3/27/2025

LUNKI SYSTEM REQUIREMENTS

Group Delta

Angelo Turano

Matthew Bautista

Carlos Escobar

Benjamin Sebiri

Table Of Contents

Introduction

The Lunki application allows users to make friends, set goals, chat with others, and become healthier in the process. This document contains the class diagram which shows the structure for our app Lunki. It contains details about the different objects and how they interact with each other using methods. It also has a use case diagram illustrating the individuals and the roles that go with them. The use cases encompass a broader look at how users interact with the Lunki system. Use Case Descriptions are included to go more into depth about how the system handles user input and examines exception scenarios that may occur for certain use cases. Finally we have system sequence charts that will with diagrams show the relationship between the user and the system through methods and the flow of information. Most if not all of the Use Cases correspond to an SSD (We narrowed our scope after some challenges to our group arose).

Description Model

For this application you need a smart device like a phone and an internet connection. With that you can fill out your information and create an account to access our database of workouts and shared information from other users. This app acts as a bridge of connection between experienced gym goers to those just starting to allow everyone to get a shared experience and grow together.

Creating an account is as simple as entering a username, password, and an email you would like to be associated to your account. After creating an account, you have the option to use the app solo, or you can add friends by inputting their Lunki username into the given field. Once inputted, a friend request will be sent to that User, and upon accepting the request you and the other user are added to each other's friend lists.

The main portion of the app involves starting and ending workouts. When a user is ready to begin a workout, they start a workout in the Lunki app. (For a better user experience, it is recommended to have an apple watch/FitBit/etc. connected to Lunki). Once finished, the user may then end the workout. If applicable, a request is sent to the user's fitness watch to retrieve the vitals information from the specified time frame. It is then stored for future access.

Users also have the ability to set goals for themselves. Given a description and a timeframe, Lunki creates a goal and then sends a notification to the user every 24 hours until the goal is finished. In our research, we noticed that many people (especially around New Year's) set goals/resolutions to go to the gym and get fit, however most lose focus and forget about their goals within a month. We hope that Lunch's push notification system will help motivate users to "keep grinding" and not give up on their goals. After completing a goal, it is stored so that the user can look back at their progress over time.

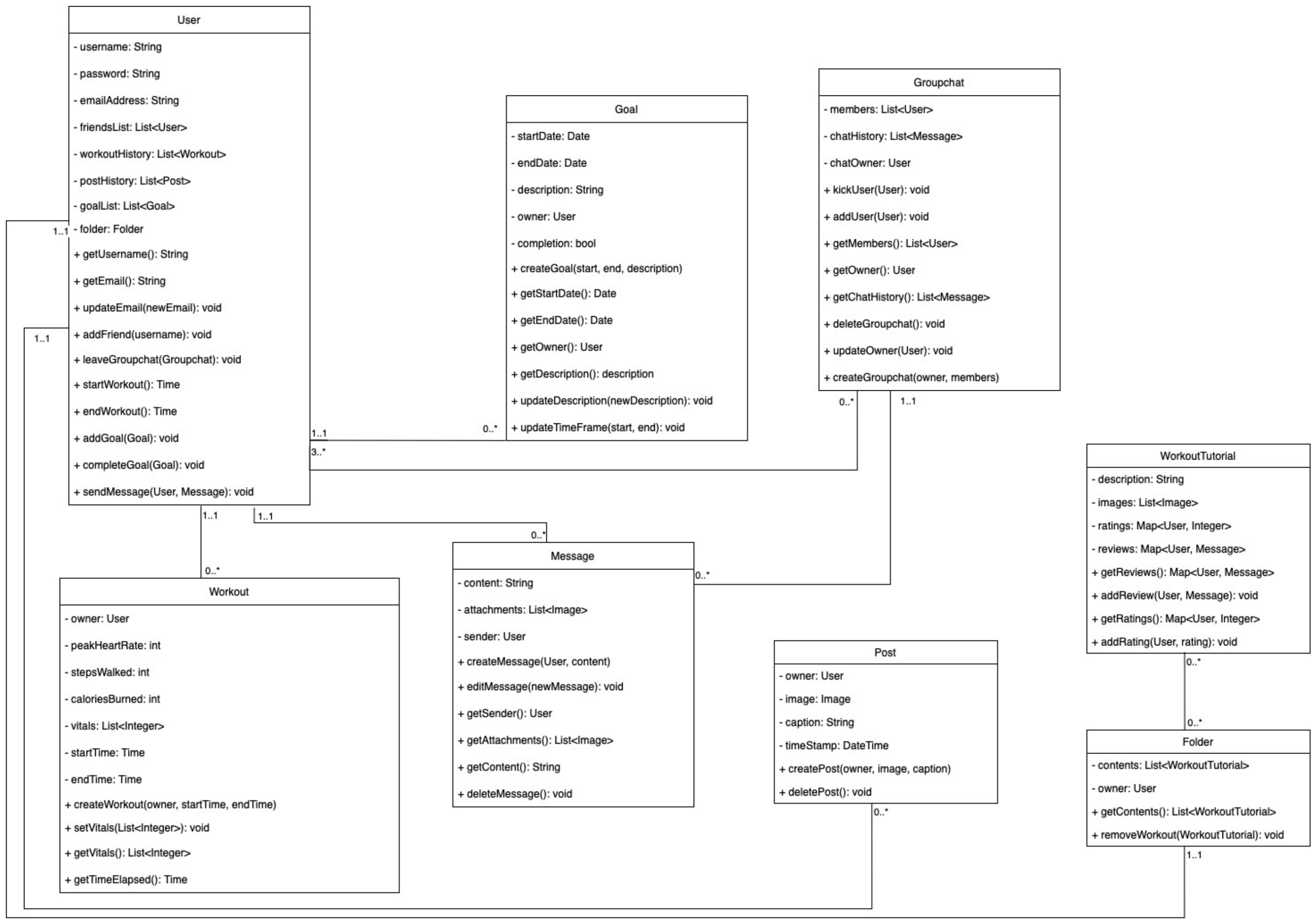
As it pertains to the social aspect of the app, as previously mentioned users can add friends just by inputting their username. Users also have the ability to create group chats that include their friends. Joining a friend's group chat is as easy as asking the friend to send you the join code for the group chat (every group chat has a specific join code), and then inputting it.

Users also have the ability to post a workout including an image and a caption. Once approved by the community guidelines, this post will be added to the feed of every user in their friends list. We hope that by adding aspects of a social media site, users are encouraged to increase the frequency of their workouts in order to show their friends their progress.

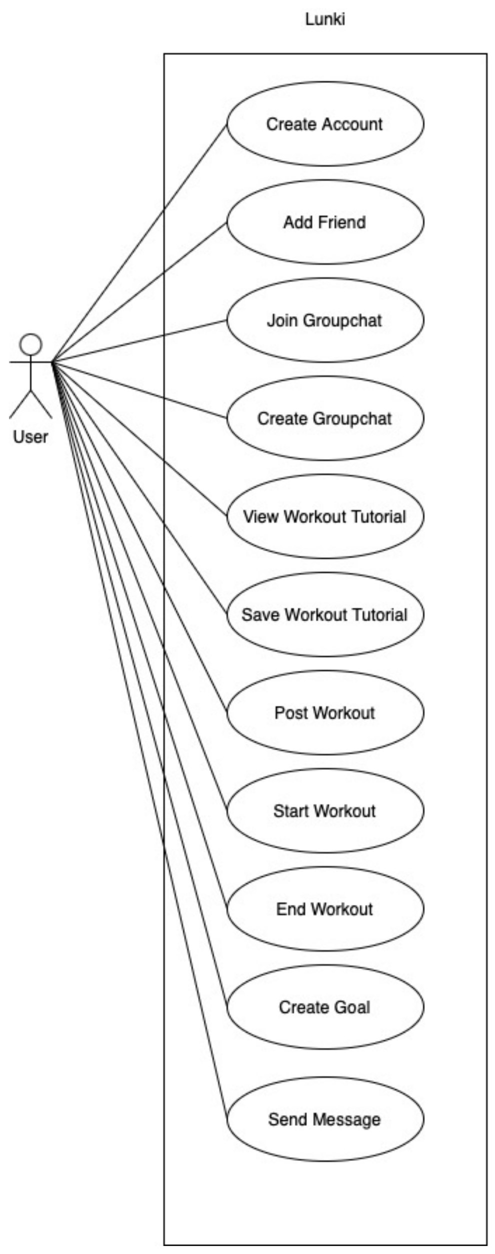
Finally, Lunki also includes a database of workout tutorials so that inexperienced users have a trusted space to learn. In our research, we found that one of the big reasons people don't go to the gym as much as they want to is because they feel inexperienced and don't know how to correctly do certain workouts. Some fear injury and judgement, while others simply don't know where to start. With Lunki, users are able to view as many workout tutorials as they please, and then add the ones that they enjoy to their own personal folder. Users can take full advantage of the Lunki database in order to become more experienced in the gym while also not spending hundreds of dollars on a personal trainer.

Lunki is intended to be the “Swiss army knife” of fitness apps. With a robust social network, combined with various motivational tools, progress monitors, and a plethora of workout tutorials, Lunki creates an experience that not many other apps can match. Working out is supposed to be a fun and rewarding activity, not a chore. The developers of Lunki are striving to change that and provide users with not just a cool new workout app, but also a change of perception towards exercising.

Class Diagram


User
- username: String
- password: String
- emailAddress: String
- friendsList: List<User>
- workoutHistory: List<Workout>
- postHistory: List<Post>
- goalList: List<Goal>
- folder: Folder
+ getUsername(): String
+ getEmail(): String
+ updateEmail(newEmail): void
+ addFriend(username): void
+ leaveGroupchat(Groupchat): void
+ startWorkout(): Time
+ endWorkout(): Time
+ addGoal(Goal): void
+ completeGoal(Goal): void
+ sendMessage(User, Message): void
Goal
- startDate: Date
- endDate: Date
- description: String
- owner: User
- completion: bool
+ createGoal(start, end, description)
+ getStartDate(): Date
+ getEndDate(): Date
+ getOwner(): User
+ getDescription(): description
+ updateDescription(newDescription): void
+ updateTimeFrame(start, end): void
Groupchat
- members: List<User>
- chatHistory: List<Message>
- chatOwner: User
+ kickUser(User): void
+ addUser(User): void
+ getMembers(): List<User>
+ getOwner(): User
+ getChatHistory(): List<Message>
+ deleteGroupchat(): void
+ updateOwner(User): void
+ createGroupchat(owner, members)
WorkoutTutorial
- description: String
- images: List<Image>
- ratings: Map<User, Integer>
- reviews: Map<User, Message>
+ getReviews(): Map<User, Message>
+ addReview(User, Message): void
+ getRatings(): Map<User, Integer>
+ addRating(User, rating): void
Workout
- owner: User
- peakHeartRate: int
- stepsWalked: int
- caloriesBurned: int
- vitals: List<Integer>
- startTime: Time
- endTime: Time
+ createWorkout(owner, startTime, endTime)
+ setVitals(List<Integer>): void
+ getVitals(): List<Integer>
+ getTimeElapsed(): Time
Message
- content: String
- attachments: List<Image>
- sender: User
+ createMessage(User, content)
+ editMessage(newMessage): void
+ getSender(): User
+ getAttachments(): List<Image>
+ getContent(): String
+ deleteMessage(): void
Post
- owner: User
- image: Image
- caption: String
- timeStamp: DateTime
+ createPost(owner, image, caption)
+ deletePost(): void
Folder
- contents: List<WorkoutTutorial>
- owner: User
+ getContents(): List<WorkoutTutorial>
+ removeWorkout(WorkoutTutorial): void
1..1
1..1
1..1
0..*
3..*
0..*
1..1
1..1
1..1
0..*
0..*
0..*
0..*
0..*
0..*
1..1

Use Case Diagram



Use Case Descriptions

Element	**Detail**
Name	Create Account
Description	This use case describes how a new user creates an account on Lunki by providing their email, username, and password
Actor(s)	User
Triggers	A user decides they would like to create a Lunki account
Preconditions	1. Email address or username provided are not attached to an existing account 2. Email address provided is valid
Postconditions	1. A new account is created and stored in the system

Main Success Path	**User creates an account using their email, username, and password. It is then stored in the system.**
Actor Actions	System Response
1. User opens Lunki	2. System displays login/create account options
3. User selects “create account”	4. System displays email address, username, and password fields
5. User completes information fields	6. System validates inputs: - Validates that email or username are not already in use - Validates password strength - Validates email address is valid 7. System creates a new account and stores it in database

Exception Path E1-7 a	**User inputs invalid information**
Actor Actions	System Response
	7. System detects invalid sign up information
	8. System displays create account error message, prompts user to input correct information
	9. Use Case ends

Element	Detail
Name	Save Workout
Description	This use case describes how a user can save a workout tutorial from the workout database
Actor(s)	User
Triggers	A user decides they would like to save a workout tutorial
Preconditions	1. User is logged into Lunki
Postconditions	1. A workout tutorial is added to the User's save folder

Main Success Path	**User views a workout from the database, saves it, and the workout tutorial is added to the User's save folder**
Actor Actions	System Response
1. User selects a workout tutorial	2. System displays workout tutorial information and options
3. User saves the workout	4. System adds the workout tutorial to the User's save folder

Element	Detail
Name	Add Friend
Description	This use case describes how a user can add a friend by inputting their username
Actor(s)	User
Triggers	A user decides they would like to add a friend
Preconditions	1. User is logged into Lunki
Postconditions	1. A friend request is sent to the other user

Main Success Path	**User inputs the username of another user they'd like to friend. A request is then sent to that user.**
Actor Actions	System Response
1. User selects add friend	2. System displays username field
3. User enters username of friend	4. System validates username is valid 5. System sends friend request to the user with that username

Alternative Path A1-1a	**User adds friend from mutual groupchat**
Actor Actions	System Response
1. User selects group chat	2. System displays groupchat members
3. User selects the profile of another user to friend	4. System displays other user's profile
5. User selects "send request"	6. System sends friend request to the other user

Exception Path E1-5 a	**User inputs invalid username**
Actor Actions	System Response
	5. System detects invalid username
	6. System displays invalid username error message, prompts user to input valid username
	7. Use Case ends

Element	Detail
Name	Join Groupchat
Description	This use case describes how a user can join a groupchat using an invite code
Actor(s)	User
Triggers	A user receives an invite code to a group chat
Preconditions	1. User is logged into Lunki 2. User has a valid invite code
Postconditions	1. User is added to a groupchat

Main Success Path	**User inputs the invite code they've received to join a group chat. They are then added to the chat.**
Actor Actions	System Response
1. User selects join group chat	2. System displays invite code field
3. User enters invite code	4. System validates invite code is valid 5. System adds user to the group chat

Exception Path E1-5 a	**User inputs invalid invite code**
Actor Actions	System Response
	5. System detects invalid invite code
	6. System displays invalid invite code error message, prompts user to input valid code
	7. Use Case ends

Element	**Detail**
Name	Start Workout
Description	This use case describes how a user can start a workout
Actor(s)	User
Triggers	A user decides they want to start a workout
Preconditions	1. User is logged into Lunki 2. User does not have an ongoing workout
Postconditions	1. A workout is started

Main Success Path	**User starts a workout**
Actor Actions	System Response
1. User selects start workout	2. System validates a workout is not already in progress 3. System begins a workout, logs date and time of start.

Exception Path E1-3 a	**Workout is already in progress**
Actor Actions	System Response
	3. System detects user has an active workout
	4. System displays error message, prompts user to end current workout before starting a new one
	5. Use Case ends

Element	Detail
Name	End Workout
Description	This use case describes how a user can end a workout
Actor(s)	User
Triggers	A user decides they want to finish their workout
Preconditions	1. User is logged into Lunki 2. User has an ongoing workout
Postconditions	1. A workout is ended 2. Workout information is logged in the User's workout history

Main Success Path	**User finishes a workout. Time elapsed and vitals are stored in user's workout history**
Actor Actions	System Response
1. User selects end workout	2. System validates there is an active workout 3. System ends workout, logs date and time of end
	4. System stores time elapsed and fetches vitals during that time (if applicable)

Exception Path E1-3 a	**No active workout**
Actor Actions	System Response
	3. System detects user does not have an active workout
	4. System displays error message, prompts user to start a workout
	5. Use Case ends

Element	Detail
Name	Set Goal
Description	This use case describes how a user can set themselves a goal
Actor(s)	User
Triggers	A user has a goal they would like to set in Lunki
Preconditions	1. User is logged into Lunki
Postconditions	1. A goal is created with the given information

Main Success Path	**User sets a workout goal with a description and timeframe.**
Actor Actions	System Response
1. User selects set goal	2. System displays description field
3. User enters a description for their goal	4. System stores description 5. System displays date field
6. User enters start and end dates for their goal	7. System validates dates are valid 8. System creates a goal with specified description and timeframe, stores in User's goal list.

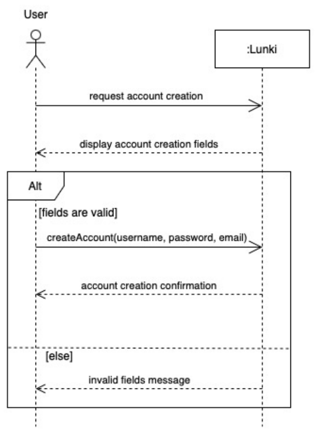
Exception Path E1-8 a	**Invalid start/end date**
Actor Actions	System Response
	8. System detects invalid dates (end date before beginning date, date has already passed, etc.)
	9. System displays error message, prompts user to enter valid dates
	10. Use Case ends

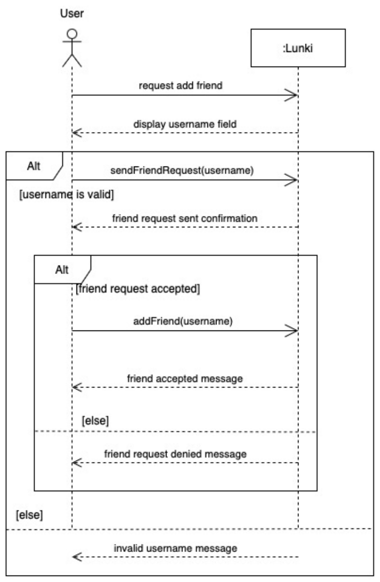
Element	**Detail**
Name	Post Workout
Description	This use case describes how a user can post a workout they have completed
Actor(s)	User
Triggers	A user completes a workout and would like to post about it
Preconditions	1. User is logged into Lunki 2. User has at least 1 friend
Postconditions	1. A post is created of the User's workout

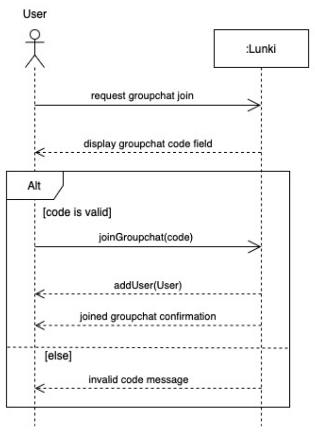
Main Success Path	**User posts a workout with a description and image, added to friend's feeds.**
Actor Actions	System Response
1. User selects create post	2. System displays workout description field
3. User enters a description of the workout they have completed	4. System stores description 5. System displays image field
6. User submits an image of themselves completing the workout	7. System validates image follows terms of service
8. User confirms workout post	9. System creates a post with the given description and image 10. System adds post to the feed of all the User's friends.

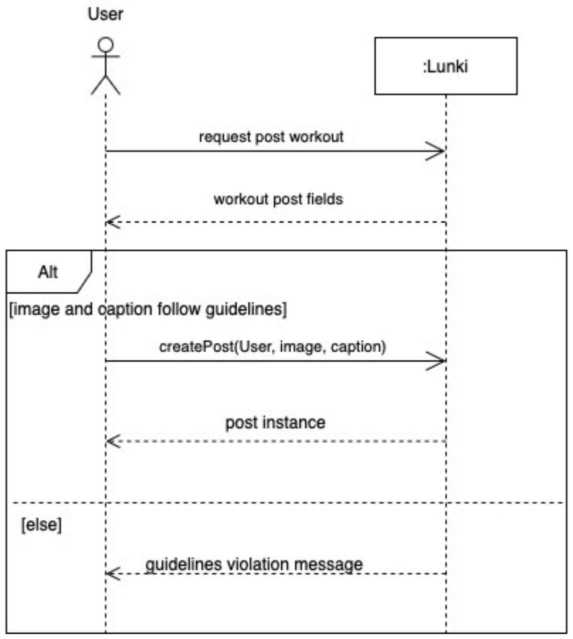
Exception Path E1-8 a	**Image does not follow terms of service**
Actor Actions	System Response
	8. System detects inappropriate or offensive image
	9. System displays inappropriate content message, prompts user to enter an appropriate image
	10. Use Case ends

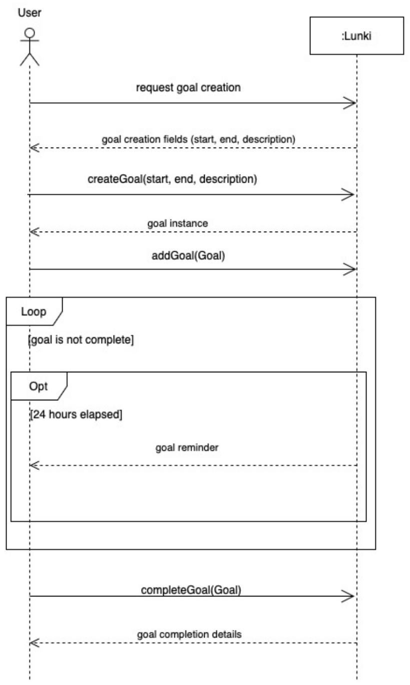
System Sequence Charts

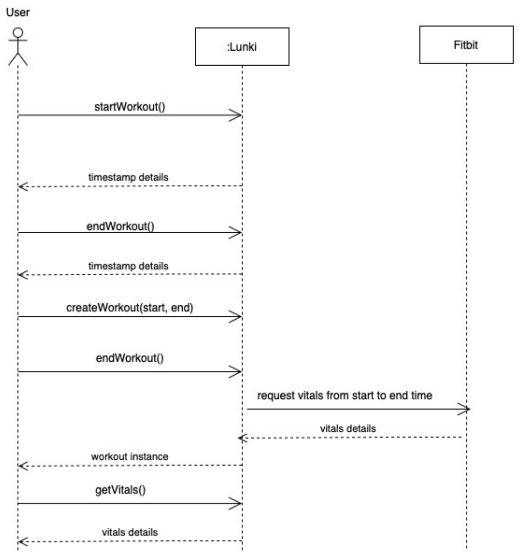

User
:Lunki
request account creation
display account creation fields
Alt
[fields are valid]
createAccount(username, password, email)
account creation confirmation
[else]
invalid fields message

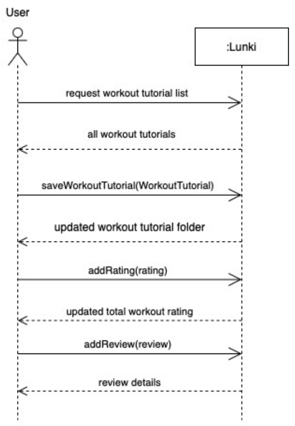





User
:Lunki
request post workout
workout post fields
Alt
[image and caption follow guidelines]
createPost(User, image, caption)
post instance
[else]
guidelines violation message


User
:Lunki
request goal creation
goal creation fields (start, end, description)
createGoal(start, end, description)
goal instance
addGoal(Goal)
Loop
[goal is not complete]
Opt
[24 hours elapsed]
goal reminder
completeGoal(Goal)
goal completion details


User
:Lunki
Fitbit
startWorkout()
timestamp details
endWorkout()
timestamp details
createWorkout(start, end)
endWorkout()
request vitals from start to end time
vitals details
workout instance
getVitals()
vitals details


User
:Lunki
request workout tutorial list
all workout tutorials
saveWorkoutTutorial(WorkoutTutorial)
updated workout tutorial folder
addRating(rating)
updated total workout rating
addReview(review)
review details