**Smart Home Gym: Detailed Specifications**

Hyojin Choi

**Abstract**

The Smart Home Gym system is a comprehensive software application designed to enhance users' fitness experiences through personalized exercise management, performance tracking, and community-based ranking. This system includes several key classes, each responsible for a specific functionality. The `User` class handles user information, credentials, exercise records, and login status. The `Feed` class provides users with relevant exercise articles to support their fitness journey. The `MembershipRegistration` class manages user registration with validation for unique usernames, secure passwords, and correctly formatted emails. The `LoginSystem` class authenticates users, enabling secure access to personalized features. The `Calendar` class allows users to schedule workouts, set goals, and track daily exercise activities. The `ExerciseSystem` class logs exercise sessions and provides feedback on workout performance, helping users meet their goals. Finally, the `Ranking` class calculates user rankings based on exercise achievements and allows users to share their standing within the community. Each class includes detailed specifications, including methods, parameters, and error handling requirements, to ensure a robust, user-friendly experience. Through this system, the Smart Home Gym aims to provide a motivating and structured environment for consistent exercise habits, personalized guidance, and community engagement.

**Class: User**

**Description**: Represents a user in the Smart Home-Gym system, including their credentials, exercise records, and rank.

**Methods:**

1. `\_\_init\_\_(username: str, password: str, email: str)`
   * Description: Initializes a new user with a username, password, and email. Also initializes an empty exercise list and sets the initial rank to None.
   * Parameters:
     + `username` (str): The username of the user.
     + `password` (str): The password of the user.
     + `email` (str): The email address of the user.
     + `calendar` (dictionary): The calendar of the user. Saves exercises to be done on specific date, the key being the date in strings, value being the list of exercises. An exercise is a dictionary with the exercise name (str) as the key and duration (int) as its value
     + `exercises` (list): A list of exercises finished today. An exercise is a dictionary with the exercise name (str) as the key and duration (int) as its value
     + `rank` (int): The user’s rank for the day, initialized to None.
     + `logged\_in` (bool): Tracks the login status of the user, initially set to False.
   * Returns: None
2. Example Inputs/Outputs:
   * Valid inputs:

|  |
| --- |
| user = User('johndoe', 'password123!', 'johndoe@example.com')  print(user.username) # Output: johndoe  print(user.email) # Output: johndoe@example.com  print(user.exercises) # Output: []  print(user.rank) # Output: None |

**Class: Feed**

**Description**: Manages the information feed for a specific user in the Smart Home-Gym system.

**Methods:**

1. `\_\_init\_\_(user: User)`
   * Description: Initializes the feed system for a specific user with a list of articles.
   * Parameters:
     + `user (User)`: The user for whom the feed is generated.
   * Returns: None
2. `add\_feed(article: str)`
   * Description: Adds a new article to the user's feed.
   * Parameters:
     + `article (str)`: The title of the article to be added.
   * Returns: None
   * Requirements
     + Article:
       - Empty string (invalid).
       - String with leading or trailing spaces (valid but trimmed).
       - Very long titles (more than 256 characters).
   * Error Handling
     + Raises a ValueError if the article title is empty or exceeds 256 characters in length.
3. `show\_feed() -> List[str]`
   * Description: Displays a list of articles relevant to the user's interests.
   * Parameters: None
   * Returns: `List[str]` - A list of article titles.
4. Example Inputs/Outputs:
   * Valid inputs:

|  |
| --- |
| user = User('johndoe', 'password123!', 'johndoe@example.com')  feed = Feed(user)  feed.add\_feed("New Article on Fitness")  articles = feed.show\_feed()  print(articles)  # Output: [  # "10 Tips for Effective Home Workouts",  # "How to Stay Motivated to Exercise",  # "Best Exercises for Building Strength",  # "New Article on Fitness"  # ] |

* + Invalid inputs:

|  |
| --- |
| user = User('johndoe', 'password123!', 'johndoe@example.com')  feed = Feed(user)  try:  feed.add\_feed("") # Empty string  except ValueError as e:  print(e) # Output: "Article title cannot be empty."  try:  long\_title = "A" \* 257 # Title exceeding 256 characters  feed.add\_feed(long\_title)  except ValueError as e:  print(e) # Output: "Article title cannot exceed 256 characters." |

**Class: MembershipRegistration**

**Description**: Manages the registration of users in the Smart Home-Gym system.

**Methods:**

1. `\_\_init\_\_()`
   * Description: Initializes the membership registration system with an empty user list.
   * Parameters: None
   * Returns: None
2. `register\_user(username: str, password: str, email: str) -> bool`
   * Description: Registers a new user if the username does not already exist.
   * Parameters:
     + `username` (str): The username of the user. Must be a string of alphanumeric characters (English letters and numbers only), 3-20 characters long. No leading or trailing spaces.
     + `password` (str): The password of the user. Must be a string, minimum 8 characters long, and include at least one special character and one number. No restriction on the position of special characters.
     + `email` (str): The email address of the user. Must be in a valid email format (example@domain.com) and can be up to 254 characters long. Leading or trailing spaces will be removed.
   * Returns: `bool` - True if registration is successful, False otherwise.
   * Requirements
     + Ensures the username and email are unique among registered users.
     + Username:
       - Length must be 3-20 characters.
       - Contains special characters (invalid).
       - Contains leading or trailing spaces (invalid).
     + Password:
       - Length is minimum 8 characters.
       - Lacks special characters or numbers (invalid).
     + Email:
       - Valid formats include subdomains and different TLDs (e.g., "user@sub.domain.com", "user@domain.co.uk").
       - Invalid formats include missing '@', missing domain, invalid characters, or spaces.
3. Example Inputs/Outputs:
   * Valid inputs:

|  |
| --- |
| membership = MembershipRegistration()  success = membership.register\_user('johndoe', 'password123!', 'johndoe@example.com')  print(success) # Output: True  success = membership.register\_user('janedoe', 'password123!', 'janedoe@example.com')  print(success) # Output: True |

* + Invalid inputs:

|  |
| --- |
| membership = MembershipRegistration()  membership.register\_user('johndoe', 'password123!', 'johndoe@example.com')  try:  success = membership.register\_user('johndoe', 'newpass456!', 'john\_doe@example.com')  except ValueError as e:  print(e) # Output: "Username already exists"  try:  success = membership.register\_user('janedoe', 'newpass456!', 'johndoe@example.com')  except ValueError as e:  print(e) # Output: "Email already exists" |

**Class: LoginSystem**

**Description**: Manages user login in the Smart Home Gym system.

**Methods:**

1. \_\_init\_\_(membership: MembershipRegistration)
   * Description: Initializes the login system with a membership registration instance and no logged-in user.
   * Parameters:
     + membership (MembershipRegistration): The membership registration instance.
   * Returns: None
2. login(username: str, password: str) -> bool
   * Description: Authenticates a user with a given username and password.
   * Parameters:
     + username (str): The username of the user.
     + password (str): The password of the user.
   * Returns: bool - True if login is successful, False otherwise.
   * Requirements:
     + If the credentials are correct, sets `logged\_in\_user` to the matched user and changes their `logged\_in` status to True.
     + If the credentials are incorrect, leaves `logged\_in\_user` as None.
     + Username and password combination:
       - Does not match any registered user.
       - Username is case-sensitive and must match exactly.
       - Password is case-sensitive and must match exactly.
3. logout()
   * Description: Logs out the currently logged-in user.
   * Parameters: None
   * Returns: None
   * Requirements:
     + Logging out when no user is logged in should have no effect.
4. Example Inputs/Outputs:
   * Valid inputs:

|  |
| --- |
| membership = MembershipRegistration()  membership.register\_user('johndoe', 'password123!', 'johndoe@example.com')  login\_system = LoginSystem(membership)  success = login\_system.login('johndoe', 'password123!')  print(success) # Output: True  print(login\_system.logged\_in\_user.username) # Output: johndoe  login\_system.logout()  print(login\_system.logged\_in\_user) # Output: None |

* + Invalid inputs:

|  |
| --- |
| membership = MembershipRegistration()  membership.register\_user('johndoe', 'password123!', 'johndoe@example.com')  login\_system = LoginSystem(membership)  success = login\_system.login('johndoe', 'wrongpassword')  print(success) # Output: False  print(login\_system.logged\_in\_user) # Output: None |

**Class: Calendar**

**Description**: Manages the exercise schedule and goals for a specific user in the Smart Home-Gym system.

**Methods:**

1. `\_\_init\_\_(user: User)`
   * Description: Initializes the calendar system for a specific user.
   * Parameters:
     + `user (User)`: The user for whom the calendar is managed.
   * Returns: None
   * Behavior: Raises a ValueError if the user is not logged in.
2. ‘input\_workout(date: str, exercise\_name: str, duration: int)’
   * Description: Inputs an exercise and its duration for a specific date.
   * Parameters:
     + date (str): The date of the exercise (format: YYYY-MM-DD).
     + exercise\_name (str): The name of the exercise. Cannot be empty or exceed 256 characters.
     + duration (int): The duration of the exercise in minutes. Must be a positive integer.
   * Returns: None
   * Requirements:
     + Date:
       - Valid date format but in the future.
     + Exercise name:
       - Invalid for empty string or very long names (more than 256 characters).
     + Duration:
       - Invalid for negative value, zero, or non-integer.
   * Error Handling:
     + Raises a ValueError if the date format is invalid, exercise\_name is empty or too long, or duration is not positive.
3. ‘show\_plan(date: str) -> List[dict]’
   * Description: Displays the user's exercise schedule for a given date.
   * Parameters:
     + date (str): The date for which the exercise schedule is requested. Must be in YYYY-MM-DD format.
   * Returns: List[dict] - A list of exercise records for the specified date. Returns an empty list if no exercises are recorded.
   * Requirements:
     + Date Format:
       - Must be a valid date format (YYYY-MM-DD).
       - Dates must be valid calendar dates (e.g., no '2024-02-30').
       - Leading or trailing whitespace in the date should be trimmed, but the date itself should not contain spaces.
     + Future Dates:
       - The function should handle future dates by returning an empty list.
     + Empty Exercises:
       - If no exercises are recorded for the given date or the user has no exercises at all, the function should return an empty list.
   * Error Handling:
     + Raises a ValueError if the date format is invalid.
4. Example Inputs/Outputs:
   * Valid inputs:

|  |
| --- |
| user = User('johndoe', 'password123!', 'johndoe@example.com')  user.logged\_in = True  calendar = Calendar(user)  calendar.input\_workout('2024-08-01', 'Running', 30)  print(user.calendar)  # Output: {'2024-08-01': [{'name': 'Running', 'duration': 30}]}  print(calendar.show\_plan('2024-08-01'))  # Output: [{'name': 'Running', 'duration': 30}] |

* + Invalid inputs:

|  |
| --- |
| user = User('johndoe', 'password123!', 'johndoe@example.com')  user.logged\_in = True  calendar = Calendar(user)  # Invalid date format  try:  calendar.input\_workout('01-08-2024', 'Running', 30)  except ValueError as e:  print(e) # Output: "Invalid date format. Expected YYYY-MM-DD."  # Empty exercise name  try:  calendar.input\_workout('2024-08-01', '', 30)  except ValueError as e:  print(e) # Output: "Exercise name cannot be empty." |

**Class: ExerciseSystem**

**Description**: Manages exercise activities and feedback for a specific user in the Smart Home-Gym system.

**Methods:**

1. `\_\_init\_\_(user: User)`
   * Description: Initializes with a User instance and assigns it to self.user for managing exercise records.
   * Parameters:
     + `user (User)`: The user performing the exercise.
   * Returns: None
2. `start\_exercise(exercise\_name: str, duration: int)`
   * Description: Records a new exercise activity for the user with validations.
   * Parameters:
     + `exercise\_name (str)`: The name of the exercise.
     + `duration (int)`: The duration of the exercise in minutes.
   * Returns: None
   * Requirements:
     + Exercise name:
       - Invalid for empty string, contain only alphanumeric characters and spaces.
       - cannot exceed 256 characters.
     + Duration:
       - Must be an integer and positive.
3. ‘provide\_feedback(date: str) -> str’
   * Description: Provides feedback on the user’s total exercise duration for a specified date, comparing it to their planned goal.
   * Parameters:
     + date (str): The date for which feedback is requested (format: YYYY-MM-DD).
   * Returns: str - A message about the total exercise duration and comparison with the planned goal.
   * Requirements:
     + Date must be in YYYY-MM-DD format.
     + Dates must be valid calendar dates.
     + No leading or trailing spaces in the date.
     + Handle future dates by returning appropriate messages.
     + Handle dates with no exercises by returning a message.
     + Raise ValueError for invalid, whitespace, or non-existent dates.
   * Error Handling:
     + Raises a ValueError if date is improperly formatted or contains leading/trailing spaces.
     + Checks if the month and day values in date are valid.
4. Example Inputs/Outputs:
   * Valid inputs:

|  |
| --- |
| user = User('johndoe', 'password123!', 'johndoe@example.com')  exercise\_system = ExerciseSystem(user)  # Valid input  exercise\_system.start\_exercise('Running', 30)  print(user.exercises)  # Output: [{'name': 'Running', 'duration': 30}]  user.calendar['2024-08-01'] = [{'name': 'Running', 'duration': 30}] # User's planned goal  print(exercise\_system.provide\_feedback('2024-08-01'))  # Output: "Total exercise duration: 30 minutes, 0 minutes short of your goal" |

* + Invalid inputs:

|  |
| --- |
| # Invalid exercise name  try:  exercise\_system.start\_exercise('', 30)  except ValueError as e:  print(e) # Output: "Exercise name cannot be empty."  # Invalid date format  try:  print(exercise\_system.provide\_feedback('08-01-2024'))  except ValueError as e:  print(e) # Output: "Invalid date format. Expected format is YYYY-MM-DD." |

**Class: Ranking**

**Description**: Manages the ranking of users based on their exercise performance in the Smart Home-Gym system.

**Methods:**

1. `\_\_init\_\_(membership: MembershipRegistration)`
   * Description: Initializes the ranking system with a MembershipRegistration instance, allowing it to access all registered users.
   * Parameters:
     + `membership (MembershipRegistration)`: The membership registration instance.
   * Returns: None
2. `calculate\_ranking()`
   * Description: Calculates and updates rankings for all users based on their total exercise durations.
   * Parameters: None
   * Returns: None
   * Behavior:
     + Sorts users in descending order by total exercise duration.
     + Assigns the same rank to users with identical durations to handle ties.
     + Sets rank to None for users without recorded exercises.
   * Requirements:
     + If there are no users, do nothing.
     + Exclude users without recorded exercises from rankings.
     + Ensure a single user with exercises is ranked #1.
     + Handle ties by assigning the same rank to users with identical exercise durations.
3. `get\_user\_ranking(username: str) -> int`
   * Description: Retrieves the rank of a specified user if they are logged in.
   * Parameters:
     + `username (str)`: The username of the user.
   * Returns: `int` - The rank of the user, or `None` if the user is not found.
   * Requirements:
     + Username:
       - User does not exist.
       - User has no recorded exercises.
       - Username is case-sensitive and must match exactly.
       - Whitespace username.
   * Error Handling:
     + Raises a ValueError if the username contains leading or trailing whitespace or if the user is not logged in.
4. `share\_ranking\_on\_social\_media(username: str) -> str`
   * Description: Shares the user’s ranking on social media if they are logged in and have a recorded rank.
   * Parameters:
     + `username (str)`: The username of the user.
   * Returns: `str` - A message about the user's ranking, or an appropriate message if no ranking is available.
   * Requirements:
     + Validate the username is not empty or whitespace.
     + Check if the user exists in the system.
     + Ensure the user has recorded exercises to have a ranking.
     + Return appropriate messages based on the user's ranking status or errors encountered.
   * Error Handling:
     + Raises a ValueError if the username contains whitespace, is empty, or if the user is not logged in.
5. Example Inputs/Outputs:
   * Valid inputs:

|  |
| --- |
| membership = MembershipRegistration()  membership.register\_user('johndoe', 'password123!', 'johndoe@example.com')  membership.register\_user('janedoe', 'password456!', 'janedoe@example.com')  user\_john = next(user for user in membership.users if user.username == 'johndoe')  user\_jane = next(user for user in membership.users if user.username == 'janedoe')  user\_john.exercises.append({'name': 'Running', 'duration': 30})  user\_jane.exercises.append({'name': 'Cycling', 'duration': 45})  ranking\_system = Ranking(membership)  ranking\_system.calculate\_ranking()  print(user\_john.rank) # Output: 2  print(user\_jane.rank) # Output: 1  user\_john.logged\_in = True  print(ranking\_system.get\_user\_ranking('johndoe'))  # Output: 2  print(ranking\_system.share\_ranking\_on\_social\_media('johndoe'))  # Output: "User johndoe is ranked #2 in the Smart Home-Gym community!" |

* + Invalid inputs:

|  |
| --- |
| # Invalid username format  try:  ranking\_system.get\_user\_ranking(' johndoe ') # Contains leading/trailing spaces  except ValueError as e:  print(e) # Output: "Invalid username format."  # User not logged in  try:  ranking\_system.share\_ranking\_on\_social\_media('janedoe')  except ValueError as e:  print(e) # Output: "User must be logged in to share ranking on social media." |