods
+getHistoryView(userID: String): List<View>
+deleteHistory(userID: String, today: TimeStamp): boolean

- getHistoryView(userID: String): This method is used to fetch user's exercise information from the database and put them into the history list.
- deleteHistory(userID: String, today: TimeStamp): This method is used to delete the history of a specific timestamp from the users previous exercise history.

3.1.12. ScheduleExerciseView

class ScheduleExerciseView
This class is responsible for providing an interface for scheduling an exercise
Attributes
-dbc: DBConnector
-CurrentUserID:String
Methods
+scheduleExercise(userID: String, exerciseID: String, day: TimeStamp): void

- scheduleExercise(userID: String, exerciseID: String, day: TimeStamp): This method is used to assign a timestamp to users exercise.

3.1.13. ExerciseView

class ExerciseView
This class is responsible for providing an interface for modifying and visualising the exercises of the user.
Attributes
-dbc: DBConnector -CurrentUserID:String -exerciseID: String
Methods
+getExerciseView(userID: String): void +deleteExercise(userID: String, exerciseID: String): boolean +updateExercise(userID: String, exerciseID: String): boolean

- getExerciseView(userID: String): This method is used to fetch and illustrate the users' exercises.
- deleteExercise(userID: String, exerciseID: String): This method is used to delete the users exercise. The userID and the exerciseID are provided.
- updateExercise(userID: String, exerciseID: String): This method is used to update the users exercise. The userID and the exerciseID are provided.

3.1.14. AudioManager

class ExerciseView
This class is responsible for providing audio feedback to the movement of the user.
Attributes
-CurrentUserID:String -exerciseID: String -AudioComputer: AudioNode -mlComputer: MachineLearningComputer
Methods
+generateOutput(feedback: Feedback): void

- generateOutput(feedback: Feedback): This method is used to generate audio output using the AudioNode instance using the feedback output generated by the MachineLearningComputer instance.

3.2. Logic Tier

3.2.1. ComputeManager

class ComputeManager
This class is responsible for the communication and combination of the computations done.
Attributes
-videoComputer: VideoComputer -machineLearningComputer: MachineLearningComputer -feedbackComputer: FeedbackComputer -appManager: AppManager
Methods
+retrieveVideo(): List<Float> +predictFromVideo(): List<Float> +retrieveFeedbacks(): List<Float> +retrieveStats(): List<Float>

- retrieveVideo(): Retrieve video from video computer.
- predictFromVideo(): Get predictions from machineLearningComputer.
- retrieveFeedbacks(): Get feedback data from feedbackComputer.
- retrieveStats(): Retrieve statistics of the current and previous exercises from feedbackComputer.

3.2.2. MachineLearningComputer

class MachineLearningComputer
This class is responsible for processing the user inputs and extracting useful information from it.
Attributes
-bodyPoints: List<Float> -criticalAngleValues: List<Float>
Methods
+poseEstimation(videoFrames: List<Float>): List<Float> +getBodyPoints(): List<Float> -calculateBodyPoints(): List<Float>

- poseEstimation(videoFrames: List<Float>): Using the video data, predict the pose of the user, which will be used later for predicting the accuracy.
- calculateBodyPoints(): Find the critical points of the body that will be used in pose estimation.

3.2.3. VideoComputer

class VideoComputer
This class is responsible for generating visual guides and animations based on the user video input and the feedback generated.
Attributes
-animator: Animator -frame_per_sec: Integer
Methods
+generateAnimation(feedback: List<Float>): List<Float> -setAnimator(animator: Animator): void

- generateAnimation(feedback: List<Float>): Based on the feedback, generate an animation that will demonstrate the correct way of doing the exercise.

3.2.4. FeedbackComputer

class FeedbackComputer
This class is responsible for generating feedback based on the predictions and ground truths.
Attributes
-mov_ids_and_angle_thresholds: List<Float>
Methods
+isMovementCorrect(predictions: List<Float>): boolean +generateFeedback(predictions: List<Float>): List<Float>

- isMovementCorrect(predictions: List<Float>): Determine whether the exercise of the user is correct in the threshold limitations.
- generateFeedback(predictions: List<Float>): Generate feedback based on the exercise correctness predictions.

3.2.5. SocialManager

class SocialManager
This class is responsible for sharing personalized exercise lists, looking at friends activities etc.
Attributes
-dbConnect: DBConnector -friendsIDs: List<Float> -exercisePlans: List<Float>
Methods
+setDbConnector(dbConnector: DBConnector): void +setFriends(frendIDs: List<Float>): void +setExercises(exercises: List<Float>): void +uploadExercises(): void +retrieveFriendsData(): List<Float>

- uploadExercises(): Upload custom exercise plans to the social hub so that friends can retrieve it.
- retrieveFriendsData(): Retrieve friends, and their shared data.

3.2.6. AnalyticsManager

class AnalyticsManager
This class is responsible for generating and retrieving exercise statistics.
Attributes
-dbConnect: DBConnector -chartGenerator: ChartGenerator
Methods
+setDbConnector(dbConnector: DBConnector): void +updateCaloriesDB(caloriesData: List<Float>): void +updateTimeDB(timeData: List<Float>): void +retrieveProgressFromDB(): List<Float> +generateCharts(): List<Float>

- updateCaloriesDB(caloriesData: List<Float>): Calculate and update the calories burnt in the exercise, and save the data to the database.
- updateTimeDB(timeData: List<Float>): Save the time spent for the exercises to the database.
- retrieveProgressFromDB(): Retrieve previous statistics of the user from the database.
- generateCharts(): Generate charts based on the data in the database.

3.3. Data Tier

3.3.1. DataManager

class DataManager
This package is used for managing the operations on the database based on the user's interactions with the app.
Attributes
-dbConnect: DBConnector
Methods
+ setConnection(dbConnect: DBConnector) : void + getContact() : Contact + addContact(contact : Contact) : void + deleteContact (name : String) : void + setSettings(i : int, p : int) : void + addRelative(name : String, picPath : String) : void + setRelative(name : String, picPath : String) : void + deleteRelative(name : String) : void + reset() : void

- dbConnect: is the connection class which holds the confidentials required to make HTML connections.

3.3.2. CloudHandler

class CloudHandler
This package is used for handling the data in the database. In database user information like username password, number of calories burnt, amount of time has been spent on each exercise, friend information are stored.
Attributes
-users: HashMap<username, userKey> -exercises: List<Exercise> -exercisePlans: HashMap<planKey: String ,ExercisePlan> -userData: HashMap<userKey, User> -complaints: List<Complaint>
Methods
+ setConnection(dbConnect: DBConnector) : void + newExercisePlan(dbConnect: DBConnector, ep: ExercisePlan): String + newExercise(e: Exercise): String + newUser(dbConnect: DBConnector, user: User): String + newComplaint(dbConnect: DBConnector, cp: complaint): String + setSettings(i : int, p : int) : void + reset() : void

- Cloud Storage holds information for social networkability.
- All users data, exercises and exercise plans will be held here for other users to reach.
- Complaints will be held here for admin users to reach.

3.3.3. LocalHandler

class LocalHandler
This package is used for handling the information obtained from VideoComputer and MachineLearningComputer. Since it can be inefficient to handle them on the server side. (Cloud side)
Attributes
-exercises: List<Exercise>
-exercisePlans: HashMap<ExercisePlan>
-user: User
-activityHistory: ActivityHistory
Methods
+ setConnection(dbConnect: DBConnector) : void
+ setExercises(dbConnect: DBConnector): List<Exercise>
+ setExercisePlan(dbConnect: DBConnector ,planKey: String): ExercisePlan
+ newExercisePlan(ep: ExercisePlan): String
+ setSettings(i : int, p : int) : void
+ reset() : void

- All the required material for the app to function on users' phones will be held here.
- Static exercises will be requested from the cloud and will be stored in local phone storage. Exercise plans will also be stored here.
- All the user data will be held here.

4. References

[1] “IEEE REFERENCE GUIDE.” [Online]. Available: https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf.

[2] “IEEE 730-2014 - IEEE Standard for Software Quality Assurance Processes,” *IEEE SA - The IEEE Standards Association - Home*. [Online]. Available: https://standards.ieee.org/standard/730-2014.html. [Accessed: 08-Feb-2021].