FRANKFURT UNIVERSITY OF APPLIED SCIENCES

FINAL PROJECT REPORT

?Watch Ultimate Pro Max Special Limited 2XXX Edition

Author:
PHAM NGUYEN QUY NAM - 1527656
PHAN KHANH NGHI - 1518564
NGUYEN DAI MINH - 1519743
PHAN HUYNH ANH THU - 1527362
NGUYEN NHUT THANH - 1520136

Supervisor:
DR. ROBERT LOKAICZYK

Burning Sprint Group
Faculty of Engineering

August 12, 2024

Declaration of Authorship

We, Phan Khanh Nghi
Phan Huynh Anh Thu
Nguyen Dai Minh
Pham Nguyen Quy Nam
Nguyen Nhut Thanh, declare that this final project report titled, "?Watch Ultimate Pro Max Special Limited 2XXX Edition" and the work presented in it are our own. We confirm that:

- This project, involving several authors, was mainly completed during our time in a software development analysis project at Frankfurt University of Applied Sciences.
- If any section of this report has been previously presented for a degree or any other qualification at this University or elsewhere, it has been explicitly mentioned.

Signed:

Date: 9th February 2024

-

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Abstract

Faculty Name
Faculty of Engineering

?Watch Ultimate Pro Max Special Limited 2XXX Edition

by Phan Khanh Nghi
Phan Huynh Anh Thu
Nguyen Dai Minh
Pham Nguyen Quy Nam
Nguyen Nhut Thanh

This report presents a comprehensive analysis of the software development process undertaken for the creation of a Smart Watch application. The project, conducted at Frankfurt University of Applied Sciences, aimed to design and implement innovative functionalities tailored to enhance user experience on wearable devices. The report provides insights into the project's objectives, methodologies employed, and the technologies utilized, with a focus on the integration of Magic Systems of Systems Architect (Magic Draw) and Balsamiq Wireframes for efficient diagramming and UI prototyping. Additionally, the collaborative nature of the project is highlighted, acknowledging the support received from Professor Dr. Robert Lokaiczyk and the use of a licensing server. The findings and outcomes of the analysis offer valuable perspectives for future developments in the realm of smartwatch applications....

Acknowledgements

On the momentous occasion of successfully completing our smart watch project, we extend our sincere appreciation to Professor Dr. Robert Lokaiczyk, whose invaluable support significantly contributed to the realization of this endeavor. Our gratitude is particularly directed towards the provision of the licensing server enabling access to the Magic Systems of Systems Architect (Magic Draw) and the Balsamiq Wireframes tool. These resources played a pivotal role in facilitating the creation of intricate diagrams and UI prototypes throughout the project. The mentorship and resources provided by Professor Dr. Lokaiczyk have been instrumental in the achievement of our project goals, for which we are truly grateful

Contents

Chapter 1

Introduction

1.1 Project Description

This report is aimed at documenting as well as presenting the development of various functionalities for our smart watch. This includes requirement analysis, creating UML diagrams, and designing UI prototypes.

Our project will start with brainstorming some ideas on Trello, adding granularity to our functions and from then we will draw the appropriate diagrams for each function. We have 5 group members so we decided to do 5 functions and additional functions if necessary.

1.2 Ideas

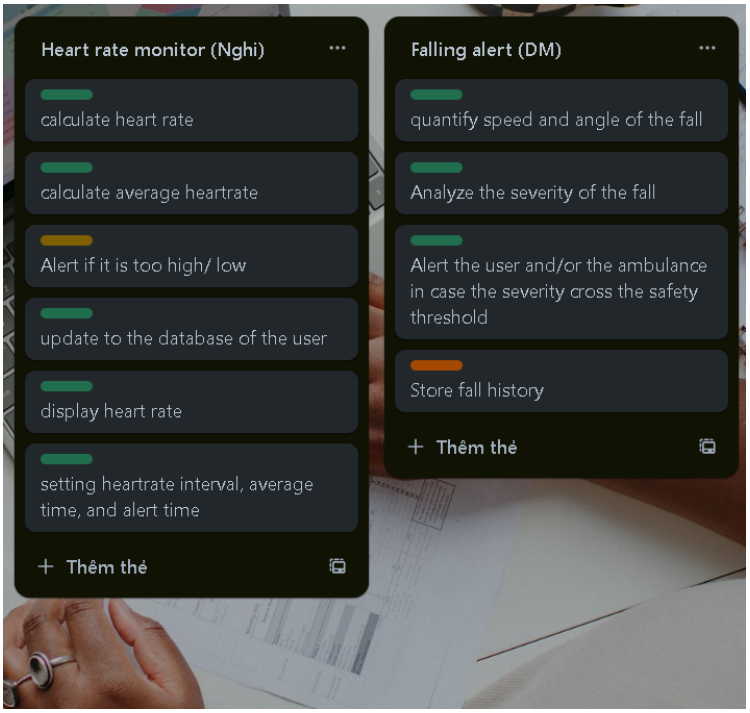

FIGURE 1.1: First two functions

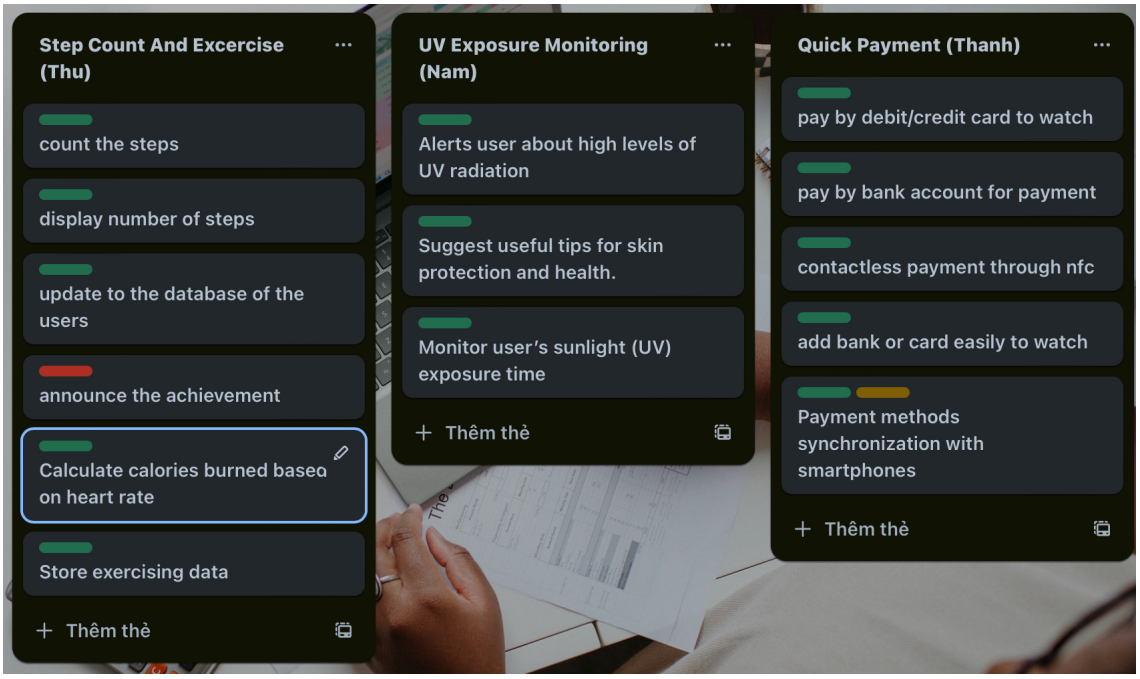

FIGURE 1.2: Another two functions

The main selling point of our watch will be fall detection alert as well as UV exposure warning function.

1.3 Implementations

1.3.1 Heart Rate Monitor

In the beginning, we decided to add a simple feature that any smartwatch must have which is to track a user's heart rate and improve upon it by adding an emergency warning and health tips for users based on their heart rate.

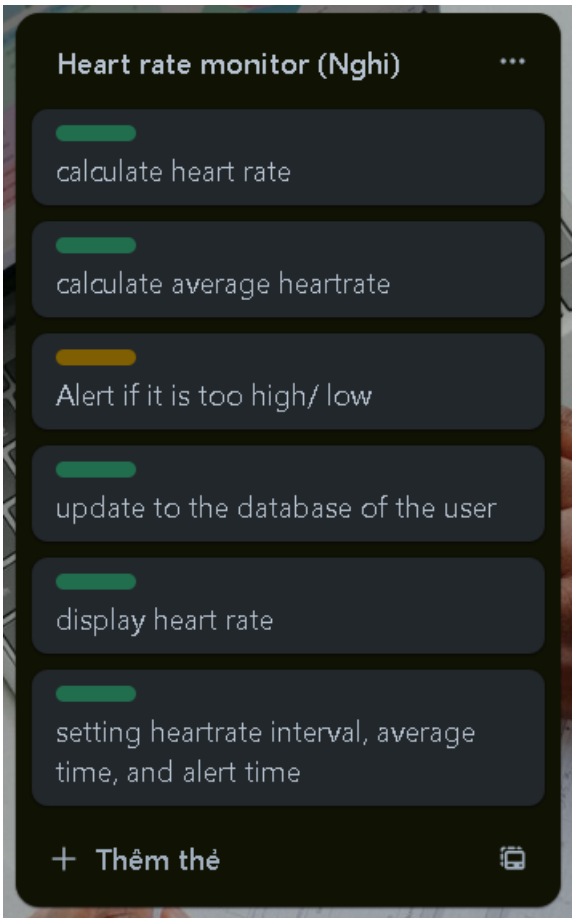

FIGURE 1.3: Heart rate idea

1.3.2 Step Count and Exercise

Next, we also have to implement another obvious functionalities of a smartwatch which is step count and exercise monitoring which we also added health tips and calories calculations.

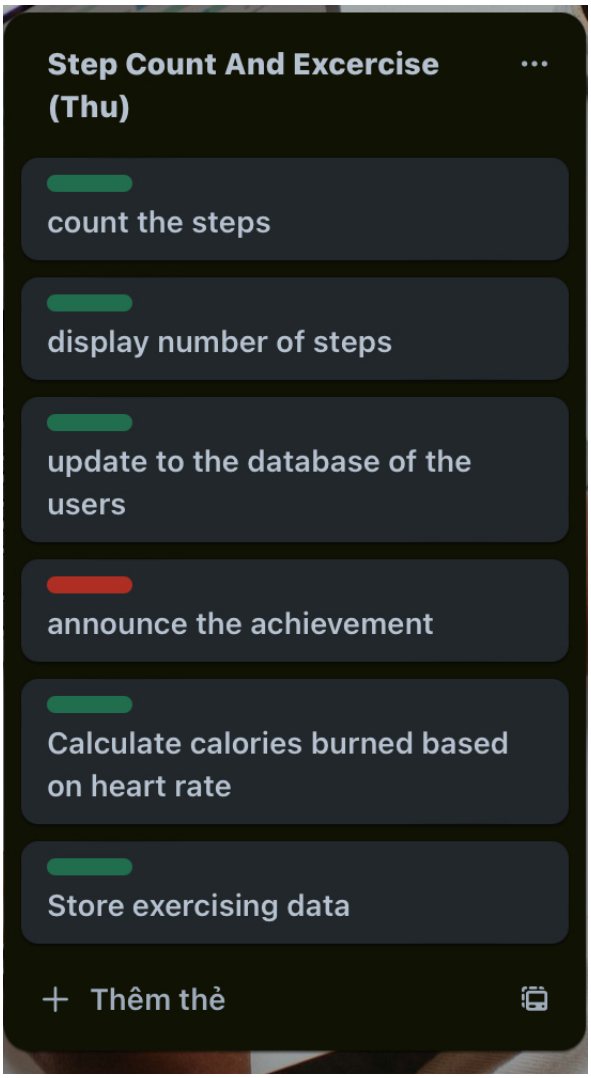

FIGURE 1.4: Step count idea

1.3.3 Falling Alert

When we think about exercise, we also think that it is very possible to attain injuries for example you can fall if you trip during your jog and that's how we came up with falling alert when user experience any form of impact.

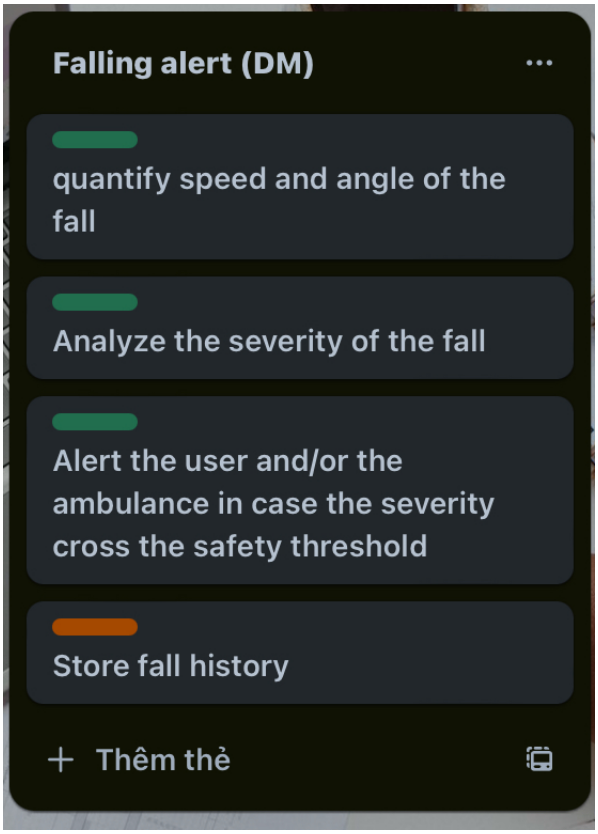

FIGURE 1.5: Falling alert idea

1.3.4 UV Exposure Sensor

On the topic of exercising, we also think it is relevant that a smartwatch should sense ultraviolet to see if it is safe to be exposed to the current weather and give some advice regarding the current UV level.

FIGURE 1.6: UV exposure idea

1.3.5 Quick Payment

Other than some essential health features, a smartwatch should also be capable of some convenient functionality which is being able to add a credit card, or bank account or synchronize with smartphones to make contactless payments.

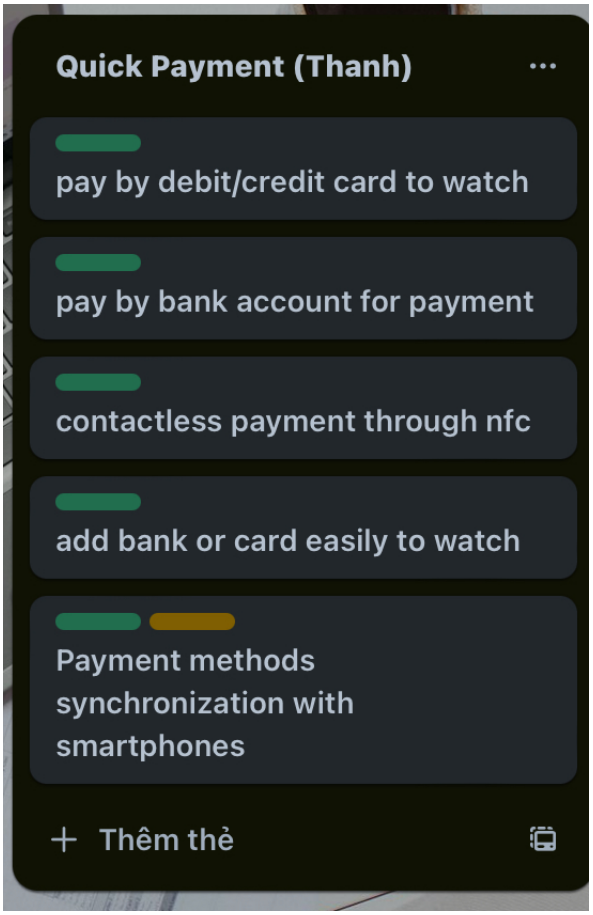

FIGURE 1.7: Quick payment idea

1.3.6 Weather alert

Another convenient functionality that should be of high priority for a smartwatch is the capability to show weather-related information such as temperature humidity or warning of any weather incident that is to come. (This is a function we thought of later on in the process so it will not be appearing in the ideas board)

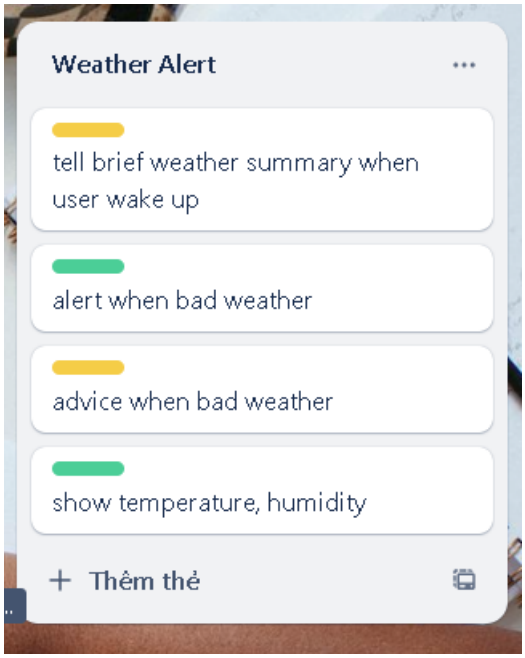

FIGURE 1.8: Weather Alert idea

1.3.7 Clock System

An important function that we forgot was that a watch should always be able to tell time, therefore adding in a clock system is of high necessity.

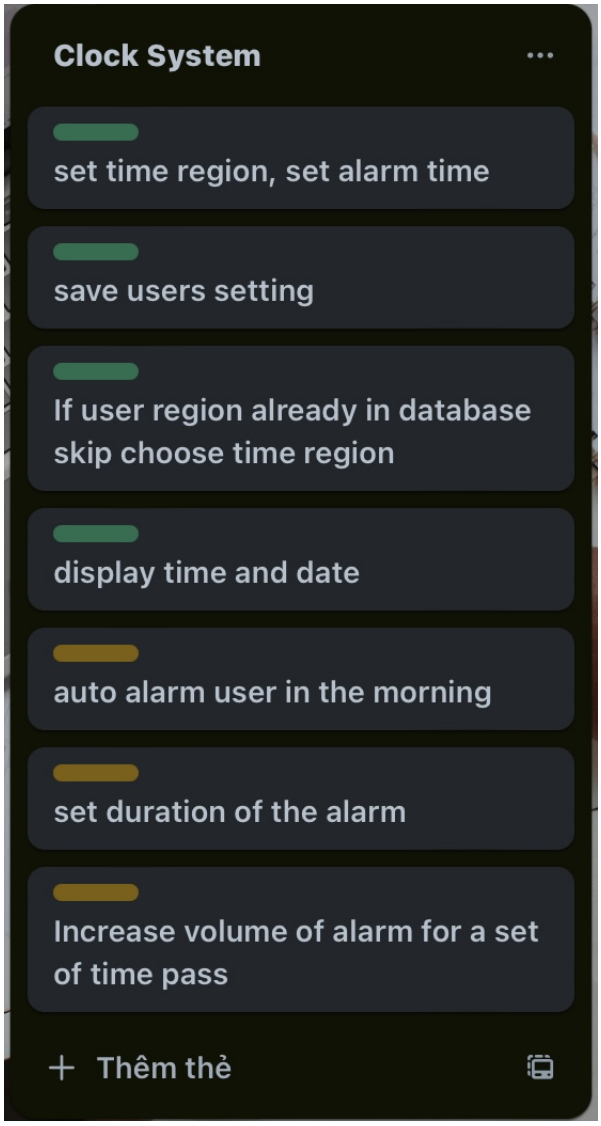

FIGURE 1.9: Clock System idea

P.S: These are just primitive ideas and functions inside each requirement may change in the diagram to fit with our member's way of thinking.

Chapter 2

Heart Rate Monitor

Heart rate monitoring is a functionality that measures users heart rate, presenting the data on the screen. In the event of the heart rate not in a user-defined interval, an alert is triggered, providing timely notifications for potential health concerns. The feature also allows users to customize the time duration for calculating, and displaying the average heart rate. This part is written by Phan Khanh Nghi.

2.1 Requirement Analysis

2.1.1 Snow cards

Requirements Type: Functional
For Whom: customer
User Satisfaction: High
User Dissatisfaction: High
Description: The smartwatch will continuously monitor the user's heart rate in real-time, providing users with the ability to set a personalized heart rate threshold. Should the detected heart rate exceed or fall below this threshold, the smartwatch will promptly generate a clear and visible alert. This alert will be accompanied by a subtle notification sound or vibration, ensuring users are immediately informed of any significant deviation. Users will also be able to customize the time period for calculating the average heart rate. The resulting average heart rate for the specified duration will be displayed on the smartwatch screen, allowing users to assess their heart rate trends over that period. Furthermore, the calculated average heart rate will be securely stored in a database. This storage capability enables users to access and review their historical average heart rate data over time.

2.1.2 Use Case Analysis

Name	Heart Rate Monitoring
ID	26
Description	The smartwatch monitors users real-time heart rate, displaying it on the screen. Users can customize their heart rate threshold, prompting an alert with sound or vibration. Customization also applies to the time period for calculating the average heart rate, allowing flexibility for monitoring. The resulting average heart rate is shown on the smartwatch. The user-friendly interface facilitates configuration of heart rate settings, thresholds, and monitoring durations. Notably, the average heart rate is stored for users to access historical data for health tracking over time.
Trigger	For heart rate monitoring: After the smartwatch is worn and turned on. For heart rate alert: If the user's heart rate surpasses the user-defined heart rate threshold.
Pre-conditions	For heart rate monitoring: After the smartwatch is worn and turned on. For heart rate alert: If the heart rate surpasses the user-defined heart rate threshold.
Post-conditions	Alert is turned off or emergency contact is reached
Basic Flow	-
Description	This is the situation where the user's heart rate surpasses the user-defined heart rate threshold.
Actions	1. User's heart rate significantly increases or decreases. 2. Sensors calculate speed and severity. 3. The sensor calculates the heart rate. 4. If the user's heart rate surpasses the user-defined heart rate threshold, alert the user. 5. If the user turns off the alert or if the countdown timer for the user-set alert reaches 0, the alert will be turned off.

2.2 UML diagrams

2.2.1 Use Case Diagram

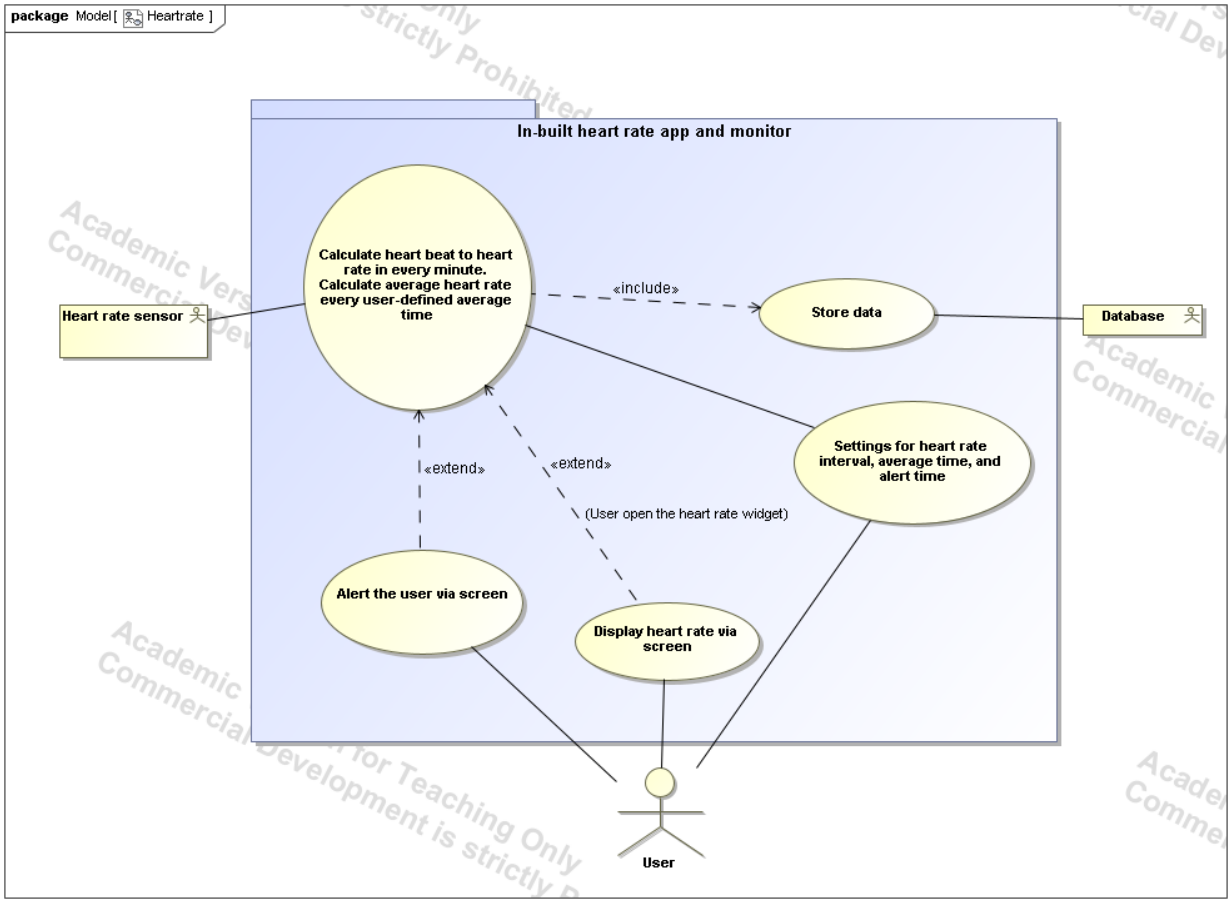

FIGURE 2.1: Heart rate monitoring use case diagram

The use case diagram illustrates the features of a smartwatch designed for heart rate monitoring. Users can employ the built-in heart rate app to monitor their heart rate. The smartwatch communicates with a dedicated heart rate sensor for data collection. The watch performs calculations and alerts based on user-defined parameters, including heart rate interval, average time, and alert time. All collected data is stored in a database for future reference.

Additionally, the diagram highlights two extension points for added functionality. The first extension point is activated when the user opens the heart rate widget, displaying the current heart rate and providing alerts if it surpasses a specified threshold. The second extension point allows for the calculation of the average heart rate at user-defined intervals.

- **User:** The person who wears the smartwatch and wants to track their heart rate.
- **Heart Rate Sensor:** The device that measures the user's heartbeat.
- **Database:** The repository that stores the user's heart rate data.

The user initiates the heart rate app, and the smartwatch measures their heart rate. The app displays the current heart rate, calculates the average heart rate, and stores it in the database.

The user can set the following parameters for heart rate tracking:

- **Heart Rate Interval:** The highest and lowest heart rate thresholds that the user sets.
- **Average Time:** The time at which the watch calculates the average heart rate.
- **Alert Time:** The time at which the watch automatically turns off alerts if the user's heart rate exceeds a certain threshold.

2.2.2 Sequence Diagram

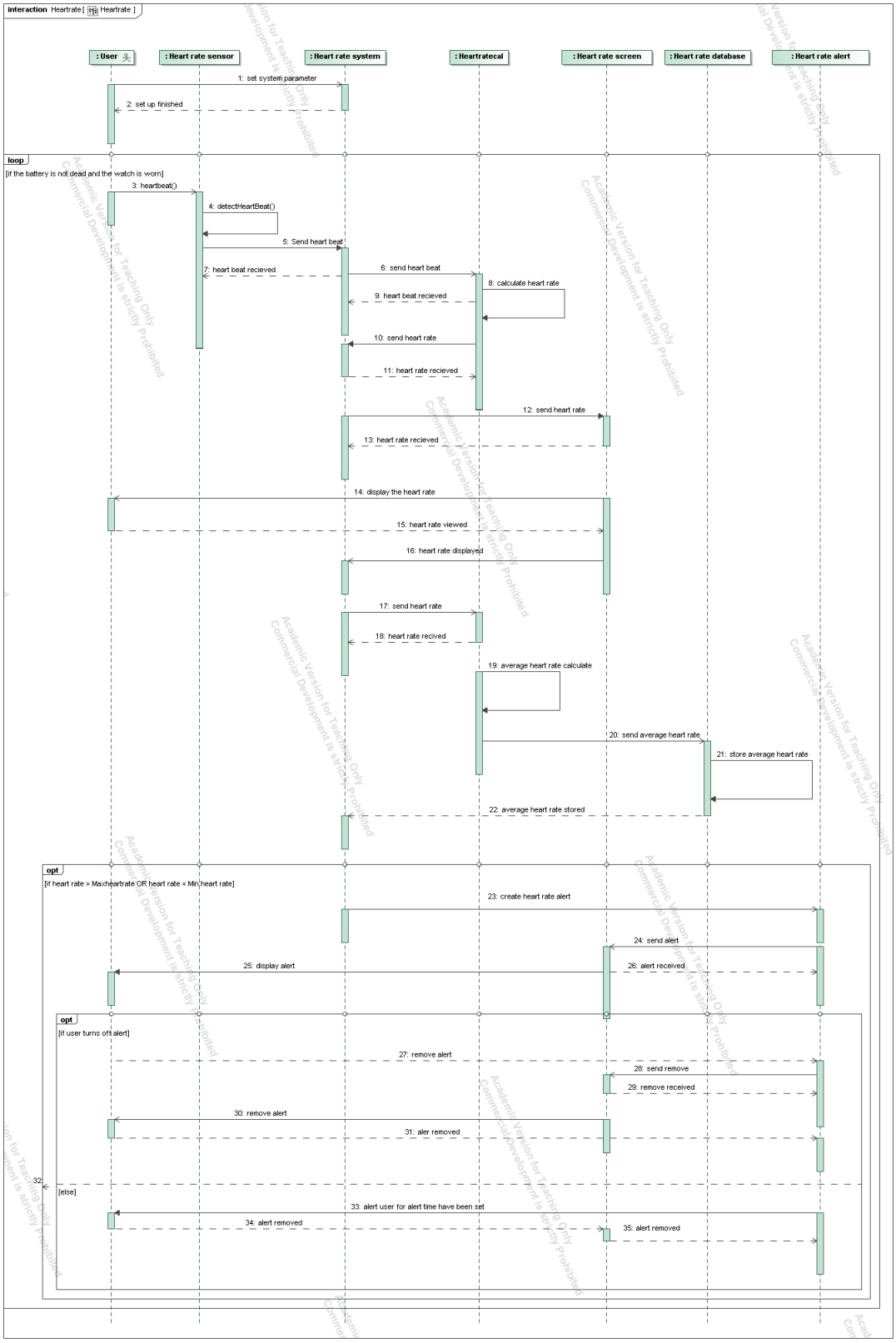

FIGURE 2.2: Heart rate monitoring class diagram

The interaction between a smartwatch user, a heart rate sensor, a heart rate app, and a heart rate database for heart rate monitoring is shown in the sequence diagram. In order for other heart rate system operations to take out their calculations,

the user must first set up system parameters including average time, heart rate interval, and alert time. The loop begins to activate the heart rate monitoring feature as soon as the user puts on the watch and sets up the system parameters. The only occasions when this loop breaks are when the user takes off the watch or when the battery runs out.

First, the user's heart rate is continuously measured by the heart rate sensor, which then transmits the data to the heart rate system. The user's heart rate varies while exercising or doing other activities, and the sensor detects these variations precisely.

The sensor provides the heart rate data to the heart rate system, which then transmits it straight to the heart rate calculator. The heart rate is then computed by the heart rate calculator and returned to the heart rate system. The heart rate screen receives the heart rate data immediately from the system. If the heart rate widget is activated, the user sees the most recent heart rate on the screen. In order to determine the average heart rate during a certain time period, the heart rate system sends data to the heart rate calculator after receiving it from the system. In the heart rate database, this average heart rate number is also kept for further use and analysis.

The heart rate alerting feature of the heart rate system alerts the user to the possibility of an irregular heart rate if their heart rate exceeds a preset threshold. With this proactive approach, consumers can treat any underlying health concerns right away.

After the alert have been created, the user will have 2 options, if the the user want to remove the alert immediately, "CANCEL" button on the screen can be pressed to cancel the alert. If the the user does not directly cancel the alert, the alert time will count down and it will automatically remove the aler when the time reachs 0s.

The real-time communication between the user of the smartwatch and the heart rate monitoring system is clearly depicted in the sequence diagram. It demonstrates how the system continuously gathers, handles, and analyzes heart rate data in order to deliver timely alerts and real-time feedback. The diagram also shows how user-defined parameters are incorporated into the system design to allow for customization of the heart rate monitoring experience.

Detailed Breakdown of Lifelines and Events

Lifelines:

- Users: The user activates the smartwatch's heart rate app.
- Heart rate sensor: The sensor continuously records the user's heart beat data.
- Heart rate system: the main system that controls the heart rate monitor application, transmits data to other systems, and initiates the activation of the heart rate alerting feature.
- Heartratecal: this calculator calculates the average heart rate and heart rate from heartbeats.
- Heart rate screen: the screen displays the user's heart rate as well as heart rate alerts.
- Heart rate database: the average heart rate is stored in the database.
- Heart rate alert: the alert generates a heart rate alert, which it automatically removes after the alert time or if the user cancels it.

Events:

- Heartbeat detection: The user's heart rate is continuously recorded by the heart rate sensor.
- Send heartbeat: The heart rate sensor sends the heart rate system of the observed heartbeat.
- Heart rate send: The heart rate system sends the heart rate data o the heart rate calculator.
- Calculate heart rate: The heart rate system calculates heart rate by counting the number of heartbeats in one minute.
- Heart rate display: The heart rate screen shows the user their heart rate on the smartwatch screen after receiving the data from the system.
- Average heart rate calculation: The heart rate calculator determines the average heart rate for a specific period of time.
- Store average heart rate: For future use, the average heart rate is stored in the database.
- Heart rate exceeds threshold: The watch notifies the user by sending out an alert if their heart rate goes above a set threshold.
- User alert: The heart rate alert generates a warning that it be displayed on the screen.
- Remove alert: The alert will be removed from the screen when the user chooses to cancel it or when the alert time limit ends.

2.2.3 Activity Diagram

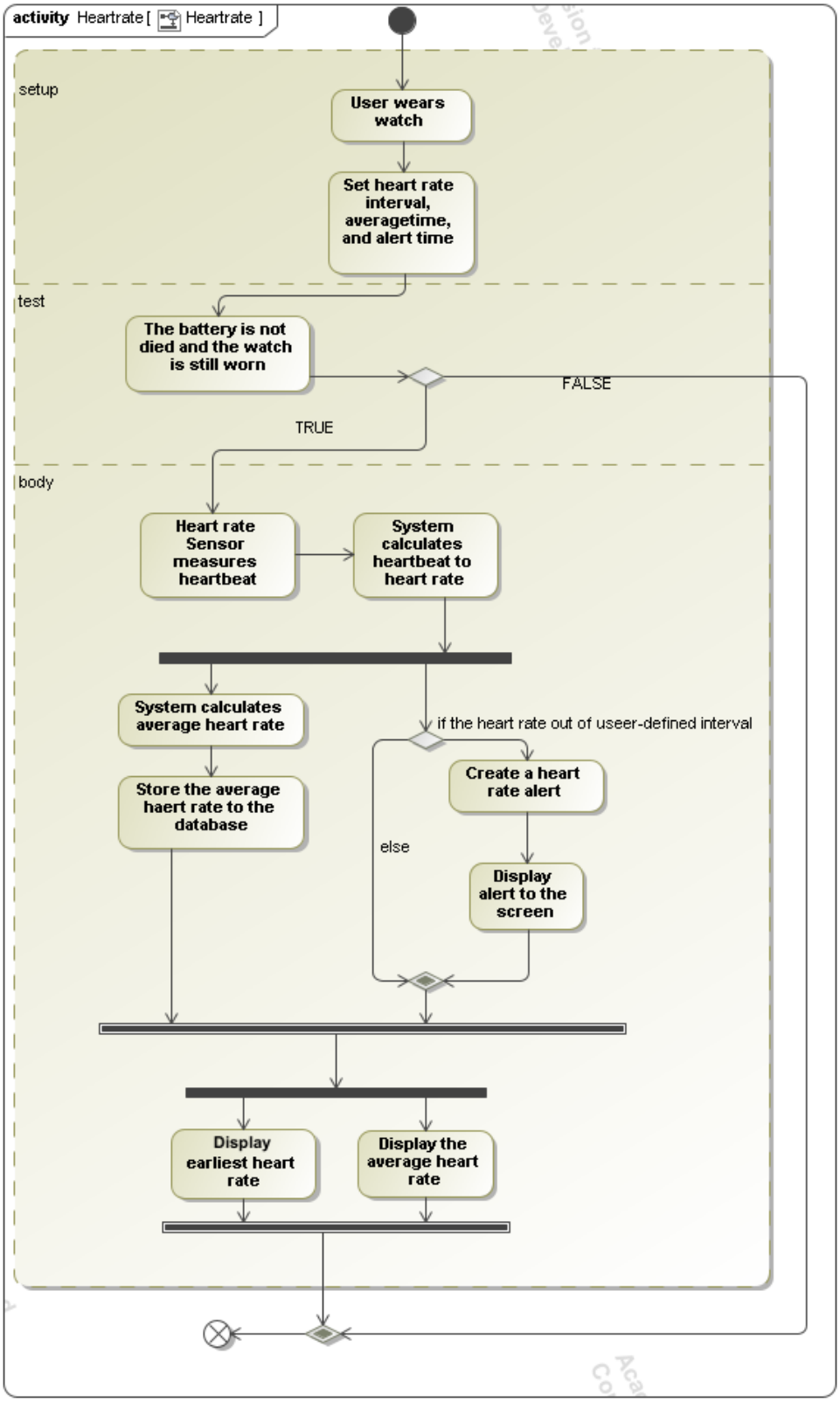

FIGURE 2.3: Heart rate monitoring activity diagram

The activity diagram shows how a smartwatch user's heart rate is monitored, and heart rate alerts are set off. Wearing the smartwatch and activating the heart rate monitoring app is the first step.

1. **Wearing a smartwatch:** The user opens the heart rate monitoring app after wearing the smartwatch.

2. **Configure heart rate parameters:** The user can set the alert time, average time, time interval, and heart rate interval. The user-specified safe heart rate is determined by the heart rate interval. The average interval denotes the duration during which the wearer's heart rate is averaged by the watch in order to determine the average heart rate. When the alert time reaches zero, it will automatically switch off, as defined by the alert time.

   Loop condition: The loop only starts if the user is still wearing the watch and the battery is not dead is TRUE after all system parameters have been set up. If the condition is FALSE, the activity flow will move on to the endpoint.

3. **Heart rate sensor checks heart rate:** The user's heart rate is continuously monitored by the heart rate sensor built into the smartwatch.

4. **calculates heart rate:** The heart rate calculator calculates the heart rate by counting the number of heartbeats in one minute.

   Start Parallel action:

5. **Calculates average heart rate:** During the specified time period, the heart rate calculator determines the average heart rate.

6. **Heart rate to threshold comparison:** The application tests the heart rate for heart rate thresholds that the user has set. An alert is generated to inform the user of an unusual heart rate pattern if the heart rate surpasses the threshold.

7. **Heart rate alert display:** The user receives an alert from the app on their smartwatch.

   End Parallel action

   Start Parallel action:

8. **Shows heart rate:** The smartwatch's screen shows the user's heart rate, which it got from the heart rate calculator.

9. **Shows average heart rate:** The smartwatch's screen shows the user the average heart rate, which it received via the heart rate calculator.

   End Parallel action

2.2.4 Class Diagram

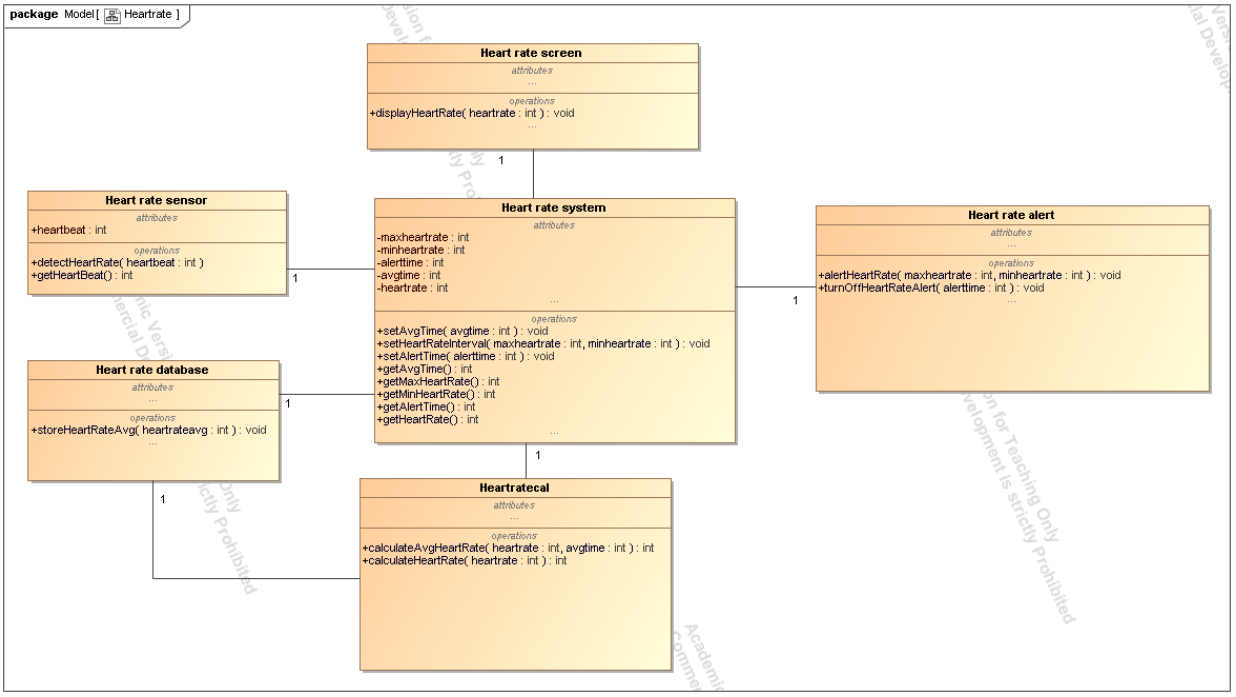

FIGURE 2.4: Heart rate monitoring class diagram

Heart rate sensor class

Attributes:

- `heartbeat`: an integer representing the number of heartbeats.

Operations:

- `detectHeartBeat(heartbeat: int)`: detects the heartbeat attribute.
- `getHeartBeat() : int`: retrieves the heartbeat value.

Heart rate system class

Attributes:

- `maxheartrate`: an integer representing the maximum value of heart rate in the user-defined interval.
- `minheartrate`: an integer representing the minimum value of heart rate in the user-defined interval.
- `alerttime`: an integer representing the countdown time (in second unit) for the automatic alert turn-off, set by the user.
- `heartrate`: an integer representing the number of heartbeats in a minute (calculated in the "Heartratecal" class from the heartbeat data).
- `avgtime`: an integer representing the period of time (in hour unit) for calculating the average heart rate, set by the user.

Operations:

- `setAvgTime(avgtime : int)`: sets the average time for calculating the average heartbeat.
- `getAvgTime() : int`: retrieves the average time set by the user.
- `setHeartRateInterval(maxheartrate : int, minheartrate : int)`: sets the maximum and minimum values safe for the user.
- `getMaxHeartRate() : int`: retrieves the maximum heart rate of the user-defined interval.
- `getMinHeartRate() : int`: retrieves the minimum heart rate of the user-defined interval.
- `setAlertTime(alerttime : int)`: sets the countdown time for the automatic alert turn-off.
- `getAlertTime() : int`: retrieves the countdown time set by the user.

Heartratecal class

Operations:

- `calculateAvgHeartRatel(heartrate : int, avgtime : int)`: calculates the average heart rate over the specified time.
- `calculateHeartRate(heartbeat : int)`: calculates the heart rate from the heartbeat data.

Heart rate alert class

Operations:

- `alertHeartRate(maxheartrate : int, minheartrate : int)`: alerts the user when the heart rate exceeds the user-defined interval.
- `turnOffHeartRateAlert(alerttime : int)`: turns off the alert automatically after the countdown.

Heart rate database class

Operations:

- `storeHeartRateAvg(heartrate : int)`: stores the heart rate and average heart rate.

Relationships:

- The `Heart rate sensor` class communicates with the `Heart rate system` class to send heart rate data.
- The `Heart rate system` class stores heart rate data in the `Heart rate database` class.
- The `Heart rate system` class also compares the current heart rate to the set alert thresholds and sends an alert to the user if the heart rate exceeds the threshold.

2.2.5 UI Prototype

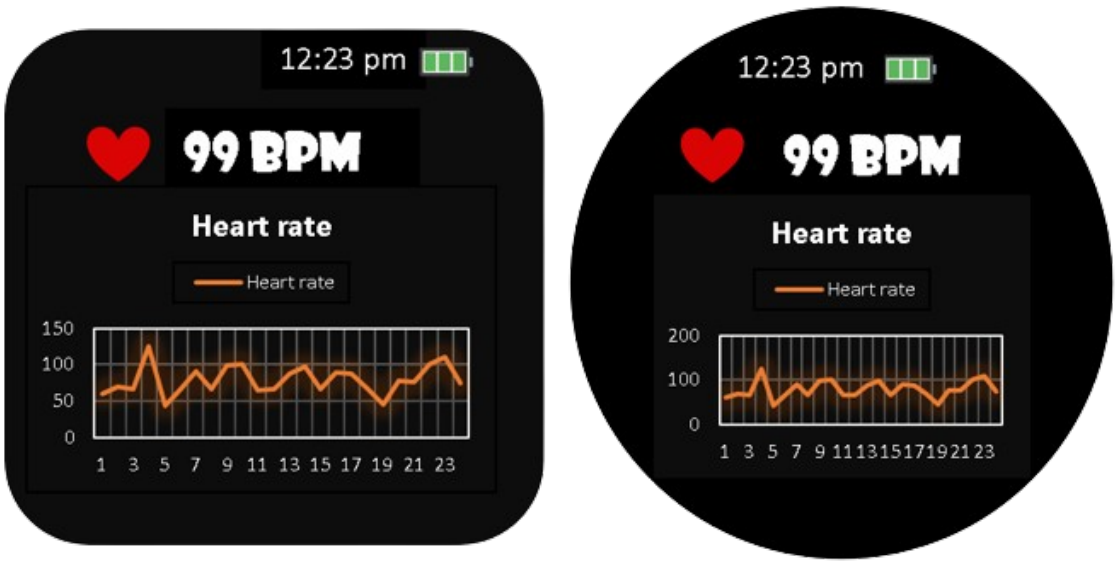

FIGURE 2.5: Heart rate monitoring UI prototype

Users can view and track their heart rate in real time with the graphical user interface (GUI) presented in the wearable heart rate monitoring prototype. With features including a circular heart rate graph, real-time heart rate readout, and history graph, the prototype has a sleek and contemporary appearance and provides users with extensive data regarding their heart rate patterns.

A clear numerical of the current heart rate is given by the real-time heart rate readout. To ensure that consumers can quickly obtain their heart rate data, this readout is placed next to the graph. The information is easy to read so it is simple to quickly understand changes in heart rate.

The history graph shows the history graph provides a visual representation of the user's heart rate over a specified time interval. Users can track patterns and trends in their heart rate over time by customizing the graph to display heart rate data over different durations. This function is especially helpful in recognizing changes in heart rate that might need getting medical assistance.

The UI prototype also shows the battery percentage and digital clock, giving consumers the necessary information to monitor the time and battery life.

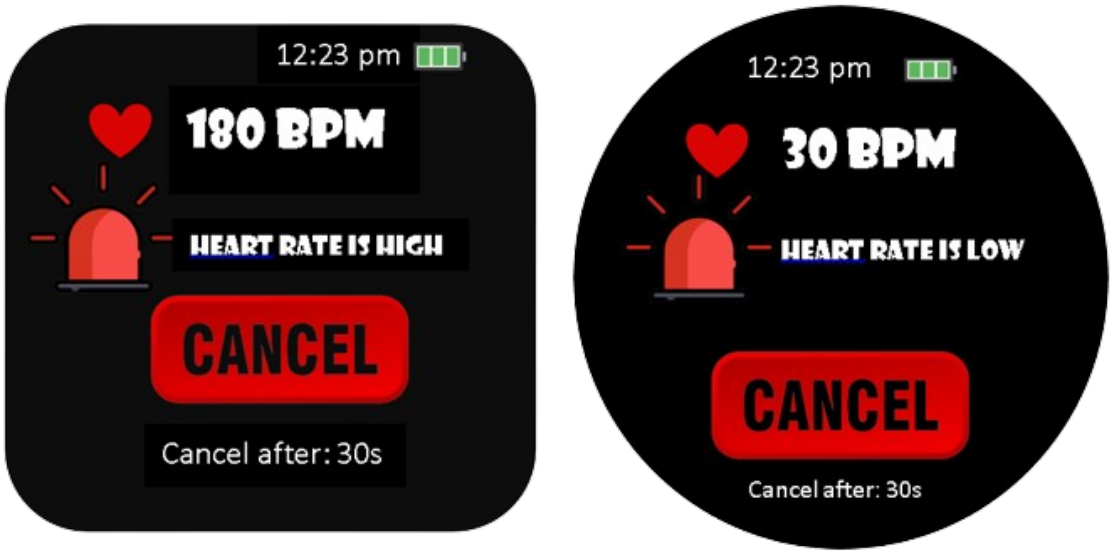

FIGURE 2.6: Heart rate alert UI prototype

A comprehensive and user-friendly interface is shown in the UI prototype for heart rate alerts on smartwatches, alerting users to problematic heart rate patterns. The prototype efficiently notifies users of potential health risks and offers recommendations on suitable measures through the use of visual components and clear wording.

The prototype's main feature is the heart rate alert message, which shows the user's predefined threshold exceeded and prominently displays the current heart rate. Large, easy-to-read typeface used to convey this message makes it possible for consumers to rapidly understand the warning and take appropriate action.

Specific visuals are added to the alert to increase its impact. For example, a red alarm icon is positioned next to the heart rate alert phrase to highlight how urgent the issue is. To further clearly convey the heart rate level, a notification indicating "HEART RATE IS HIGH" or "HEART RATE IS LOW" is incorporated.

Two separate buttons are included in the notification to help direct the user's response. With the first button, "CANCEL," the user can choose to ignore the alert and go on without doing anything else. This is helpful if there is a transient cause for the high heart rate, like stress or exercise. In addition, when the countdown hits zero, the heart rate alert will switch off automatically.

Chapter 3

Quick Payment

Quick payment function refers to streamlined features across different platforms for efficient transactions. This includes fast mobile payments through QR codes, quick online banking transfers, streamlined e-commerce checkouts, and rapid point-of-sale transactions. **Requirement Type:** Functional
For Whom: Customer
User Satisfaction: High
User Dissatisfaction: High
Description: The quick payment function is designed to offer customers a seamless and efficient payment experience. Users can swiftly complete transactions using various methods, such as QR codes, NFC technology, or streamlined online interfaces. The primary goal is to enhance user satisfaction by reducing the time and effort required for payments. The function should accommodate various payment sources and provide a straightforward, user-friendly interface.

3.0.1 Use Case Analysis

Name	Quick Payment
ID	03
Description	A quick payment function is designed to minimize the time and complexity typically associated with payments, providing users with a more efficient and user-friendly experience. This function is designed to minimize the time and complexity typically associated with payments, providing users with a more efficient and user-friendly experience.
Trigger	The customer decides to make a purchase and chooses the quick payment option to complete the transaction.
Preconditions	Sufficient funds or available credit are present in the chosen payment method and the chosen item or service is available for purchase.
Post-conditions	The customer is provided with confirmation of the payment, and both the purchased item or service is acknowledged as paid, with the corresponding deduction reflected in the customer's account.
Basic Flow	
Description	This is the process when a user wants to make a payment or transfer money to another bank account.
Actions	1. The user makes a payment 2. The system asks if the user wants to alter the payment method or not. 3. The system checks the balance 4. If the balance is sufficient, the system transfers money to the merchant and sends a detailed transaction email to the user. 5. If the balance is insufficient, the system will cancel the transaction.

3.1 UML diagrams

3.1.1 Use Case Diagram

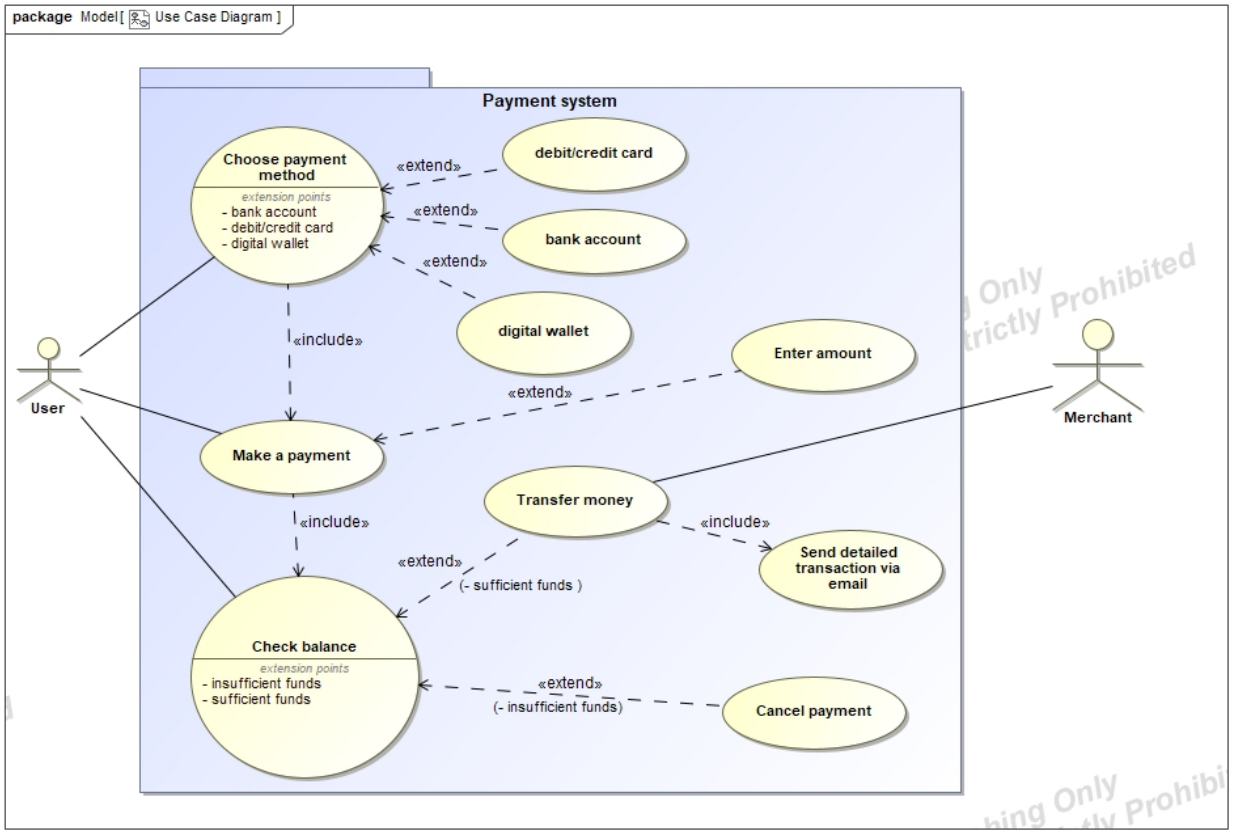

FIGURE 3.1: Quick Payment Use Case diagram

This use case diagram illustrates the interactions among the user, the smartwatch payment system, and the merchant. This document offers a comprehensive summary of the primary features associated with expediting payments through the utilization of a smartwatch.

Actors

- **User:** The person who owns and uses the smartwatch for making payments.
- **Merchant:** An external entity representing the business or service provider who accepts payments through the smartwatch.

Use Cases

- **Choose Payment Method:** The user chooses a desired payment method from the various alternatives on the smartwatch.
- **Extended Use Cases:**
  - Choose debit/credit card
  - Choose bank account
  - Choose digital wallet

- **Make a payment:** The user commences a payment transaction with the business by utilizing the chosen payment method.
- **Enter amount:** The user enters the desired payment amount using the smartwatch.
- **Check balance:** The user checks the balance, and the system verifies the balance of the user's associated payment account or card on the smartwatch.
- **Transfer money:** The user initiates a transfer of funds from their associated account to another user or receiver.
- **Send detailed transaction via email:** The user is sent a comprehensive transaction report via email for a particular payment or transfer.
- **Cancel payment:** The system terminates a payment transaction prior to its completion.

Associations:

- The user actor is associated with "Choose payment method", "Make a payment", "Check balance".
- The merchant is associated with "Transfer money".
- The "Choose Payment Method" use case is extended to include three options: "Choose Debit Card," "Choose Bank Account," and "Choose Digital Wallet."
- The "Make a payment" includes "Choose payment method" and "Check balance", and is extended to include "Enter amount".
- "Transfer money" and "Cancel payment" are two new use cases extended to the "Check balance" function.
- "Transfer money" includes "Send detailed transaction via email".

3.1.2 Class Diagram

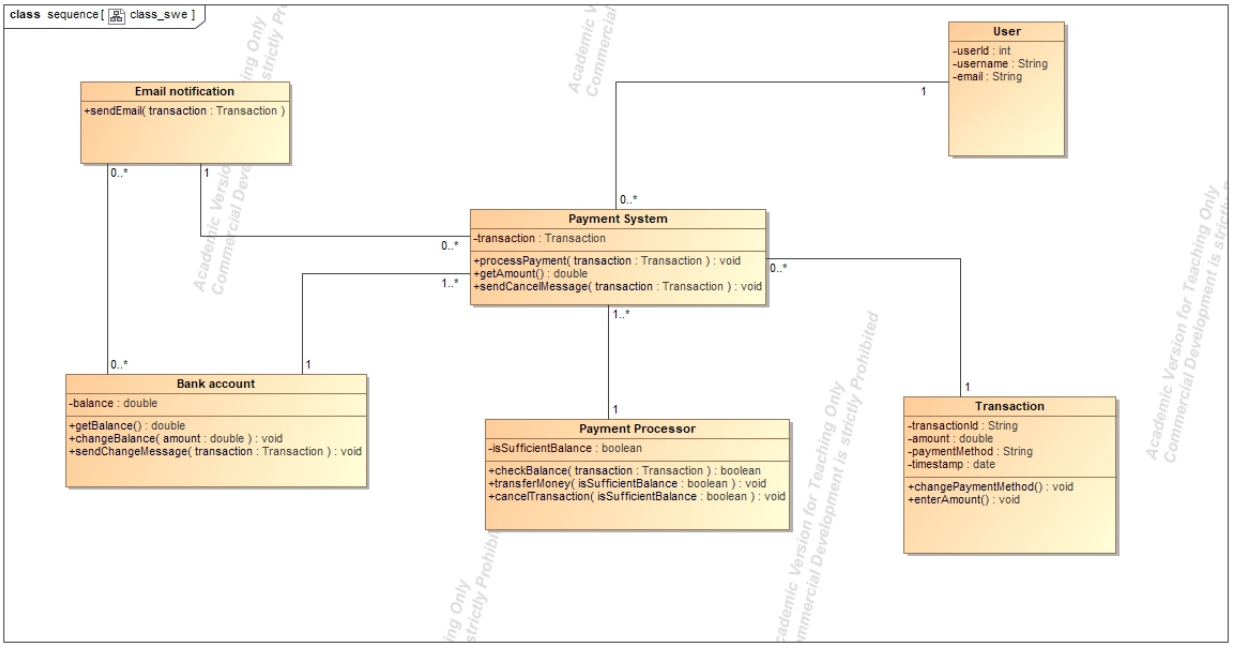

FIGURE 3.2: Quick Payment Class diagram

The class diagram illustrates the fundamental entities and connections within the 'Payment System'. Important classes are 'Email notification', 'Bank Account', 'Payment Processor', and 'User'. The 'Transaction' class includes attributes like 'amount' and 'payment-method', as well as methods such as 'changePaymentMethod'. The associations between classes illustrate the connections between the user and the payment system.

Transaction class

Attributes:

- `transactionId`: Represents a unique identifier for the transaction. Example: "TRX123456789"
- `amount`: Indicates the monetary value of the transaction. Example: 50.0
- `paymentMethod`: Describes the method used for the transaction (e.g., credit card, digital wallet). Example: "Credit Card"
- `timestamp`: Records the date and time when the transaction occurred. Example: "2024-01-30 15:30:00"

Operations:

- `changePaymentMethod()`: Allows the modification of the payment method associated with the transaction.
- `enterAmount()`: Updates the 'paymentMethod' attribute based on user input or system requirements.

Payment System class

Attributes:

- `transaction`: Represents a single transaction associated with the payment system. Example: A transaction with a specific transactionId, amount, paymentMethod, and timestamp.

Operations:

- `processPayment()`: Executes the necessary steps to process the payment, involving interactions with the payment processor, and deduction of funds from the linked bank account.
- `getAmount()`: Retrieves the total amount associated with the current transaction.
- `sendCancelMessage()`: Notifies relevant parties or systems about the cancellation of the specified transaction, providing necessary details.

Email notification class

Operation:

- `sendEmail()`: Send detailed transaction details to the user's email address.

Bank Account class

Attributes:

- `balance`: Represents the current balance in the bank account. Example: €733,30

Operations:

- `getBalance()`: Retrieves the current balance of the bank account.
- `changeBalance()`: Updates the balance of the bank account based on a given amount.
- `sendChangeMessage()`: Sends a message/notification related to a change in the bank account balance.

Payment Processor class

Attributes:

- `isSufficientBalance`: Represents whether there is sufficient balance to process a transaction. Example: true (if there is sufficient balance), false (if there is not sufficient balance)

Operations:

- `checkBalance()`: Checks if there is sufficient balance for the specified transaction.
- `transferMoney()`: Transfers money as part of processing a transaction.
- `cancelTransaction()`: Cancels a transaction if there is insufficient balance.

User class

Attributes:

- `usedId`: Represents a unique identifier for the user. Example: 12345
- `username`: Stores the user's chosen username. Example: "ga1nang"
- `email`: Stores the user's email address. Example: "nnthanhmx@gmail.com"

3.1.3 Sequence Diagram

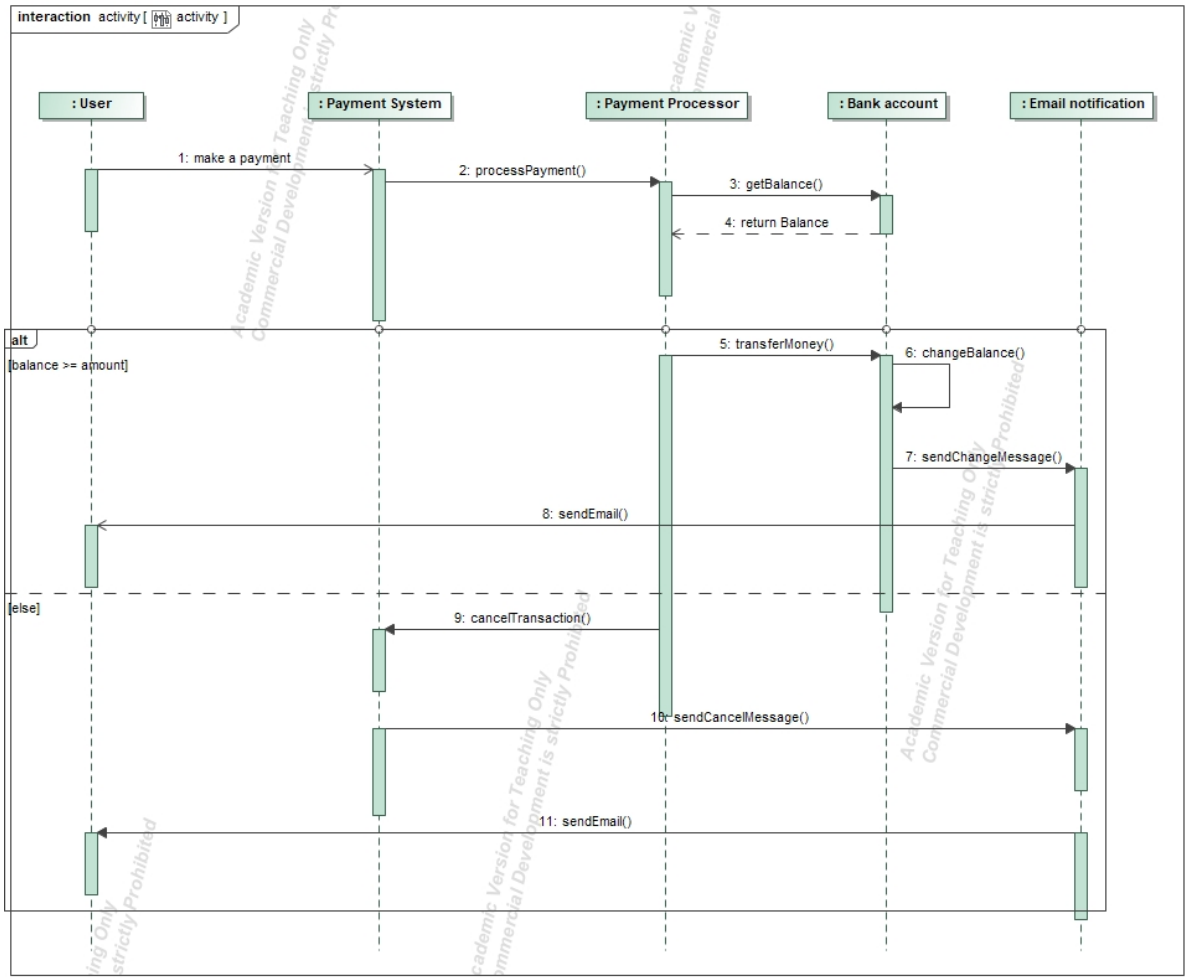

FIGURE 3.3: Quick Payment Sequence diagram

- The sequence diagram represents the flow of messages and interactions between different components during the quick payment process.
- Steps 2 to 4 involve interactions between the Payment System, Payment Processor, and Bank Account to verify the availability of funds.
- Depending on the balance, the sequence splits into two branches, one for successful payments and the other for canceled transactions.
- Email Notification and User are involved in communicating relevant information at different stages of the process.
- This sequence diagram provides a high-level overview of the interactions between the components involved in the quick payment process.

Step 1: Make a Payment

User initiates a payment through the Payment System.

Step 2: `processPayment()`

Payment System communicates with Payment Processor to initiate payment processing.

Step 3: `getBalance()`

Payment Processor queries the Bank Account for the current balance.

Step 4: Return Balance

Bank Account returns the balance to the Payment Processor.
If Balance > Amount:
Proceed with the following steps.

Step 5: `transferMoney()`

Transfer money within the Bank Account if there is sufficient balance.

Step 6: `changeBalance()`

Update the Bank Account balance based on the payment amount.

Step 7: `sendChangeMessage()`

Email Notification is triggered to inform about the change in the Bank Account balance.

Step 8: `sendEmail()`

User receives an email notification regarding the successful payment.
Else (Balance $\leq$ Amount):
Execute the following steps.

Step 9: `cancelTransaction()`

Payment System cancels the transaction due to insufficient balance.

Step 10: `sendCancelMessage()`

Email Notification sends a cancellation message.

Step 11: `sendEmail()`

User receives an email notification regarding the canceled transaction.

3.1.4 Activity Diagram

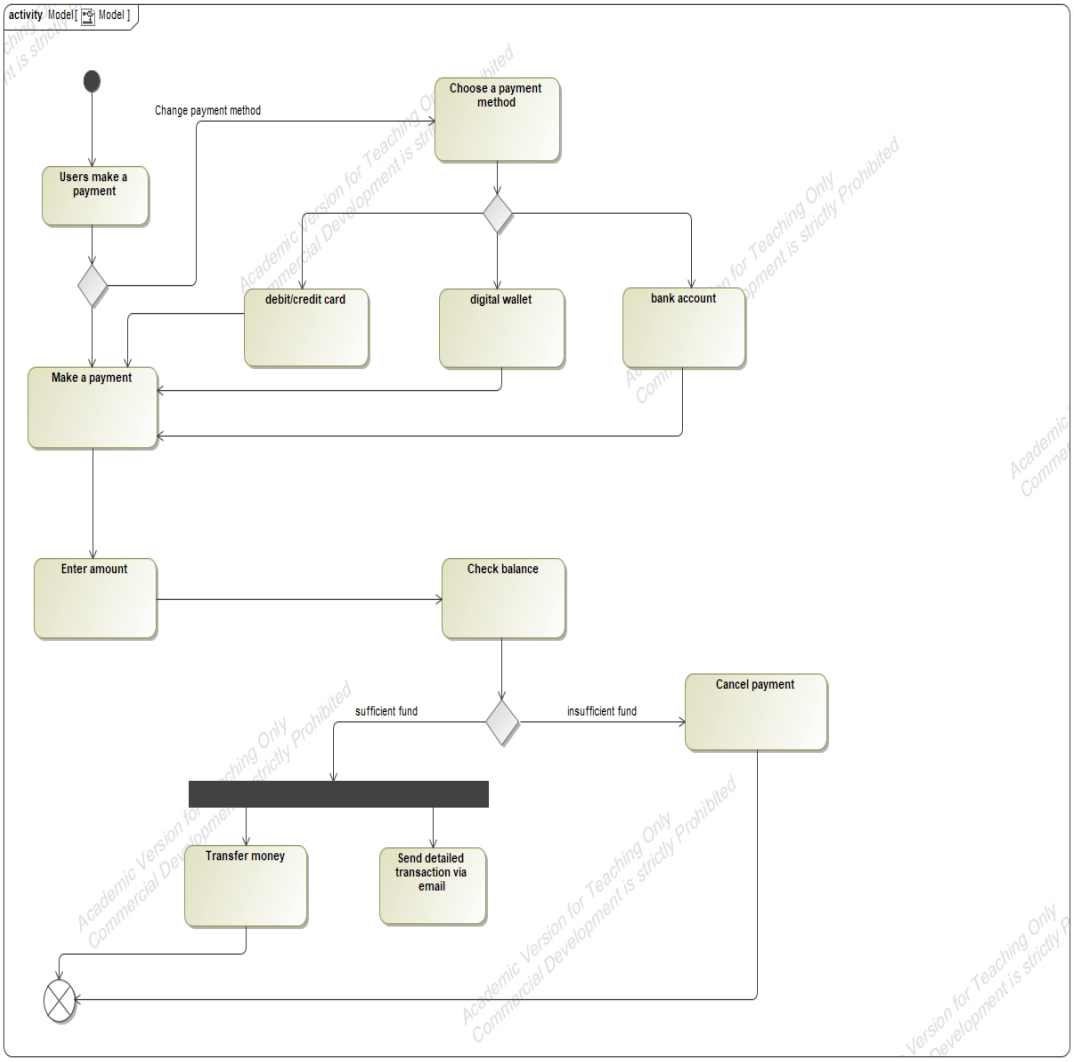

FIGURE 3.4: Quick Payment Activity diagram

The enhanced Quick Payment Activity Diagram depicts the dynamic process of a user initiating a payment. It incorporates various user decisions, system responses, and alternative flows to simulate a realistic and engaging payment experience. The diagram covers scenarios such as changing payment methods, user hesitations, and email delivery issues, providing a comprehensive view of the interaction between the user and the payment system.

- **Step 1: User Makes a Payment**
  - User initiates the payment process.

  If User Wants to Change Payment Method:
  - The process diverges based on user choice.
- **Step 2: Choose a Payment Method**
  - User selects the option to change the payment method.
- **Step 3: Choose Between Debit/Credit Card, Bank Account, Digital Wallet**
  - User selects the desired payment method.

  Alternative Flow:
  - If the user struggles to decide, the system prompts a suggestion based on recent transactions or preferences.

  If User Doesn't Want to Change Payment Method:
  - The process continues for making a payment with the existing method.
- **Step 4: Make a Payment**
  - Proceed to make a payment.

  Alternative Flow:
  - If the user hesitates, the system provides a quick tutorial or assistance.
- **Step 5: Enter Amount**
  - User enters the payment amount.
- **Step 6: Check Balance**
  - System checks the balance in the chosen payment method.

  Alternative Flow:
  - If the balance is slightly insufficient, the system prompts the user to use a different payment method or reduce the amount.

  If Sufficient Balance:
  - Execute simultaneous activities.
- **Step 7: Transfer Money**
  - Transfer money from the user's account.

- **Step 8: Send Detailed Transaction via Email**
  - Send a detailed transaction summary via email.

  Exceptional Flow:

  - If the email fails to send, the system retries or provides alternative notification options.

  If Insufficient Balance:

  - Cancel the transaction.
- **Step 9: Cancel Transaction**
  - The transaction is canceled.

3.1.5 UI Prototype

Home UI

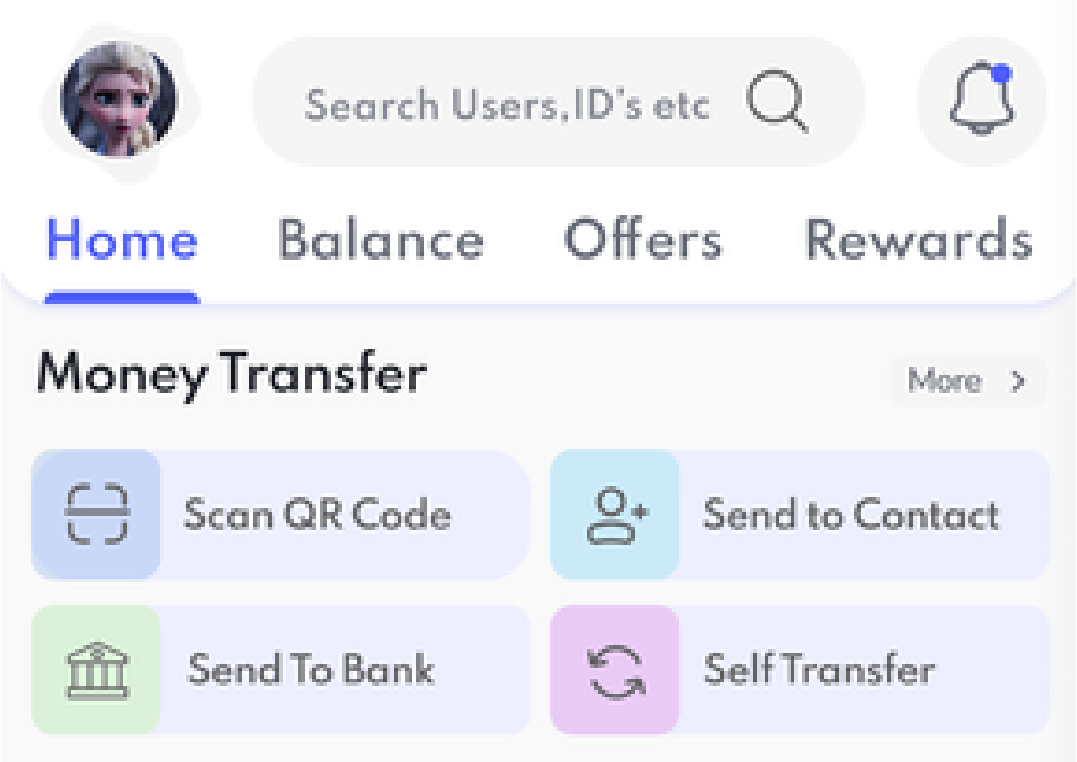

FIGURE 3.5: Quick Payment UI prototype

The Quick Payment User Interface offers a streamlined and user-friendly experience for efficient transactions. The Home UI, featuring key information like balance and rewards, provides quick access to essential functions such as scanning QR codes, sending money to banks or contacts, and facilitating self-transfers. With clear instructions and straightforward design on each dedicated page, users can navigate effortlessly, ensuring a smooth and enjoyable payment process.

User UI

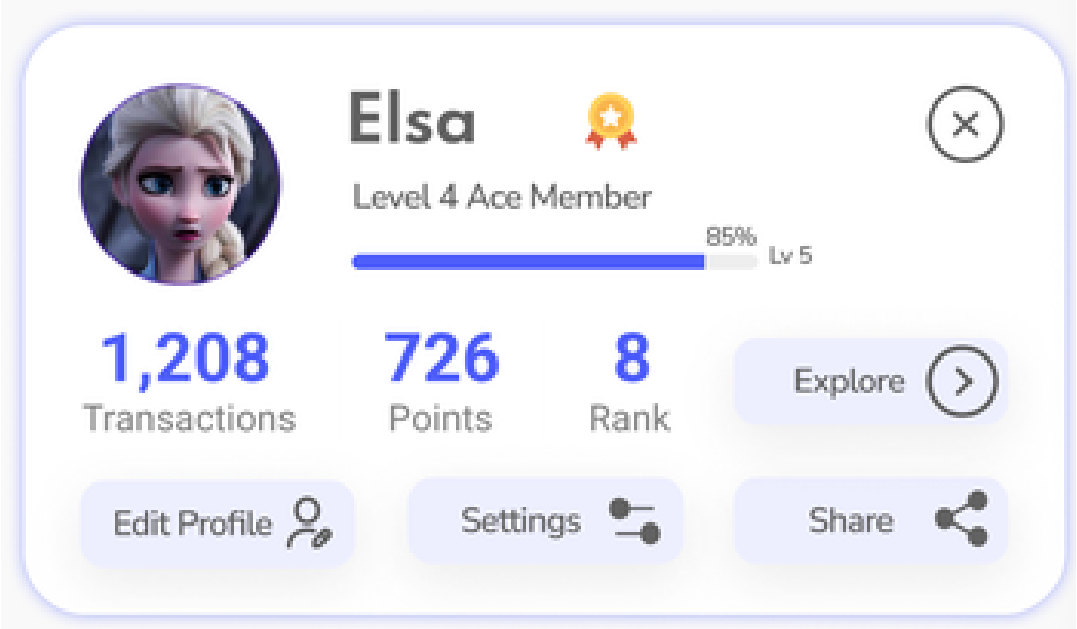

FIGURE 3.6: Quick Payment User UI

Username and User Avatar:

- Clearly displays the user's chosen username and a personalized avatar.

Transaction Overview:

- Provides a snapshot of the user's transaction history, offering insights into their payment activities.

Points and Rank:

Showcases the user's accumulated points and current rank, reflecting their engagement and status within the platform.

Functional Buttons:

- **Edit Profile:** Enables users to update and customize their profile information, ensuring accuracy.
- **Settings:** Grants access to various customization options, allowing users to tailor the app to their preferences.
- **Share:** Facilitates the sharing of user achievements or the app itself through different channels.

- This Profile UI is designed to not only present essential account information but also empower users with control over their profile settings and the ability to share their experiences with others.

Chapter 4

UV Exposure Monitoring

UV Exposure Monitoring is a function that measures everyday UV levels and displays the level on screen throughout the day. In case the UV level is so high it breaks the threshold, there will be an alert for the user including navigation to the nearest proofed structure and recommendations for skin and health protection against the UV.

4.1 Requirement Analysis

4.1.1 Snow cards

Requirements Type: Functional
For Whom: customer
User Satisfaction: High
User Dissatisfaction: High
Description: It will measure, forecast, and display the peak UV level for the entire day at the start of the day. The device will buzz to inform the user if the UV level crosses the threshold. If the user is outside, it will determine where they are and direct them to the closest building, structure with a ceiling proof, or shade. Additionally, they will record the user's location, UV level, and UV exposure time in a database when they are not inside. Through database analysis, they will identify the user's regular exposure patterns and provide advice and information on protecting their skin and health from UV radiation.

4.1.2 Use Case Analysis

Name	UV Exposure monitoring
ID	04
Description	It will measure, forecast, and display the peak UV level for the entire day at the start of the day. The device will buzz to inform the user if the UV level crosses the threshold. If the user is outside, it will determine where they are and direct them to the closest building, structure with a ceiling proof, or shade. Additionally, they will record the user's location, UV level, and UV exposure time in a database when they are not inside. Through database analysis, they will identify the user's regular exposure patterns and provide advice and information on protecting their skin and health from UV radiation.
Trigger	For monitoring, it turns on when the watch is turned on. For alert function, is activated when the UV level passes the threshold. For the navigation function, it is turned on when the user is outside on a harsh day.
Pre-conditions	The smartwatch is wore and turned on.
Post-conditions	For the alert function, when the user turns it off or has read the information. For the navigation function, when the user turns it off or when he or she is already located under shades.
Basic Flow	
Description	This scenario describes the situation when it is a harsh day which there is a period when the UV level passes the threshold and the user is also outside.
Actions	1. The sensor measures UV level of the day. 2. It checks the UV level. If the influence is too strong, it alerts the user. 3. It locates users, and navigate the user to the closest shade. 4. Store the data of each of user's exposure sessions of the day. 5. Analyzing the data. 6. Give out recommendations for user.

4.2 UML Diagrams

4.2.1 Use Case Diagram

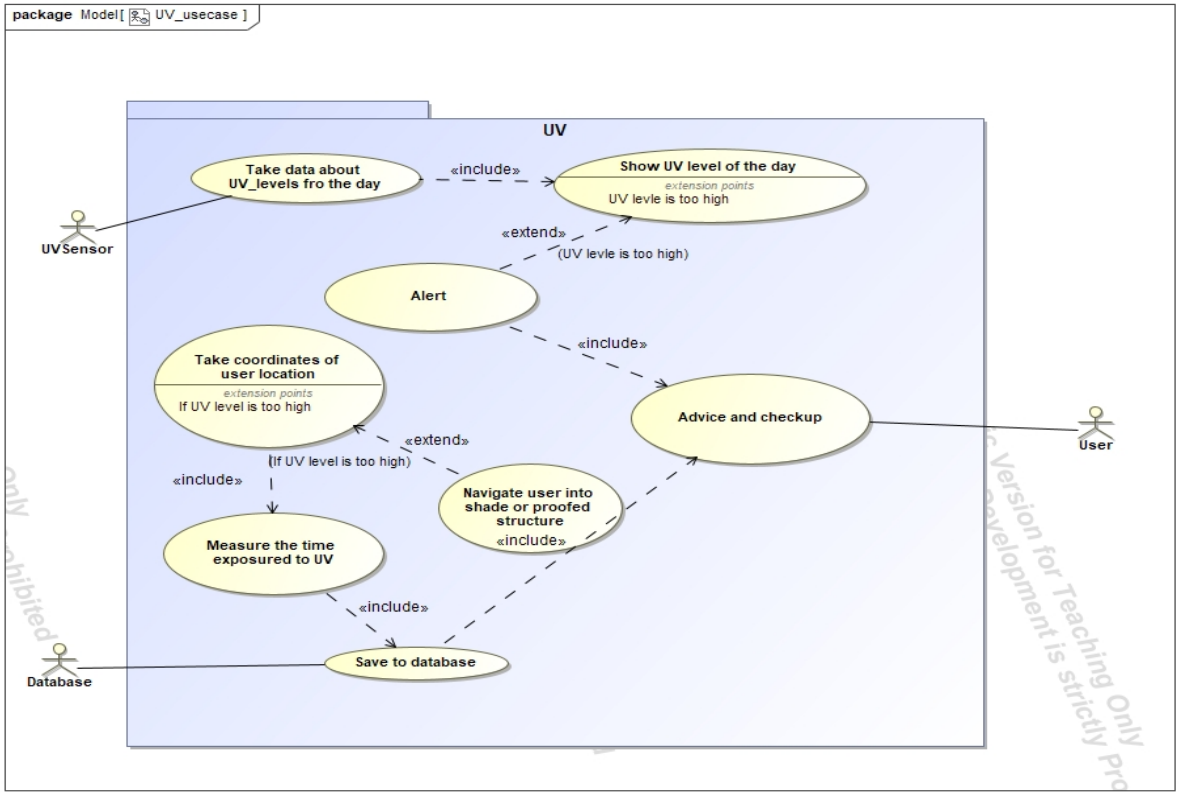

FIGURE 4.1: UV Exposure Monitoring use case diagram

In this diagram, there are 3 actors:

- Sensor
- User
- Database

Overall function

The sensor will measure the level of UV radiation of the day and show it to the screen. At the same time, the system will take user location coordinates. If the user is outside, the system will store the exposure session in the database. Furthermore, in case the radiation is too harsh that it passes a threshold, it will alert the user and on the condition that the user is outside, an extended event will occur which the system navigates the user to the nearest shade. After analyzing the sessions inside the storage, the system will give recommendations and advice for skincare and health protection to the user.

Detailed breakdown

Thorough analysis of this use case diagram: The main use cases of this function will be:

- The system will take data about the radiation level of the day and display it for the user.
- It will also determine user's location from time to time for other operations.
- Store each of user's exposure session into the database.

Extension points and relationships in this use case diagram:

- If the UV level is too high that it passes a certain number, it will alert the user.
- If the user is outside, there are two use cases will be taken into consideration:
  - If the user is outside, it will call a timer to measure the duration of that period.
  - If the user is outside under harsh radiation, a navigation system will be activated and will guide the user into shade.

Include relationship:

- Save to database: This use case will always store the time, UV level and the location everytime the user is under direct influence of UV.
- Advice and checkup: the system will analyze the database to give advice for health and skincare or give a check up after a duration under high level of radiation.

Note that the advice and checkup use case will be developed as a separate function so it will not be included in other diagrams.

4.2.2 Class Diagram

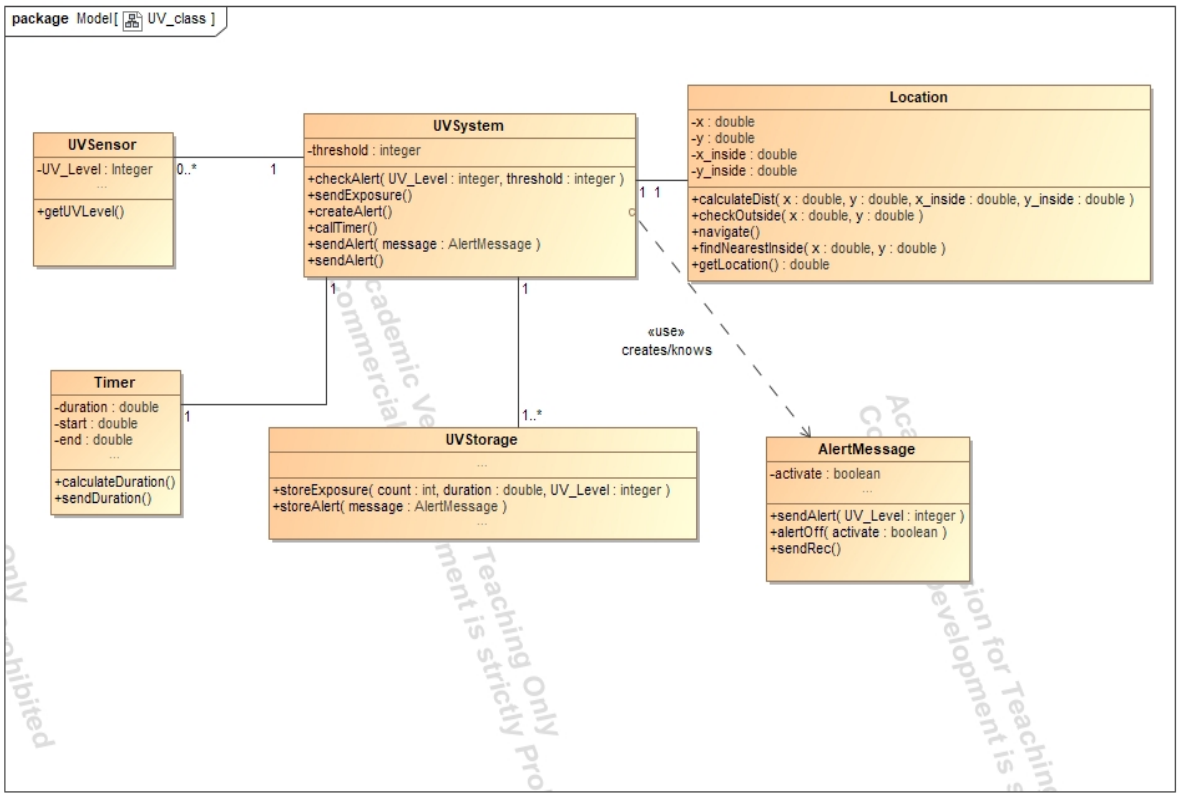

FIGURE 4.2: UV Exposure Monitoring class diagram

In the class diagram, there are 4 main classes:

- Sensor: this class represents the sensor that measures the UV level. It has an attribute for the UV level and a method for getting the UV level.
- System: this class is the system that manages activities of the function. It check the condition for an alert, create the alert and timer. Also, it sends the exposure session to the storage.
- Location: it will take the location of the user and the nearest shade in case of emergency to navigate the user out of the influence of UV radiation.
- Storage: this class is responsible for storing the time when user is under influence of UV light as well as the strength of the radiation.

There are 2 other classes that are under influenced of the others:

- AlertMessage: This class is created when the checkAlert() method return true. Which means the UV level passed the threshold. It acts as a way to notify the user about the harsh sunlight.
- Timer: This class is responsible for measuring the duration when the user is exposed to UV.

There are some other vital functions that play a big role for the software:

- checkOutside(): It will check whether the user is outside.
- findNearestLocation(): In case the user is under harsh influence of UV, it will take user location and find the nearest place to navigate.
- sendExposure(): this function is responsible for sending sessions to the database.

4.2.3 Sequence Diagram

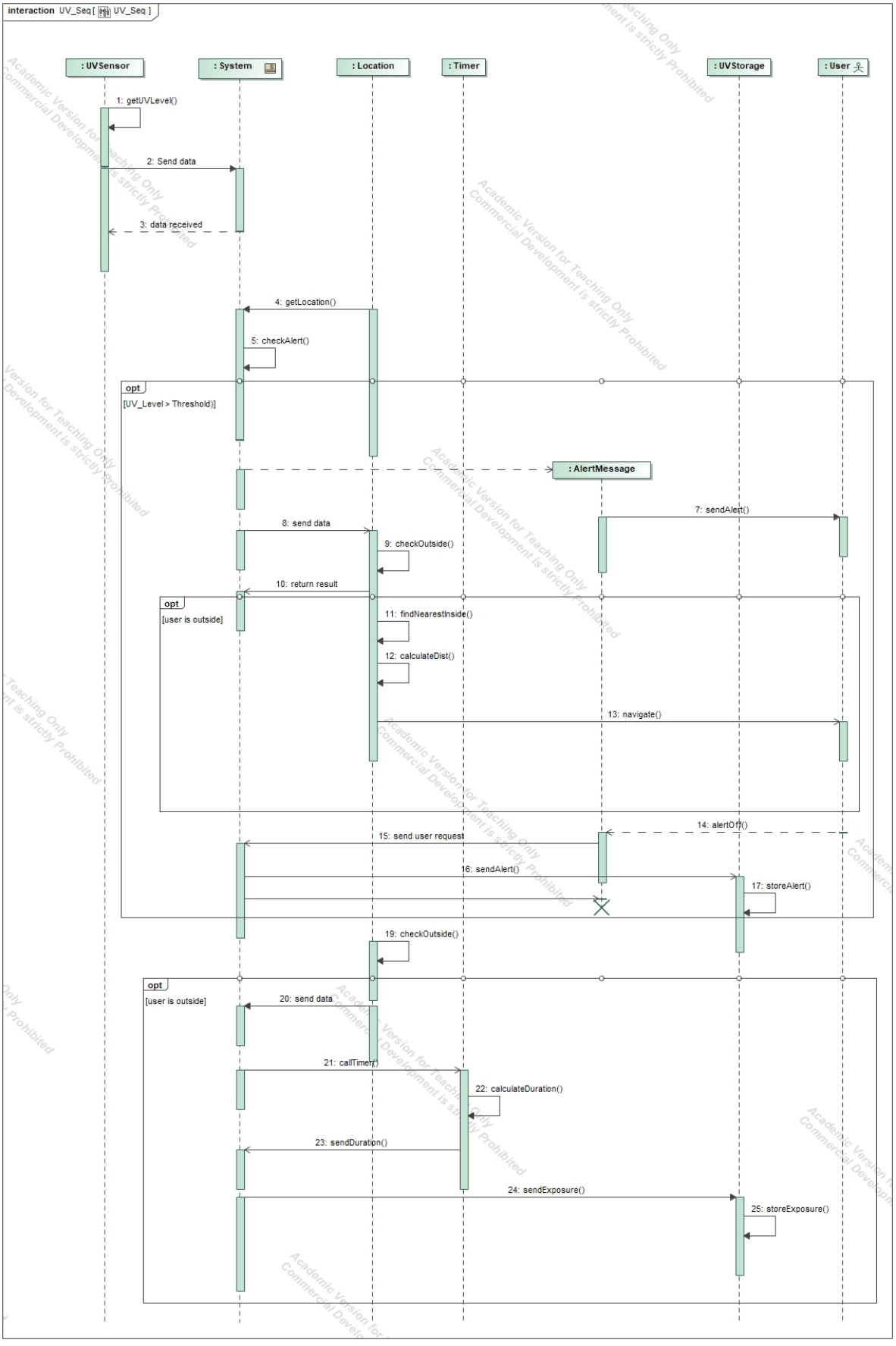

FIGURE 4.3: UV Exposure Monitoring sequence diagram

This sequence diagram illustrates the interaction between the user, the sensor, the system, and the location to monitor the user's behavior while the UV is still having an effect. First and foremost, the system fetches data from the sensor and displays it to the user while taking the user's coordinates. Then, the system will check if the UV is too strong or not. If it is too strong, it will alert the user. In case the user is outside, it will find the nearest place with proof to navigate the user. The user can turn off the alert message at will. The system then will store each alert in the database. In case the user is outside (including when the radiation is too harsh), the system will call a timer to time the period in which the user is under direct influence of UV. After that, the system will store the time, including the location and UV level in the database.

Detailed Breakdown of Lifetimes and Events

Lifetimes

There are 7 lifetimes in the diagram:

- Sensor: The sensor records the UV level of the day.
- System: The system controls most operations of the functions, transmits data to other lifetimes and also initiates the existence of the AlertMessage.
- Location: The location monitors the user's coordinates, clarifies if user is outdoor or not and indirectly activates the Timer. Also, in extreme circumstances, it is also responsible for navigating the user.
- Timer: The timer measures the duration which the user is under direct effect of UV radiation.
- AlertMessage: The alert message is created when the UV level is too high and it is use for only alarming.
- Storage: The storage keeps records of time, date and place of each of user exposures to the UV.
- User: The user represents the real user who uses the function.

Events

The flow of actions of this sequence diagram will be:

1. The UV sensor measure the strength of the radiation and send to system.
2. The location of the user is taken and sent to the system.
3. The system check the level.
4. If the level is too high, it will create an AlertMessage to the user and take the user's coordinates to navigate the user to another location with less influence of the radiation.
5. A record of the alert will be sent to the databse. AlertMessage is terminated.
6. If user is outside, a timer is called to measure the duration the user is exposed to the UV.
7. Store the session to storage.

4.2.4 Activity Diagram

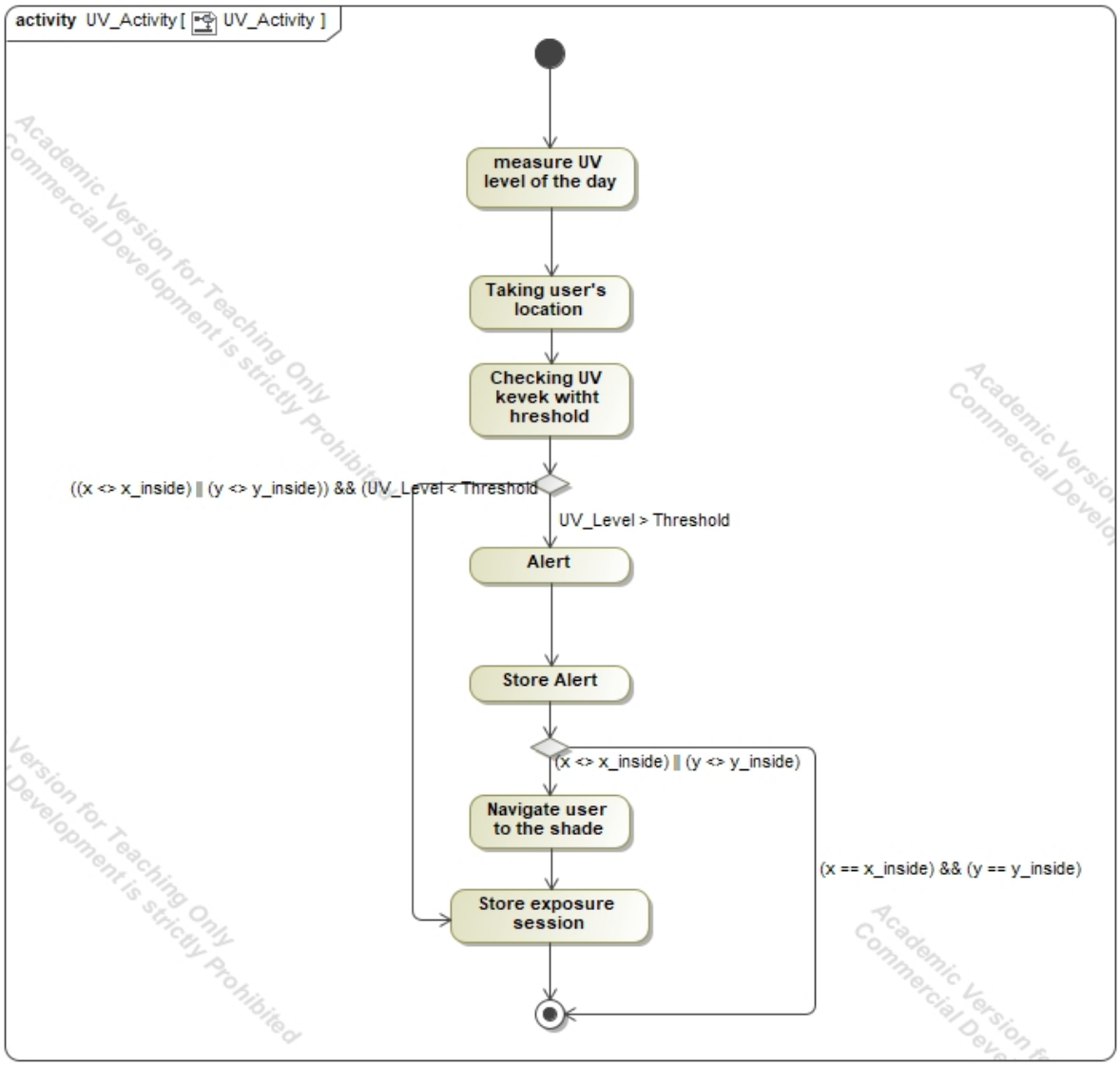

FIGURE 4.4: UV Exposure Monitoring activity diagram

The activity diagram displays the order of the actions within the UV exposure monitoring function. The sequence consists of measuring UV level, taking location, and checking conditions for alerting and storing exposure sessions in the database for further analysis.

The flow of the actions will be:

1. **Measure UV level of the day:** The process starts with the user measuring the UV levels in the sun.
2. **Taking user's location:** The user's coordinates will be fetched and sent to the system.
3. **Checking UV level with threshold**: The system will check if the UV level passes the threshold or not,

   Decision selection: if UV level passes threshold
4. **Alert:** An alert message is sent to the user to inform about the harsh radiation.

5. **Store Alert:** The alert will be stored.

   Decision selection: if user is outside:

6. **Navigate user to the shades:** The system will guide the user to a building or a shade.

   If the user is inside during high level of UV, it skips to the end.

7. **Store exposure session:** Time, UV level and location relate to the user while the user is outside will be stored.

   If user is outside on a normal day (decision on step 3) it will skip alert and navigate activities to store session

4.2.5 UI Prototype

Default UI

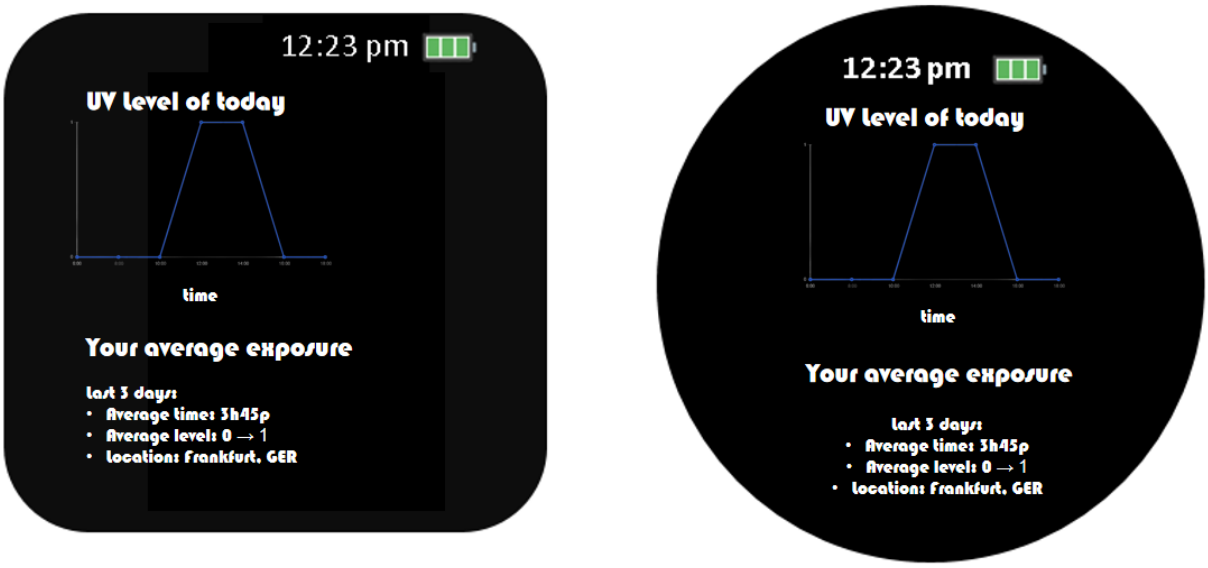

FIGURE 4.5: UV Exposure Monitoring default UI

The default UI for the smartwatch UV monitoring function provides the user with a detailed set of features in order to keep him or her up to date with the current radiation condition. The set consists of

- **Display for UV level of the day:** This components represents as a graph which there are time intervals and the UV levels according to the time. Its values will change if the user moves to another location which is not in the same time zone
- **Average Exposure:** This will show an "average" session of user when exposed to the UV in the last 3 days. The attributes in an average period includes:
  - **Average time:** It is the time that the user would usually spend under direct influence of UV lights for a day
  - **Average level:** It is the average UV index that the user stand under during the period.
  - **Location:** It is the location that the records occurred the most in last 3 day.

Storage UI

FIGURE 4.6: UV Exposure Monitoring storage UI

The storage UI of UV exposure monitoring applications provides the user with an archive that stores each session of user exposure to direct UV in the form of a table. The attributes of the table:

- **Date:** The date which the event occurred on.
- **Time:** The begin and end time of the record of the exposure period.
- **Location:** The place where the user is exposed to the radiation.
- **Level:** The UV index at that time.

There is also a sorting function for the user to find out the longest time they exposed themselves to the radiation or the highest UV index they faced.

Alert UI

FIGURE 4.7: UV Exposure Monitoring alert UI

The Alert UI of UV Exposure Monitoring application provides user a method to notify the user when the UV get high and can be bad for the user health. It consists of:

- A red alert as the form of text follows with an icon of the UV radiation.
- the peak level that is harmful to the user and also the time the UV index will go to that level.

In addition to that, it will also buzzing or even ringing if the condition gets extreme. This operation can not be demonstrated in this report.

Chapter 5

Step Count

The majority of people are not aware that they exercise regularly, so step counting was created to show them that they do and to encourage them to continue. It is a function that is specifically designed to count the user's steps as well as measure the distance the user has traveled and the number of calories they have burned.

5.1 Requirement Analysis

5.1.1 Snow cards

Requirements Type: Functional
For Whom: customer
User Satisfaction: High
User Dissatisfaction: High
Description: It measures the number of steps of the user from start to end and display duration, distance and calories burnt to the screen. In addition, there is an encouragement, when user surpasses their last session, it will encourage the user for exercising well as motivate them to continue. At the end, the function will give them the recommendation for a better health status and store the session to database.

5.1.2 Use Case Analysis

Name	Step Count
ID	05
Description	It measures the number of steps of the user from start to end and display duration, distance and calories burnt to the screen. In addition, there is an encouragement, when user surpasses their last session, it will encourage the user for exercising well as motivate them to continue. At the end, the function will give them the recommendation for a better health status and store the session to database.
Trigger	When user start to run or walk and start the timer.
Pre-conditions	The watch must be wore by user, has enough battery and be turned on.
Post-conditions	The user finish their exercise activities and then end the timer.
Basic Flow	-
Description	This scenario describes the situation in a normal sessions
Actions	1. It measures the number of steps. 2. It converts the measurements into distance the user travelled and amount of calories was burnt. 3. It checks if the measurements have surpassed the last session in the database. If it passed, it will give the user an encouragement 4. It then gives the users advice based on the measurements 5. It store the measurements into the database.

5.2 UML diagrams

5.2.1 Use Case Diagram

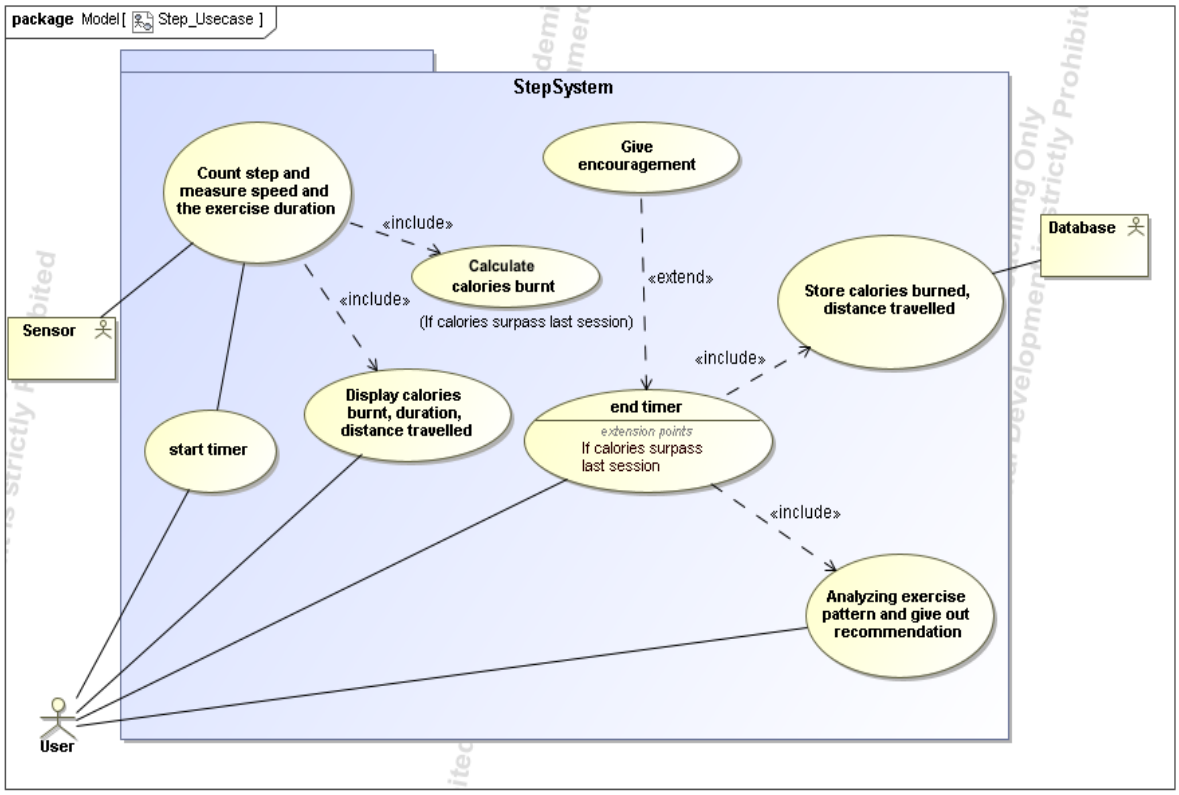

FIGURE 5.1: Step count use case diagram

In this diagram, there are 3 actors:

- Sensor: The sensor is the device that will enable the detection of a step and therefore step counting
- User: The user is the main actor who will interact with the smartwatch to exercise and use the step count.
- Database: The database store every session that the user have gone through

The main use cases will be:

- Count step and measure speed and distance travelled: This use case is the most essential because it will be the information used to calculate every other data available.
- End timer: This use case will end the session and will lead to various other use case:
  - Store calories burnt, distance travelled: This use case will record the session with the following measurements from the StepSystem.
  - Analyzing exercise patterns and give out recommendation: This use case will based on the calories burnt and distance travelled to give you advice on what users should do on their next session.

There are also extension points and extends:

- If calories burnt by users in this section surpass the last session.

Include relationships:

- Calculate calories burnt: The StepSystem will always calculate calories.
- Display calories burnt, duration and distance travelled: The StepSystem will always display to the user.

5.2.2 Class Diagram

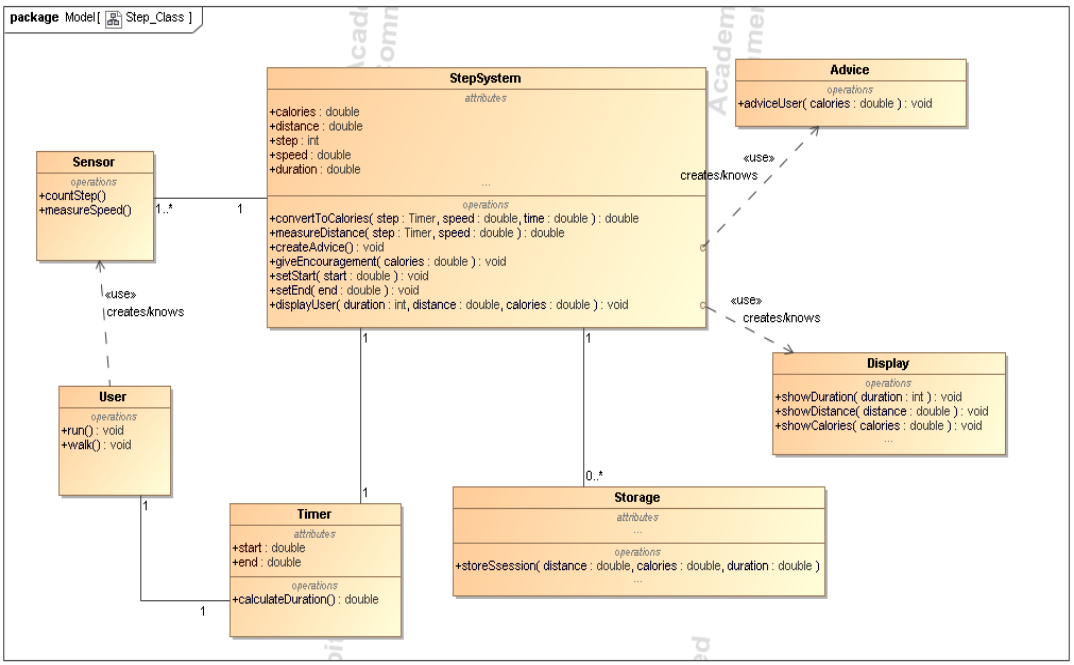

FIGURE 5.2: Step count class diagram

This diagram has seven classes:

- User: This class represents the user.
- Sensor: This class is accountable for the sensor that counts the step and measures the speed.
- Timer: This class is the timer, it will measure the time for each session.
- StepSystem: This class is responsible for convert the steps, speed, and time into calories and distance. It also sends and takes data from database while giving advice and encouragement.
- Storage: This class represents the database which stores the user past sessions.
- Advice: This is a class which is created by the system to give user advice.
- Display: This class is the display to show data.

In this class, there are some essential functions that plays an important role in the operation of the system:

- storeSession(): This operation allow the system to store the session of the user to the database.
- adviceUser(): This operation will advice the user base on the measurements done by the system.
- convertToCalories(), measureDistance(), calculateDuration(): These operation will calculate every measurements the StepSystem need to carry out its function.

5.2.3 Sequence Diagram

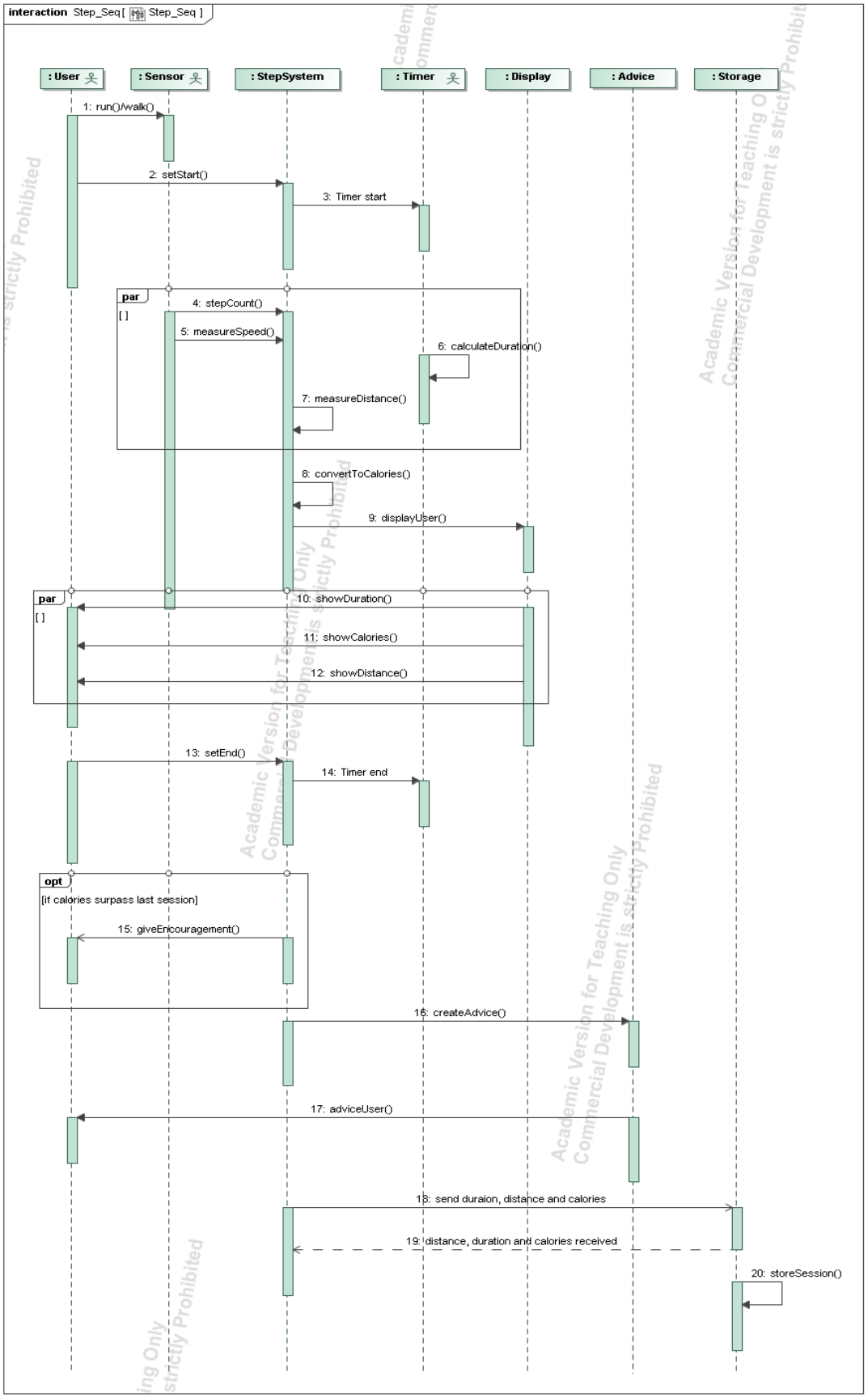

FIGURE 5.3: Step count sequence diagram

There are 7 lifelines in the diagram:

- User: This is the lifeline representing the user who will wear the smartwatch and interact with many other lifelines.
- Timer: This is the lifeline representing the timer what will calculate the duration of the session.
- Sensor: This lifeline represent the sensor that will be calculating step and speed of the user.
- StepSystem: This lifeline will calculate all kinds of measurements for other lifeline to use.
- Advice: This lifeline give user advice upon ending a session.
- Storage: The storage lifeline will store the user measurement.
- Display: This lifeline will display the necessary information to user.

The flow of activities will be:

1. The user runs or walks while wearing the watch and starts the timer.
2. The sensor counts the step, and measures the speed while the timer calculates the duration of the session and the StepSystem will measure distance traveled and calories.
3. The display will show duration, calories burnt, and the distance traveled to the user.
4. The user finishes and ends the timer
5. If the calories surpass the past session then the system will give the user encouragement.
6. The system will always give advice to the user.
7. Finally, the system will send data to storage and store the session in the database.

5.2.4 Activity Diagram

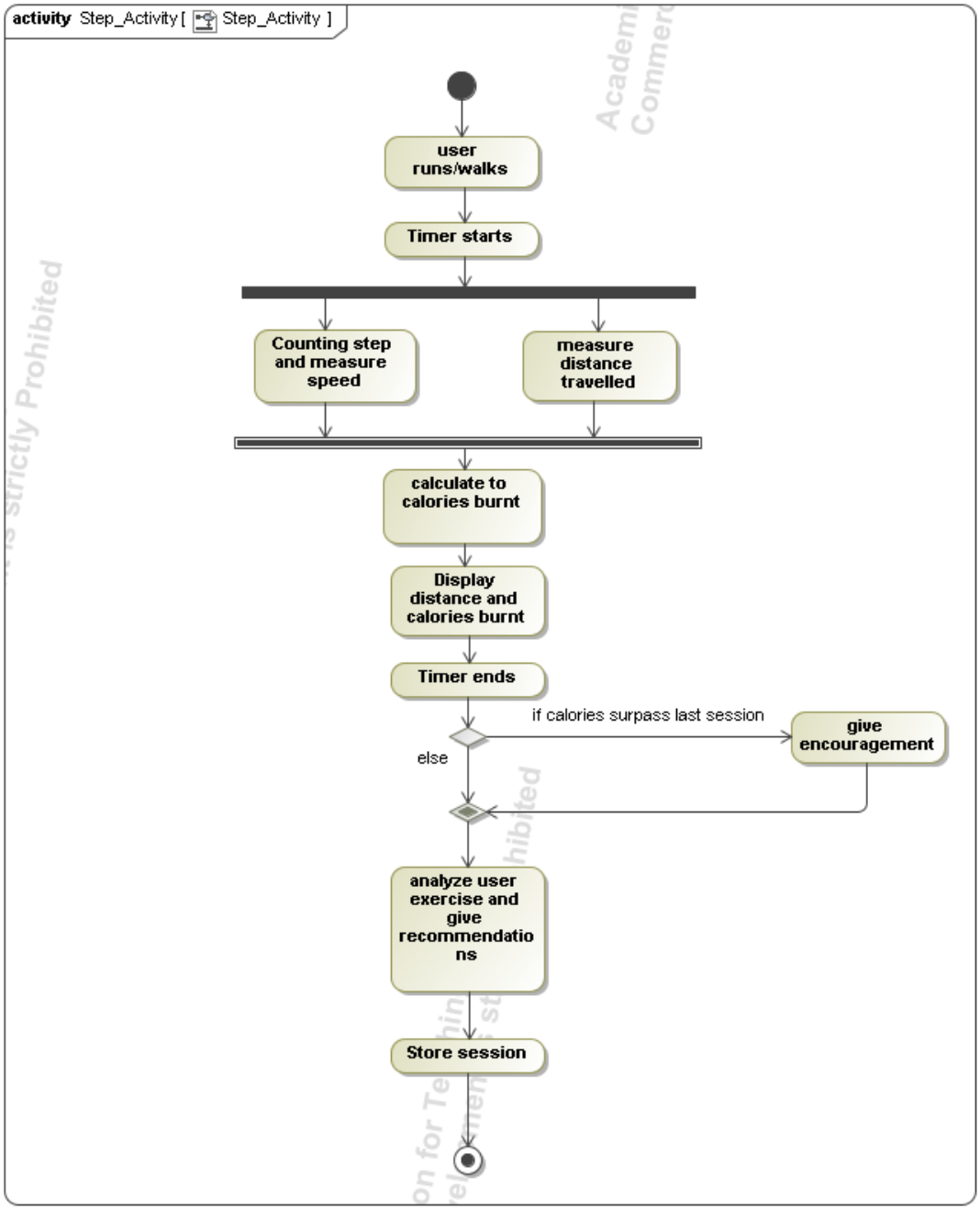

FIGURE 5.4: Step count activity diagram

The "step count" activity diagram presents the flow of the actions of the step count application of the watch. The sequence of steps such as step counting, speed measuring, calorie calculation, and distance measuring was done to generate data to show the user and store in the database for further analysis. The activity diagram shows the action flow of the step count feature:

1. User runs or walks: the user wears the watch and starts to run or walk so that the sensor can detect the activity. Start parallel action:

2. Counting steps and measuring speed: the sensor will detect the step and count the number of steps. The speed when the user runs or walks will also be captured by the sensor for further calculation in the system.

3. Measure distance: after the timer starts to run, the system will calculate how far the user has walked or run. End parallel action:

4. Calculate calories burnt: Based on the step count, speed, and distance data, the system will calculate how many calories were burnt by the user during the activity(running or walking)

5. Display the distance and calories burnt: the distance and calories burnt will be displayed on the screen so that the user can track their exercise activity.

6. Timer ends: after the exercise activity is done, and all of the measurements are taken and displayed the timer will stop the time.

   Decision selection: If the calories surpass the last session:

7. Encourage: the system will encourage the user to their harder work and higher calorie burning. Else:

8. Analyze user exercise and give recommendations: the system will advise the user that can help them burn more calories.

9. Store session: all of the measurements such as step count, speed, distance, and calories burnt data will be stored in the database for future reference.

5.2.5 UI Prototype

Default UI

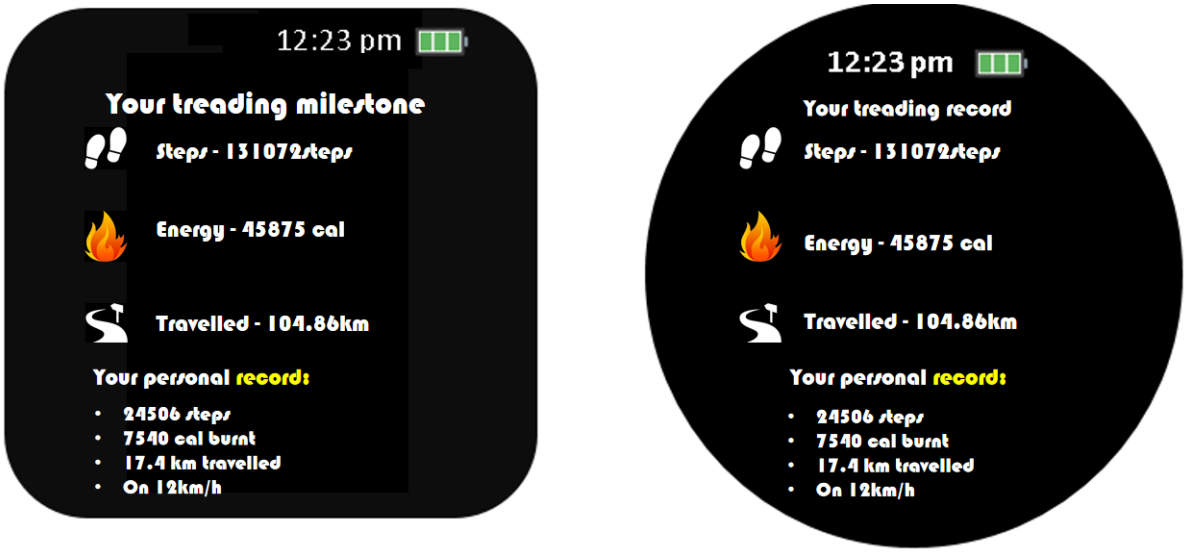

FIGURE 5.5: Step count default UI prototype

The default UI of step count functions provides user a way to look for their all-time stats as well as their personal records in a single session. To demonstrate the information mentioned above, the attributes within the UI are:

- Step: It is represented by an icon and by words. It shows the total of steps the user have made while wearing the watch.
- Energy: It is illustrates by the fire icon and the unit for calories we used is cal. It shows the total calories they have burnt for running and walking since the user started to use the watch.
- Travelled: It shows the distance the user have travelled in total. The unit we use for this attribute is kilometer.
- Personal record: It show the user record for each of the categories above and including speed which is calculated by other attributes. Each attributes is on a line separate with each other.

Encouragement UI

FIGURE 5.6: Step count Encouragement UI prototype

The encouragement UI of the step count applications whenever the user have made a new personal record in any of these categories: number of steps, distance travelled, amount of energy burnt and accordingly, their stop speed in a single session The encouragement UI includes:

- **Congratulation:** It is a congratulation by text to inform the user about the start of the encouragement session.

  Note: To announce the encouragement, a buzzing system is also installed.

- It shows all the attributes of the record breaking period.
- The categories that surpass their previous will be colored in yellow.
- A sentence of encouragement will be given at the end.

Chapter 6

Falling Alert

Falling alert is a function specifically designed to detect falling or strong changes in movement such as accidents, falls, ...etc.

6.1 Requirement Analysis

6.1.1 Snow cards

Requirements Type: Functional
For Whom: customer
User Satisfaction: High
User Dissatisfaction: High
Description: After experiencing a sudden increase in speed and rotation e.g.: a fall, bike accident, trip, ...etc, the sensors will calculate the impact and severity of the situation and if that speed and impact cross a threshold that is considered safe it will automatically contact emergency contact. If the impact and severity of the situation are still within the safe threshold, the smartwatch will display a message with an alert sound and if after a certain amount of seconds, it is not turned off, this will also trigger automatic contact to emergency services.

6.1.2 Use Case Analysis

Name	Falling Alert
ID	03
Description	After experiencing a sudden increase in speed and rotation e.g.: a fall, bike accident, trip, ...etc, the sensors will calculate the impact and severity of the situation and if that speed and impact cross a threshold that is considered safe it will automatically contact emergency contact. If the impact and severity of the situation are still within the safe threshold, the smartwatch will display a message with an alert sound and if after a certain amount of seconds, it is not turned off, this will also trigger automatic contact to emergency services.
Trigger	After user have a sudden change in speed and rotation.
Pre-conditions	The smartwatch is wore and turned on.
Post-conditions	Alert is turned off or emergency contact is reached
Basic Flow	-
Description	This is the main scenario where the speed and rotation of the user will be changed and the users is wearing the watch.
Actions	1. User experience sudden speed and rotation 2. Sensors calculate speed and severity 3. If severity pass threshold, alert emergency contact 4. If severity does not pass threshold notify user 5. If the notifications is not turned off after a set amount of seconds, alert emergency contact

6.2 UML diagrams

6.2.1 Use Case Diagram

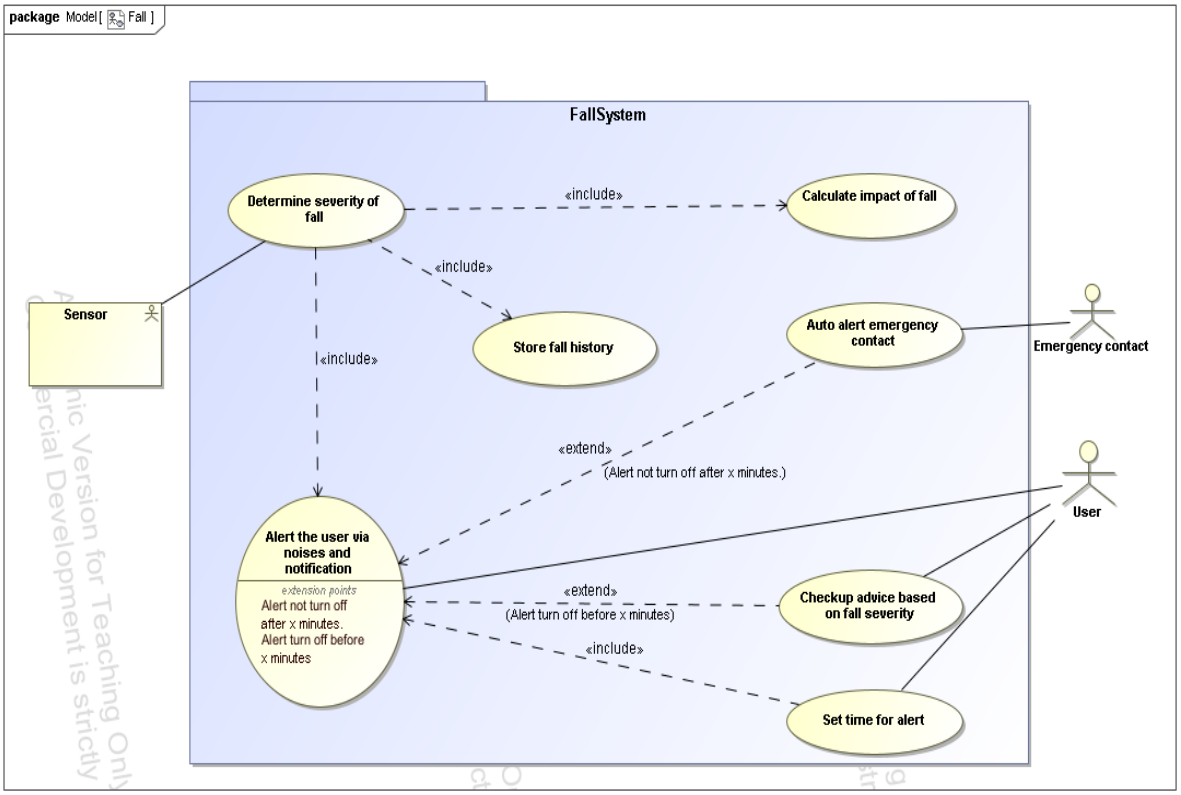

FIGURE 6.1: Fall Alert use case diagram

In this diagrams, there are 3 actors:

- Sensor : device used to record and sense coordinate to calculate and detect sudden change in velocity.
- Emergency contact : Contacts of users that can be called during an emergency or the ambulance.
- User : the users who wears the smartwatch to alert the emergency contact after a fall incident.

The main use cases will be:

- Determine the severity of the fall, calculate impact of fall: this use case will calculate the severity and the impact of the fall incident.
- Alert the user via noises and notifications: this use case will put a notification on the watch screen and also vibrate.
- Auto alert emergency contact: this use case will alert the emergency contact of the user including loved ones, ambulance, police,...etc.

There are also extension points and extends:

- If the alert is not turn off after a set amount of minute, then it will alert the actor emergency contact.

- If the alert is turn off before a set amount of minute, then it will provide some checkup advice.

Include relationships:

- Store fall history: this include relationships will always store fall impact and severity everytime the sensor detect a fall.

Note that checkup advice and set time for alert will be presented as a separate function so i will not be including them in other diagrams. Overall function:

- The actor sensor will sense real time data and will send the data to the falling alert system to calculate impact and the severity of the impact and it will also store these into the database. After that if the alert is not turn off after a while it will alert emergency contact.

6.2.2 Class Diagram

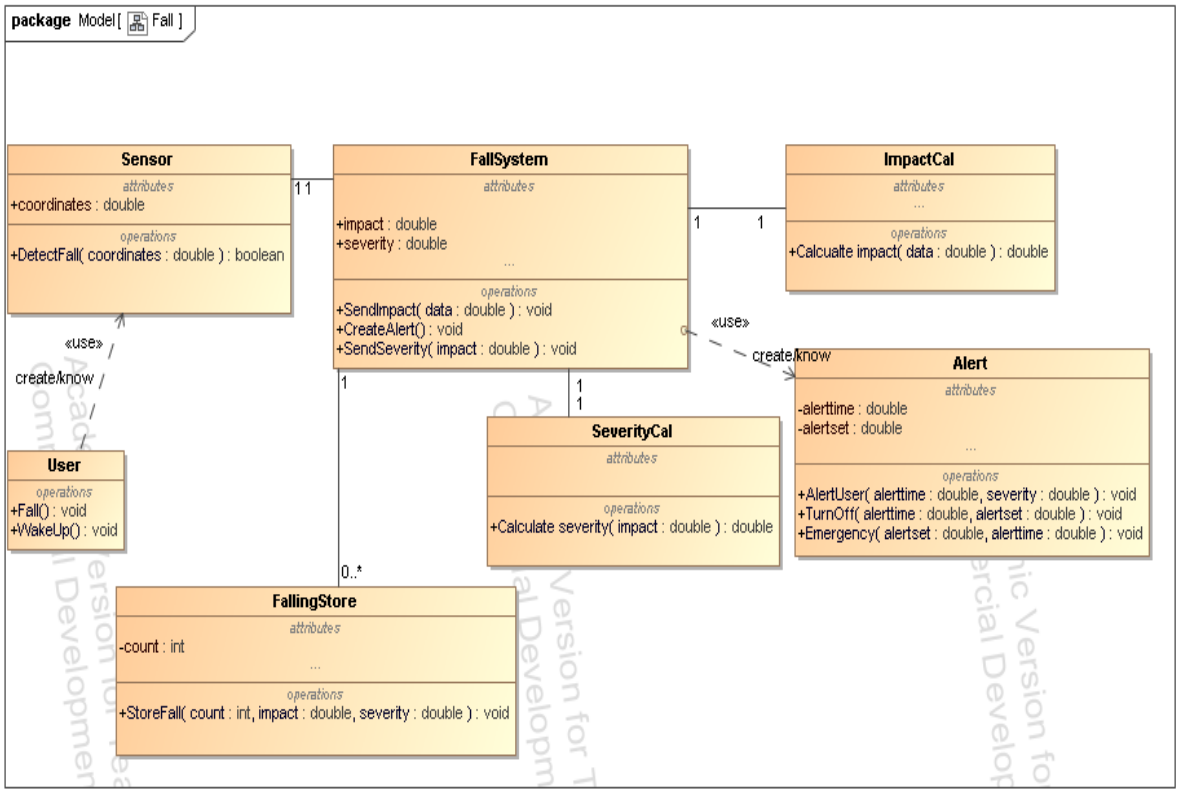

FIGURE 6.2: Fall Alert class diagram

In the class diagram, there are 7 classes:

- System: this is the center class where it will receive every values from associates class and will generate alert.
- ImpactCal: this is the class will calculate the impact of the sudden motion.
- SeverityCal: this is the class that will determine the severity of the impact.
- FallingStore: this is the class responsible for storingn the impact as well as the severity of the fall.

- Sensor: this is the class that will detect sudden motion and send the data to the system for further calculation.
- User: This class will be used to represent the user of the smartwatch.
- Alert: This is a class that will be created by the system based on certain condition and it will alert the users.

There are also some notable class methods essential for the software:

- Calculate Impact(), Calculate Severity() are functions to transform the raw data of sensors to an comprehensible data for the users.
- CreateAlert() is used to create the instance of the class alert everytime there is an accident.
- StoreFall() is also important to store the data in the database.

6.2.3 Sequence Diagram

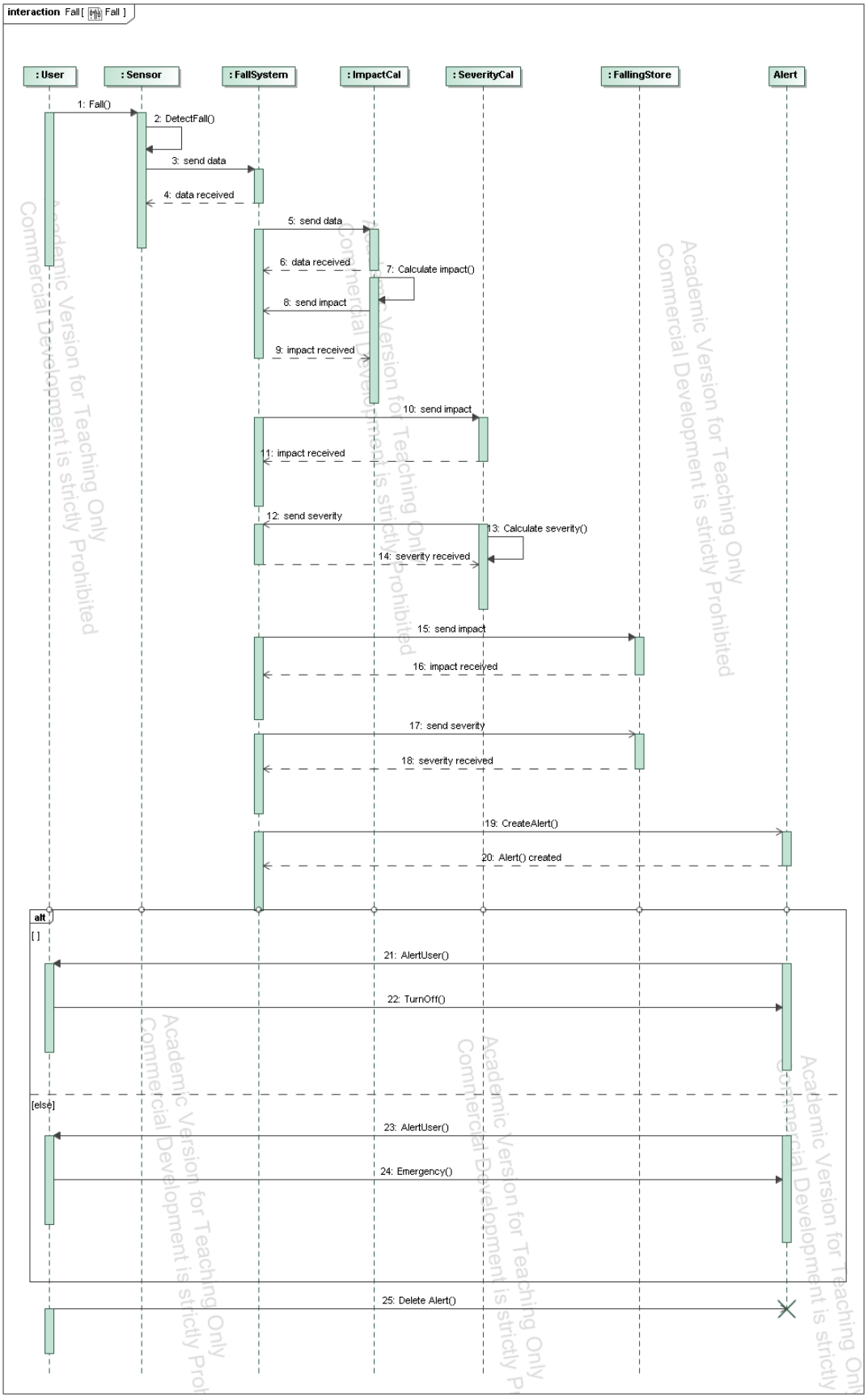

FIGURE 6.3: Fall Alert sequence diagram

There are 6 lifelines in the diagram:

- User: This lifeline represent the user who would wear smartwatch and interact with the falling alert
- Sensor: This life line represent the sensor that record data and detect fall accidents.
- System: This life line represent the system that will calculate severity and impact of the fall, storing it and alerting the user.
- ImpactCal: This life line represent the impact calculator inside of system.
- SeverityCal: This life line represent the severity calculator inside of system.
- FallingStore: This life line represent the storing of the fall including the impact and the severity
- Alert: This life line represent the Alert being created to notify the user.

In this sequence diagram, the sequence of action will be:

1. The user fall.
2. The sensors detect the fall.
3. The sensors send data to the system.
4. The system will send data to calculate impact and severity.
5. After this, the system will send impact and severity to the database.
6. Then, the system will create an Alert().
7. in the case the alerttime < alertset the system will remove object and turn off.
8. However, if this condition is not met the alert instance will call emergency function.

6.2.4 Activity Diagram

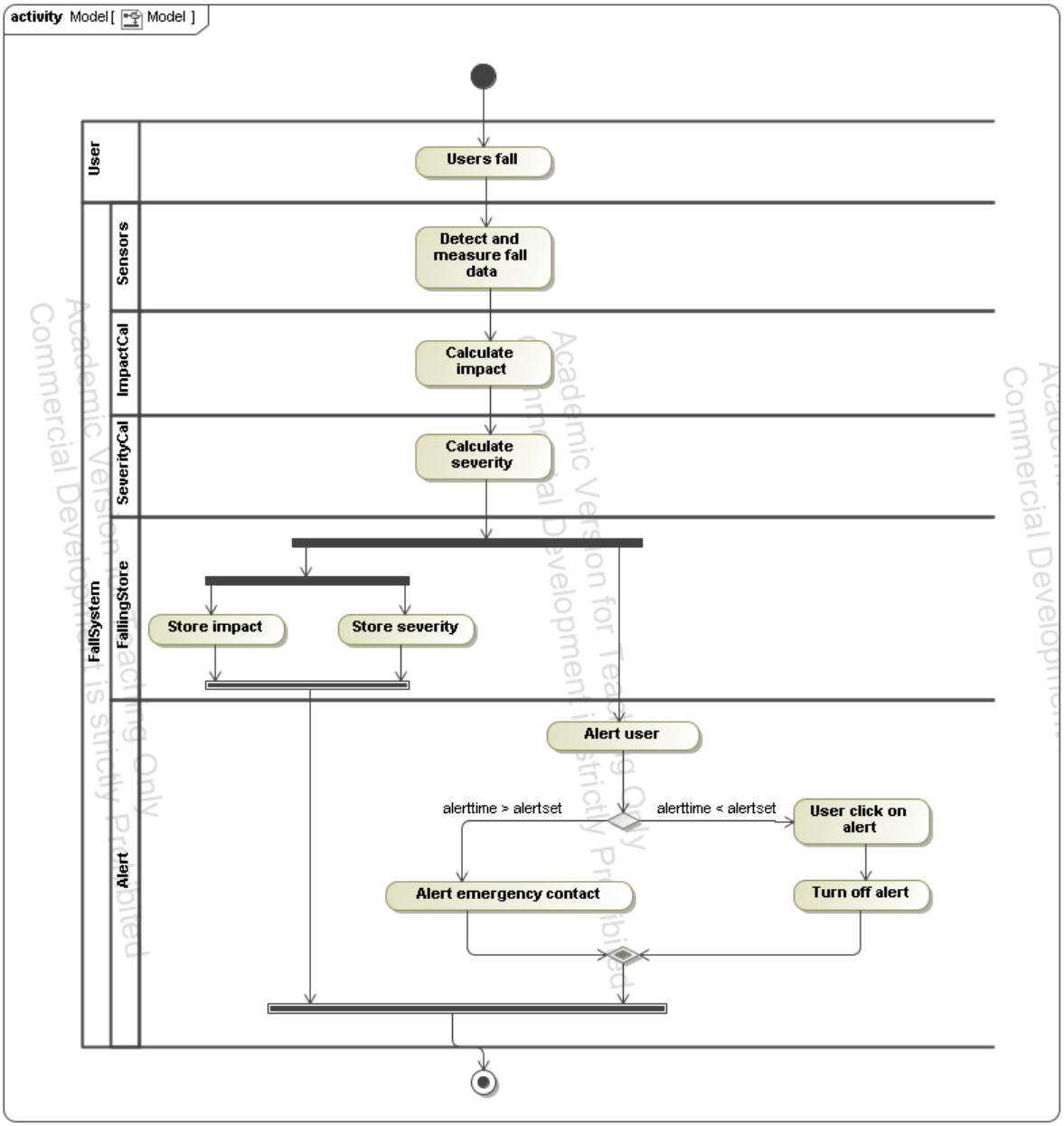

FIGURE 6.4: Fall Alert activity diagram

The following diagram include 2 main lane:

- User: determines the user action
- System. In system there are also sublane:
  - Sensors: Determines the sensors actions
  - ImpactCal: Determines actions of object ImpactCal
  - SeverityCal: Determines actions of object SeverityCal
  - FallingStore: Determines actions of falling storage
  - Alert: Determines actions of object Alert

The diagram follows this activity flow:

1. Users fall
2. Sensors will detect and measure the fall
3. ImpactCal inside the system will calculate the impact of fall
4. SeverityCal inside the system will calculate the severity of fall
5. The system will store both the impact and severity at the same time while the system will also alert the user:
   - If alerttime > alertset, than system will alert emergency contact
   - Else user will click on alert and the alert will turn off

6.2.5 UI Prototype

Notification UI

FIGURE 6.5.1: Notification UI

This is the sample User Interface of the notification when the sensors detected a fall, it will include:

- Blackening the screen
- A red alert will pop up with vibration and high pitch noise
- There will also be a clickable button with the letter IM OKAY to see whether the user is fine or not
- If this button is clicked in a set amount of time, the pop up will turn off and disappear.

Storage UI

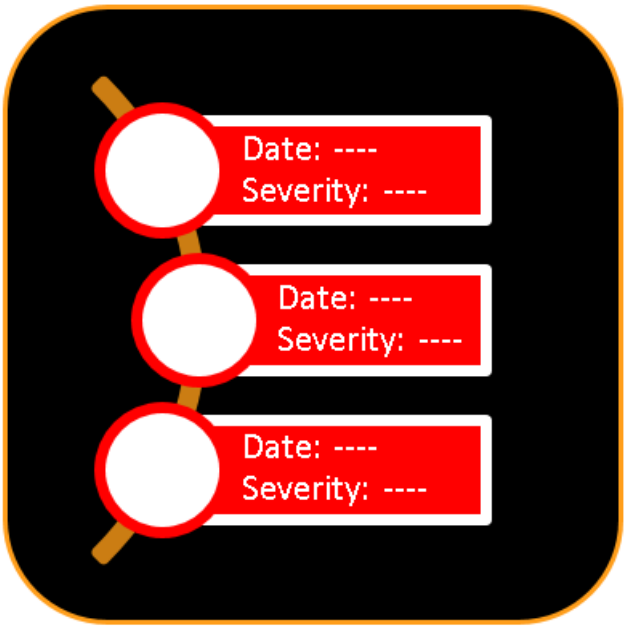

FIGURE 6.5.2: Storage UI

This is the sample of User Interface of the fall storage where user can access, this will include:

- A scrollable ring to scroll through various information slot
- There is also a square slot displaying the date and severity briefly.

Chapter 7

Weather Forecast

This feature was created expressly to track and present the current day's weather. In the event of extreme weather, it also notifies the user.

7.1 Requirement Analysis

7.1.1 Snow cards

Requirements Type: Functional
For Whom: customer
User Satisfaction: High
User Dissatisfaction: High
Description: The feature has a sensor built inside which will recognize when the user wakes up. When the user awakens, an overview of the day's weather will be provided. It will notify the user in the event of severe weather by beeping and ringing and showing an alarm along with the current weather conditions.

7.1.2 Use Case Analysis

Name	Weather Forecast
ID	07
Description	The feature has a sensor built inside which will recognize when the user wakes up. When the user awakens, an overview of the day's weather will be provided. It will notify the user in the event of severe weather by beeping and ringing and showing an alarm along with the current weather conditions.
Trigger	When the user awakens.
Pre-conditions	The smartwatch is turned on in range.
Post-conditions	
Basic Flow	-
Description	This is the main scenario when the weather is normal.
Actions	1. User wakes up. 2. The watch retrieves weather data from a database. 3. The watch provides a notification about the weather status of the day.
Alternative flows	-
Description	This scenario describes when weather is in harsh conditions.
Trigger	When the watch receives knowledge about an awful weather occurrence.
Pre-condition	The watch is turned on in-range and the weather is awake.
Post-condition	When the alarm is turned off or when the watch is turned off.
Actions	1. The watch retrieves information regarding the approaching extreme weather. 2. The watch sends out a notification by buzzing, ringing and displaying the incoming weather status.

7.2 UML diagrams

7.2.1 Use Case Diagram

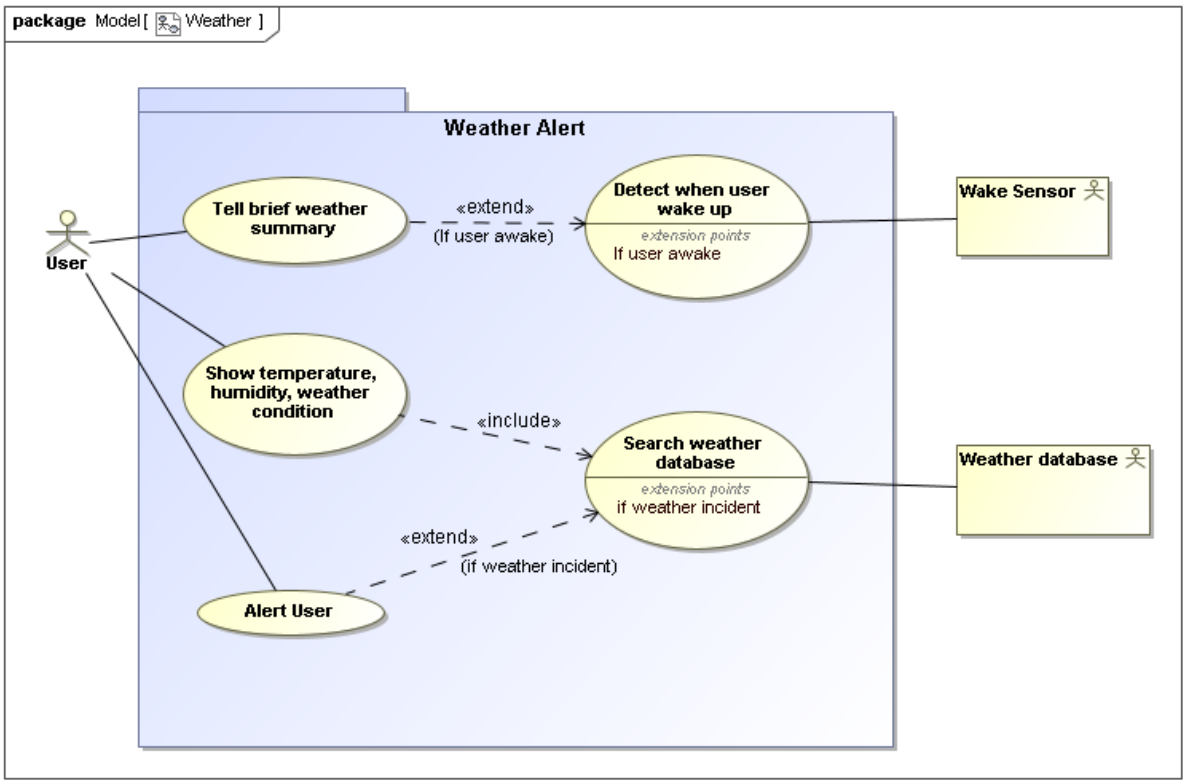

FIGURE 7.1: Weather Alert use case diagram

The diagram shows the following main actors: the User, the Weather Database, the Sensors and the System. The System takes input from the sensors that are triggered if the user wakes up. This extension calls the "Tell brief weather summary" use case. If the user want a detailed description, he or she can activate the "Show temperatures, humidity, weather condition" use case. It has an include relationship which will search inside the weather database based on user's location. During that, the system also determines whether an extreme weather incident is about to occur. If so, the user will receive an alert from the system.

Main Actors

- **User:** The smartwatch user who interacts with the weather feature to receive real-time weather updates and notifications.
- **Weather Alert:** A system component that monitors weather conditions and triggers notifications when necessary, such as severe weather warnings.
- **Sensors:** The sensor is used to detect whether the user wakes up every morning, allowing the weather system to provide a brief weather summary to the user.
- **Weather Database:** The database provides real-time weather status and approaching extreme climatic conditions to the system in order to notify the user.

Primary Use Cases

- **Show Temperature, Humidity, and Weather Condition:** This use case retrieves real-time weather data from sensors and displays it on the smartwatch screen.
- **Tell Brief Weather Summary:** This use case provides a concise summary of the current weather conditions, including only temperature, humidity, and wind strength for the next few hours. It is triggered by the user waking up every morning.
- **Search Weather Database:** This use case collects long-term forecasts, weather maps, and severe weather alerts, among other extensive meteorological data, from a distant weather database. To give more detailed weather information, it is included within Tell Brief Weather Summary use case.

Extension Points and Extensions

- **Extension Point 1:** This extension point is triggered when the user wakes up. The "Tell brief weather" summary is invoked to provide a quick synopsis of the current weather data to the user. This ensures that the user is always greeted with up-to-date weather information upon waking up.
- **Extension Point 2:** This extension point is triggered when there is a weather incident, such as a severe weather warning or extreme temperature fluctuations. The extended relationship, Alert User, are invoked to notify the user of the weather incident and provide additional information.

Include Relationship

- The "Search Weather Database" use case is included within the "Show temperatures, humidity, weather conditions" use case. This means that the "Search Weather Database" use case is always executed when the "Tell Brief Weather Summary" use case is invoked, ensuring that the user receives comprehensive weather information.

Overall Functionality

- Users with smartwatches can access real-time weather updates, including forecasts, current conditions, and comprehensive data from a remote meteorological database via the weather app. Additionally, it has tools that notify users and their linked devices of severe weather events.

7.2.2 Sequence Diagram

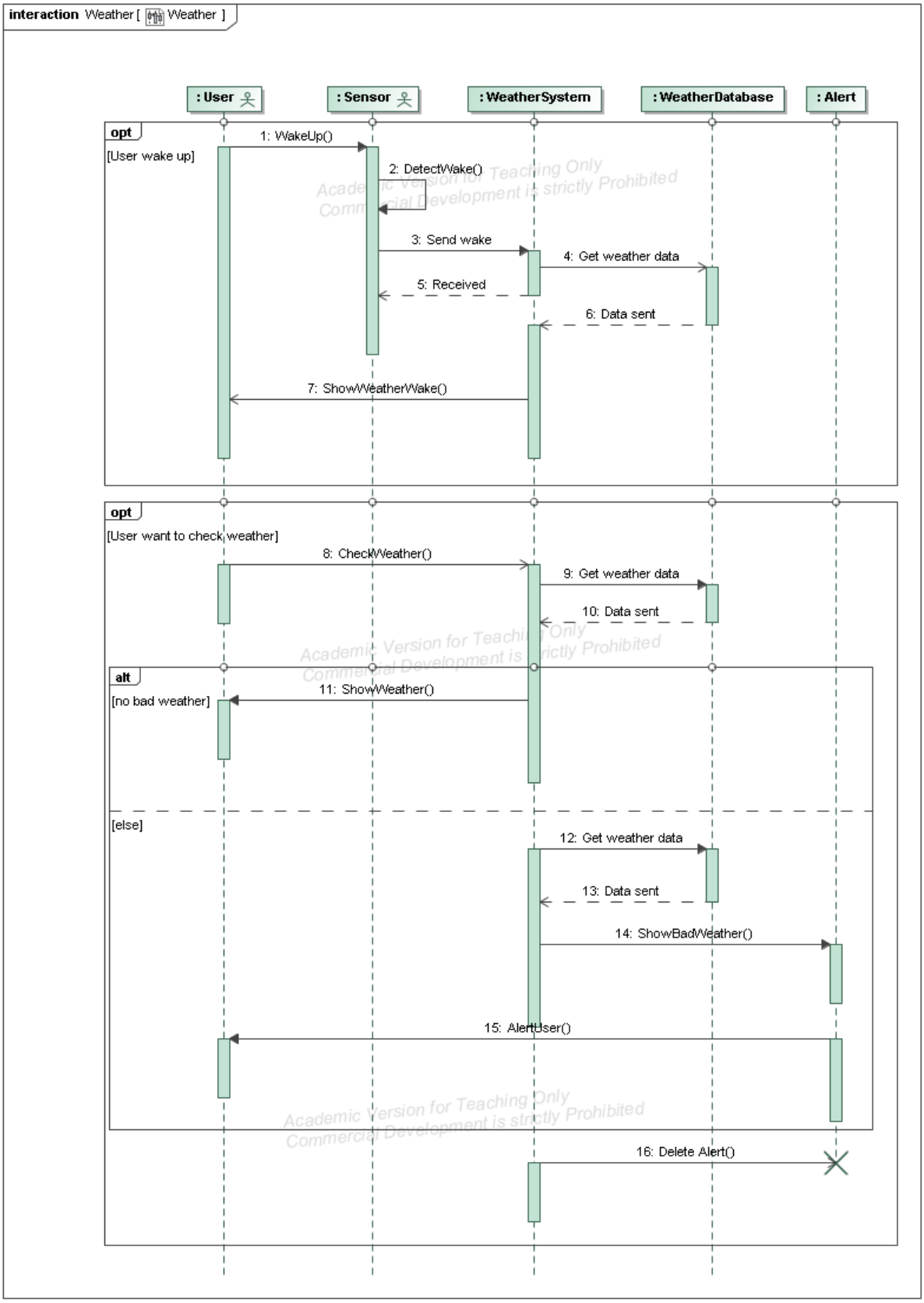

FIGURE 7.2: Weather Alert sequence diagram

The provided sequence diagram illustrates the interaction between various components of a smartwatch weather feature, showcasing the sequence of steps involved in retrieving and displaying weather data.

The diagram begins with the user initiating the weather feature by either waking up or manually requesting weather information.

Option 1: User Wakes Up

The smartwatch user invokes the "WakeUp()" operation, triggering the "Detect/Wake()" method within the "WSensor" component. This method detects that the user is awake and sends a notification to the "Weather System" component via the "Send wake" operation. Upon receiving the wake notification, the "Weather System" initiates the process of fetching weather data. It first calls the "Get weather data" operation on the "Weather Database" component. The "Weather Database" retrieves the relevant weather information from its storage and sends it back to the "Weather System". Finally, the "Weather System" displays the weather summary to the user.

Option 2: User Manually Requests Weather Information The user executes the "CheckWeather()" operation, which activates the "Get weather data" inside the "Weather System" then the database returns relevant weather information to the user in case of there are no predicted extreme events. In the event of there is a severe climate occurrence, the "Weather Database" will inform the "Weather System". The "Weather System" will create a "WAlert" via "ShowBadAlert()" operation. which will alarm the user by "AlertUser()" function.

Lifelines and Events for Smartwatch Weather Feature

Lifelines:

- **WUser:** The smartwatch user who initiates the weather feature and interacts with it to obtain weather information.
- **WSensor:** A component inside the smartwatch that detects the user's wakefulness and sends notifications to other components.
- **Weather System:** A central component responsible for retrieving weather data from the Weather Database and displaying it to the user.
- **Weather Database:** A remote database that stores real-time and historical weather information.
- **WAlert:** An optional component that displays unusual weather information on the smartwatch screen, complementing the Weather System's functionality.

Events

- **WakeUp():** This event is triggered by the user when they wake up or manually request weather information.
- **Detect/Wake():** This method is executed by the "WSensor" component to detect the user's wakefulness and send a notification to the Weather System.
- **Send wake:** This message is sent by the "WSensor" component to the "Weather System", indicating that the user has awakened.
- **Get weather data:** This operation is called by the "Weather System" to retrieve weather data from the "Weather Database".

- **Received:** This message is sent by the "Weather Database" to the "Weather System", confirming the receipt of the weather data request.
- **Check for severe weather:** This operation is performed by the "Weather System" to analyze the retrieved weather data for any severe weather conditions.
- **ShowWeather():** This method is called by the Weather System to display the current weather data to the user.
- **ShowBadWeather():** This operation is triggered by the Weather System if severe weather is detected, notifying the user of the hazardous conditions.
- **Get weather data:** This operation is called by the Weather System to retrieve weather data from the Weather System, potentially for display on the smartwatch screen even when it is in standby mode.

7.2.3 Class Diagram

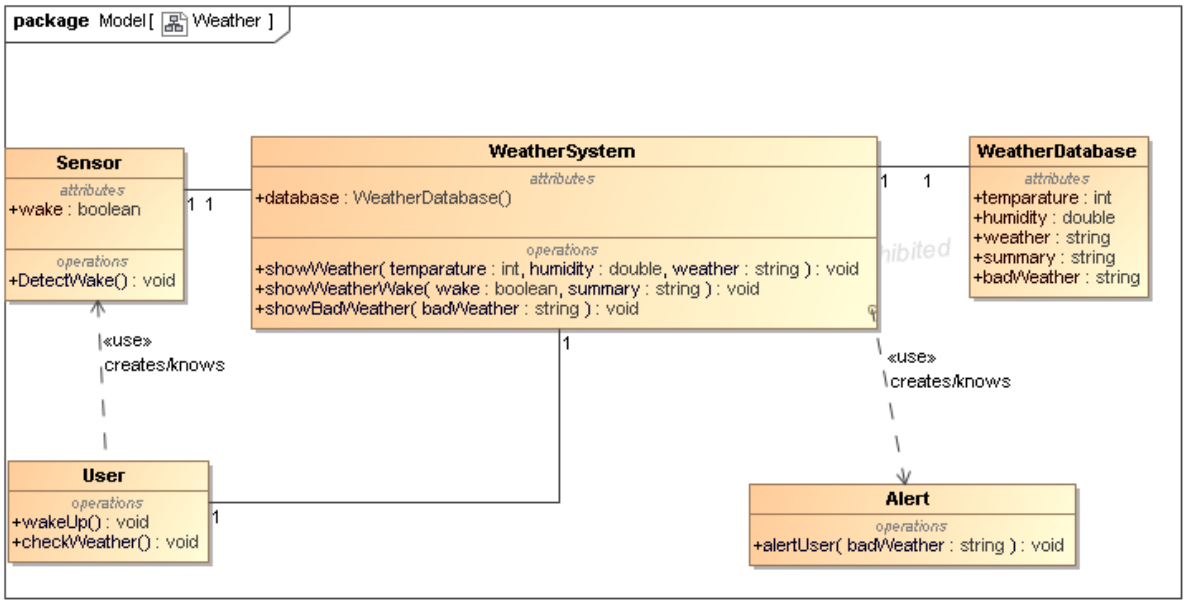

FIGURE 7.3: Weather Alert class diagram

The Weather Forecast Class Diagram offers a systematic depiction of the essential elements contained in a weather forecasting system. This figure represents the connections and exchanges between five fundamental classes: 'Weather System', 'WSensor', 'WUser', 'WAlert', and 'Weather Database'.

Weather System class

Operations:

- `Alert()`: Initiates the generation and distribution of weather alerts to relevant users based on predefined criteria. Triggers the WAlert class to generate and send notifications about significant weather events or changes.
- `ShowWeather()`: Displays the current weather information to users.

- `ShowWeatherWake()`: Displays wake-up notifications to users based on weather conditions. Triggers wake-up notifications for users, particularly when specific weather conditions are met.
- `ShowBadWeather()`: Presents information about adverse weather conditions. Communicates alerts or warnings regarding potentially hazardous weather situations to WUser instances.

WSensor class

Attributes:

- `wake`: Represents the wake status, indicating whether specific weather conditions warrant a wake-up notification. Example: true if wake-up conditions are met, false otherwise.

Operations:

- `DetectWake()`: Evaluates current weather conditions to determine if a wake-up notification is warranted. Utilizes weather data and predefined criteria to assess whether users should be alerted with a wake-up notification.

WUser class

Operation:

- `WakeUp()`: Initiates a wake-up routine for the user based on predefined preferences and real-time weather conditions. Triggers wake-up activities, such as adjusting the alarm time or sending notifications, as per the user's preferences and detected weather conditions.
- `CheckWeather()`: Enables the user to manually check current weather conditions. Retrieves and displays real-time weather information, providing the user with the latest updates on temperature, humidity, and weather type.

WAlert class

Operations:

- `AlertUser()`: Sends weather-related alerts and notifications to WUser instances. Initiates the process of notifying users about significant weather events, changes, or other relevant information based on their preferences.

Weather Database class

Attributes:

- `temparature`:Represents the recorded temperature in degrees. Example: 25 (for 25 degrees Celsius).
- `humidity`: Represents the recorded humidity level. Example: 0.65 (for 65
- `weather`: Describes the prevailing weather conditions. Example: "Clear Sky," "Rainy," "Snowfall."

- `summary`: Provides a concise summary of the weather conditions. Example: "Partly cloudy with a chance of rain."
- `badWeather`: Indicates the occurrence of severe or adverse weather events. Example: "Storm Warning," "Heavy Snowfall Alert."

7.2.4 Activity Diagram

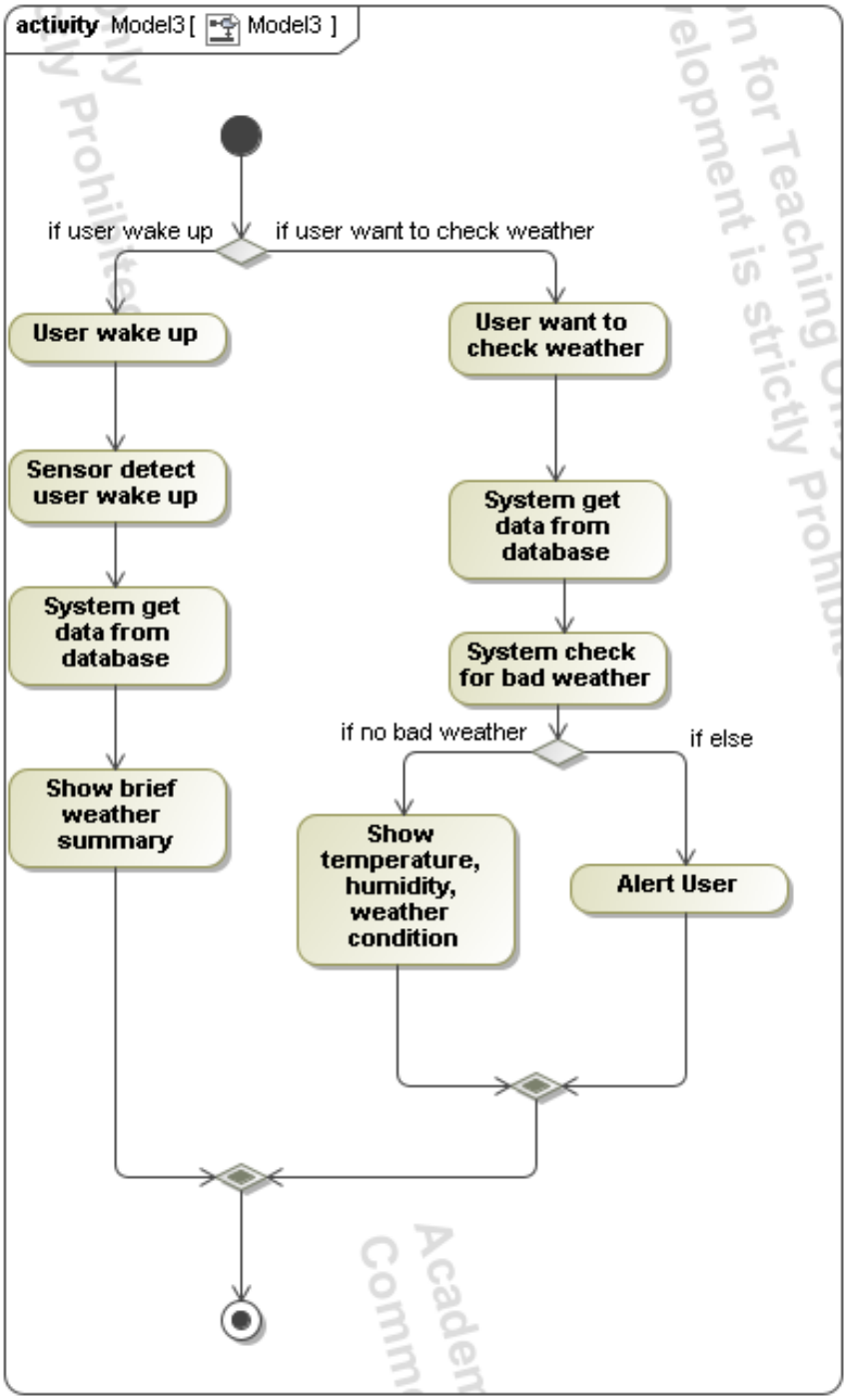

FIGURE 7.4: Weather Alert activity diagram

- The activity diagram captures two primary user scenarios: automatically showing a brief weather summary and manually checking the weather.

- The first branch focuses on automatically showing a brief weather summary.
- The second branch involves the user checking the weather manually, with different flows for fine and bad weather conditions.
- This diagram represents the dynamic interactions between the user, system components, and the database during weather-related activities.

First Branch:

Step 1: User Wakes Up

User initiates the wake-up process.

Step 2: Sensor Detects User Wake Up

'WSensor' class detects the user waking up.

Step 3: System Gets Data from Database

'Weather System' class retrieves relevant weather data from the 'Weather Database'.

Step 4: Show Brief Weather Summary

'Weather System' displays a concise weather summary to the user.

Second Branch:

Step 1: User Wants to Check Weather

User decides to manually check the weather.

Step 2: System Gets Data from Database

'Weather System' class retrieves real-time weather data from the 'Weather Database.

IF THE WEATHER IS FINE:

Step 3: Show Temperature, Humidity, Weather Condition

'Weather System' presents detailed weather information to the user.

IF THE WEATHER IS BAD:

Step 4: System Checks for Bad Weather

'Weather System' verifies if adverse weather conditions are detected.

Step 5: Alert User

'WAlert' class initiates an alert to inform the user about the bad weather conditions.

7.2.5 UI prototype

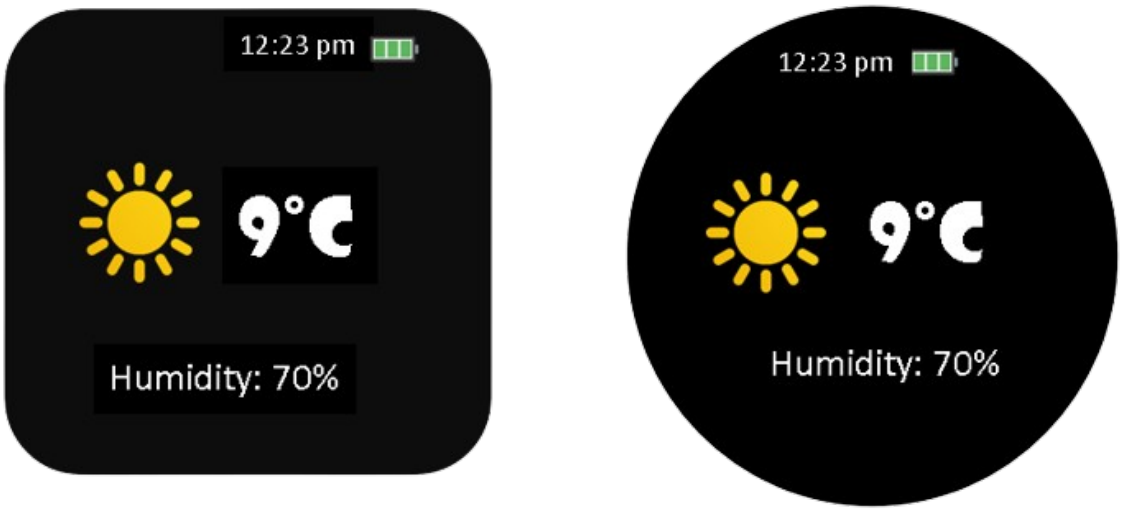

FIGURE 7.5.1: Weather Alert UI prototype

The UI prototype for the smartwatch weather application offers a comprehensive array of features aimed at keeping users well-informed about their surrounding weather conditions. These features encompass:

- **Current Weather Display:** The UI prototype presents a clear and easily readable interface showcasing the current weather conditions.
- **Temperature Information:** The UI prototype provides users with real-time updates on the current temperature.
- **Humidity Details:** Users can readily access information about the current humidity through the intuitive UI prototype.
- **Real-time Updates:** The application seamlessly fetches real-time weather updates from a remote weather database, ensuring users have the latest information at their fingertips.

Additionally, users have the convenience of tracking both the current time and the battery percentage.

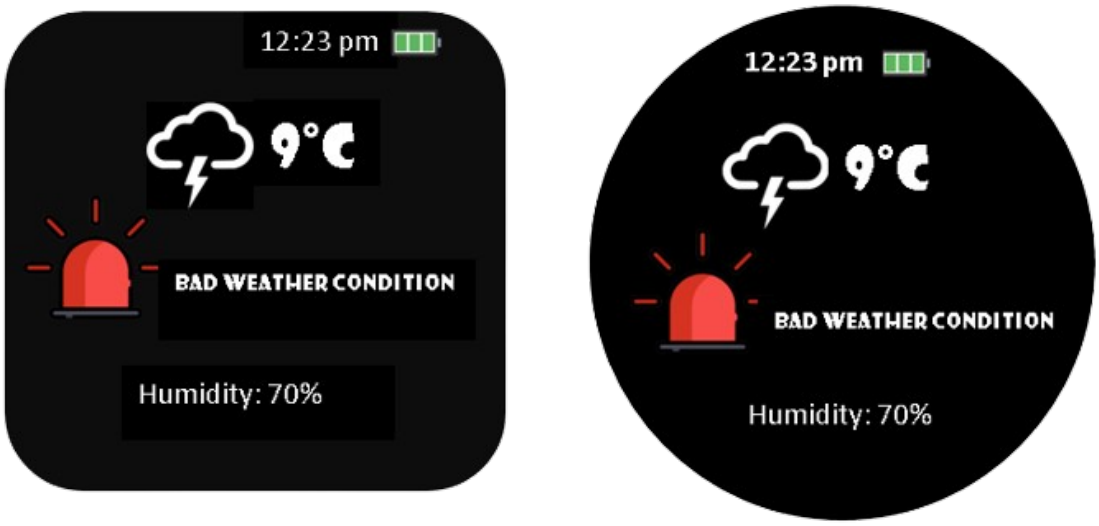

FIGURE 7.5.2: Weather Alert UI prototype

The UI prototype depicts a simplified UI for a weather feature on a smartwatch. The design utilizes a clean and minimalist aesthetic, with a focus on displaying essential weather information in a clear and concise manner.

The weather information panel occupies in the middle of the screen, divided into two sections: temperature and humidity. The temperature is displayed in bold, white digits, preceded by the current temperature in degrees Celsius. The humidity is displayed in a smaller font below the temperature, accompanied by a percentage symbol.

A notable feature of the prototype is the use of red text to indicate a severe weather condition. The text "BAD WEATHER CONDITION" appears in White and the weather information panel, alerting the user to the presence of hazardous weather conditions. This emphasis on red draws attention to the potential danger and encourages the user to take necessary precautions.

Chapter 8

Clock system

The clock system application provides user with a method to keep track of date and time as well as a function to set up alarm.

8.1 Requirement Analysis

8.1.1 Snow cards

Requirements Type: Functional
For Whom: customer
User Satisfaction: High
User Dissatisfaction: High
Description: The clock system application will take user settings and retrieve information about date and time to send back to the user. It also has a function for user to create an alarm, if the time is match, the alarm will be notify by displaying on the screen, vibrating. After a certain period of time, if the alarm is not turned off, it will increase it intensity until the user turn it off or the watch is shut down.

8.1.2 Use Case Analysis

Name	Time System
ID	08
Description	The clock system application will take user settings and retrieve information about date and time to send back to the user. It also has a function for user to create an alarm, if the time is match, the alarm will be notify by displaying on the screen, vibrating. After a certain period of time, if the alarm is not turned off, it will increase it intensity until the user turn it off or the watch is shut down.
Trigger	After the configuration for the clock system is set
Pre-conditions	The smartwatch is wore and turned on.
Post-conditions	
Basic Flow	-
Description	This is the main scenario which the clock system runs normally, update time and date real-time.
Actions	1. The clock get the information of date and time from the database. 2. The clock show the date and time to the user.
Alternative Flow	-
Description	This scenario describes the situation when you have to set up the clock.
Actions	1. The user inputs his or her location, alarm interval and alarm times. 2. The system stores the data.
Alternative flow	-
Description	This scenario describes when an alarm time is met.
Actions	1. The system create an alarm. 2. It alarm the user by vibrating. 3. If it is not turned off within the alarm interval, its strength increases. 4. The user turns off the alarm.

8.2 UML diagrams

8.2.1 Use Case Diagram

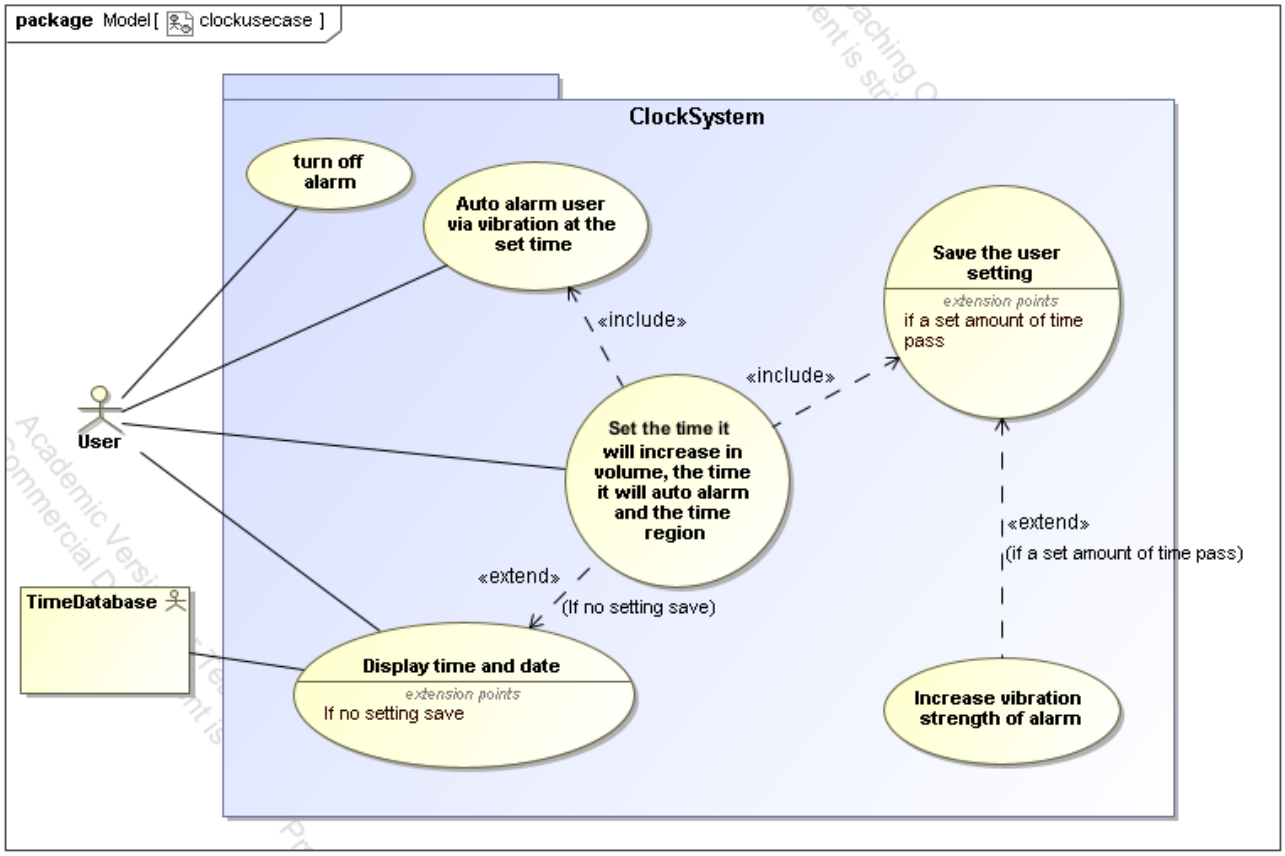

FIGURE 8.1: Time system use case diagram

The Clock System involves two main actors: User and TimeDatabase, interacting through various use cases.

Actors

- **User:** The primary user interacting with the Clock System.
- **TimeDatabase:** Responsible for providing the current time and date information to the Clock System.

Use Case

- **turn off alarm:** The user has the capability to manually turn off the alarm when it is active.
- **Auto alarm user via vibration at the set time:** The user can set the alarm to vibrate automatically at a specified time.
- **Set the time it will increase in volume, the time it will auto alarm and the time region:** The user has the ability to configure settings such as the time at which the alarm volume increases, the time for the auto alarm to trigger, and the time region preferences.

- **Display time and date:** The user can view the current time and date on the clock display.
- **Save the user setting:** The user can save the configured settings for future use.
- **Increase vibration strength of alarm:** The user has the option to enhance the vibration strength of the alarm.

Associations

User Associations

- **Association with "turn off alarm":** The user interacts directly with the clock system to manually turn off the alarm when it is active. This association signifies the user's control over alarm deactivation.
- **Association with "Auto alarm via vibration at the set time":** The user, through the clock system, sets a specific time for the alarm to vibrate automatically. This association represents the user's ability to customize alarm behavior.
- **Association with "Set the time it will increase in volume, the time it will auto alarm and the time region:":** This association indicates that the user can configure various settings, including the time at which the alarm volume increases, the time for the auto alarm to trigger, and preferences for specific time regions.
- **Association with "Display time and date":** The user engages with the clock system to check and view the current time and date. This connection demonstrates the user's desire for up-to-date information.

TimeDatabase Associations

- **Association with "Display time and date":** The TimeDatabase supplies the clock system with the essential data to precisely showcase the current time and date. This connection emphasizes the importance of the TimeDatabase in providing crucial time-related information.

Use Case "Set the time it will increase in volume, the time it will auto alarm and the time region"

- **Include "Auto alarm user via vibration at the set Time":** Configuring the overall alarm behavior involves setting the time for auto alarm vibration, which is a crucial step. By including this feature, we take into account the user's preferences in a more comprehensive way.
- **Include "Save user setting":** Ensuring user settings are saved is an essential aspect of the configuration process. This feature indicates that, when adjusting preferences, users can choose to save those settings for later use.

In Case No Settings Are Saved

- **Extend "Display time and date" to ""Set the time it will increase in volume, the time it will auto alarm and the time region":** If the user has not saved any settings, the clock system might ask the user to set up these configurations. This extension guarantees a seamless experience by providing step-by-step guidance during the configuration process, if needed.

Use Case Save user setting

- **Extend "Save User Setting" to "Increase Vibration Strength of Alarm":** If a certain period of time elapses without the user saving their settings, the clock system might automatically increase the intensity of the alarm's vibration. This extension includes an automated response to motivate users to promptly customize their preferences.

8.2.2 Sequence Diagram

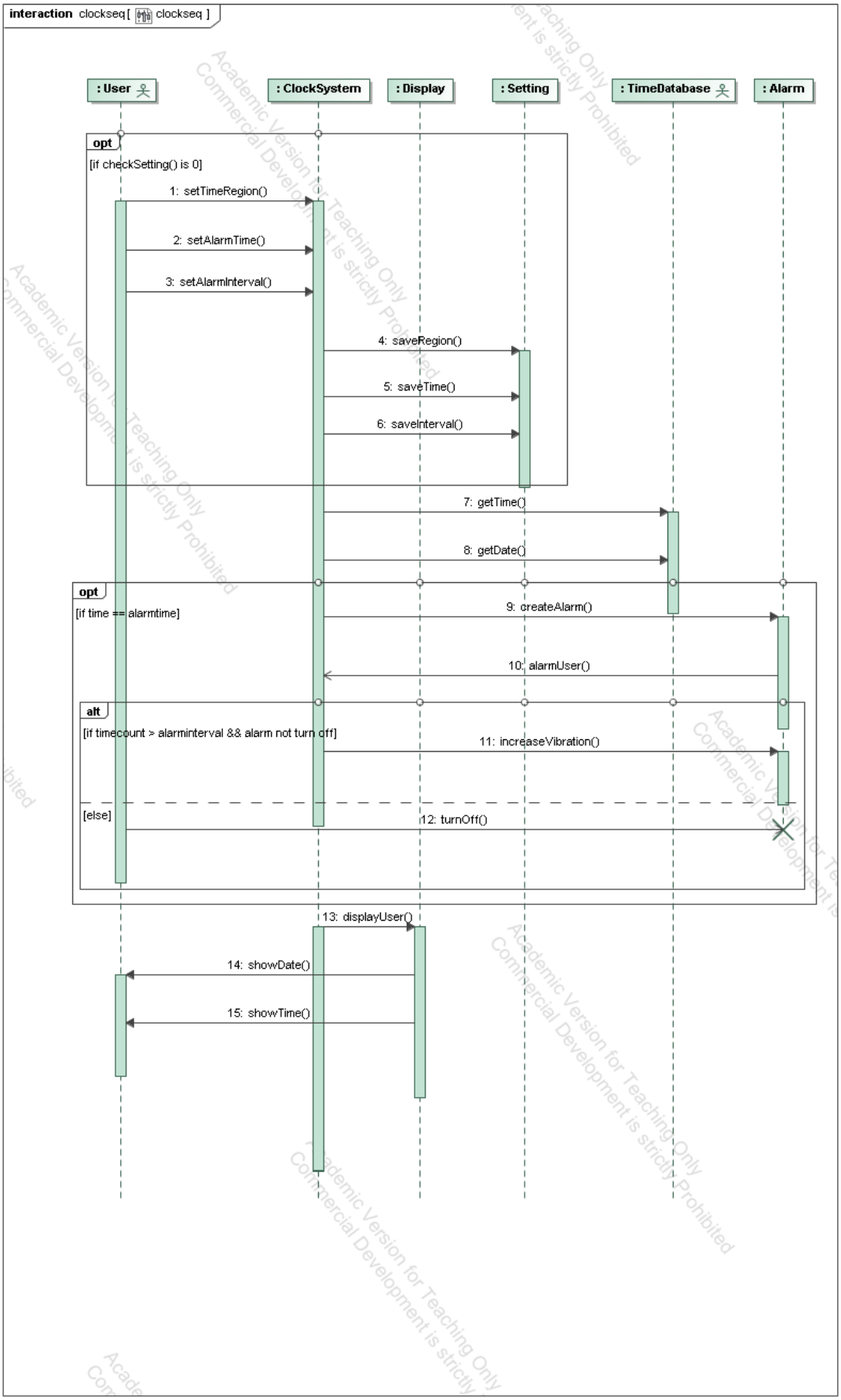

FIGURE 8.2: Time system sequence diagram

The sequence diagram of the Time System application describes the interactions between components of the application as well as the order of actions that take place in order that the function will work properly. It starts by checking whether the time system is modified the user or not. If not it will ask user for the configuration of the system. After retrieving input from the user, the system stores the configuration and get the date and time according to the provided information from the user. A display is provided to show the date and time is provided to the user. In case the user create an alarm, it will notify the user by buzzing, ringing and also by the UI to show that the time has come. It also has a timeout for the alarm, after a certain amount of time, if it is not turned off, the extend of methods of notification will be enhanced until it is turned off.

Detailed Analysis about Lifetimes and Events

Lifetimes

There are 6 lifetimes in this sequence diagram:

- **User:** This represents the user. The user is responsible for configuring the time system for the first time and also the receiving end of the date and time data delivered by the system.
- **Clock system:** This lifetime is responsible for most of the actions of the application. It acts as an intermediate to retrieve user settings and send out information to the user. It also manages the Alarms.
- **Display:** This lifetime is responsible for showing the data supplied by the system to the user.
- **Setting:** This lifetime stores the user's configuration when he or she first start to use the application.
- **TimeDatabase:** This is where the clock system will take data from to show the user.
- **Alarm:** This class only comes to existance when the user create it. It will alarm to the user according to the signals from the system. After a period, if the alarm is not turned off, it will increase its strength. It is terminated when turned off by the user.

Events

Here is the order of actions in this sequence diagram:

1. The user gives input about region, time and time interval for alarm (setTimeRegion(), setAlarmTime, setAlarmInterval()).

2. The system sends the input to the setting and store it there(saveRegion(), saveTime(), saveInterval()).

3. The system retrieves data about date and time based on the configuration(getTime(), getDate()).

Note: Number 1 to 3 is under **Option1: checkSettings() == 0:** These events only occur if the system has not received any configuration from the user before, it happens in case of a new smart watch or resetting, etc.

Option2: If time == alarmTime: These events occur when the alarm is set and alarm time is met.

4. An alarm is created via signals from the system and inform the user about the time(createAlarm(), alarmUser()).

   (Alternative: Case: if timecount > alarminterval " alarm did not turn off:) This case implies the situation when the alarm is going for a very long time but has not been turned off.//

5. The intensity of the vibration for alarming will go up (increaseVibration()).

   Alternative: Case: Else: The user turn off the alarm manually

   Note: This is a sub-event of Option 2. Note: This is the end of both option 2 and the sub-event alternative

6. The system tells the display to show the data to the user (displayUser(), showDate(), showTime()).

8.2.3 Activity Diagram

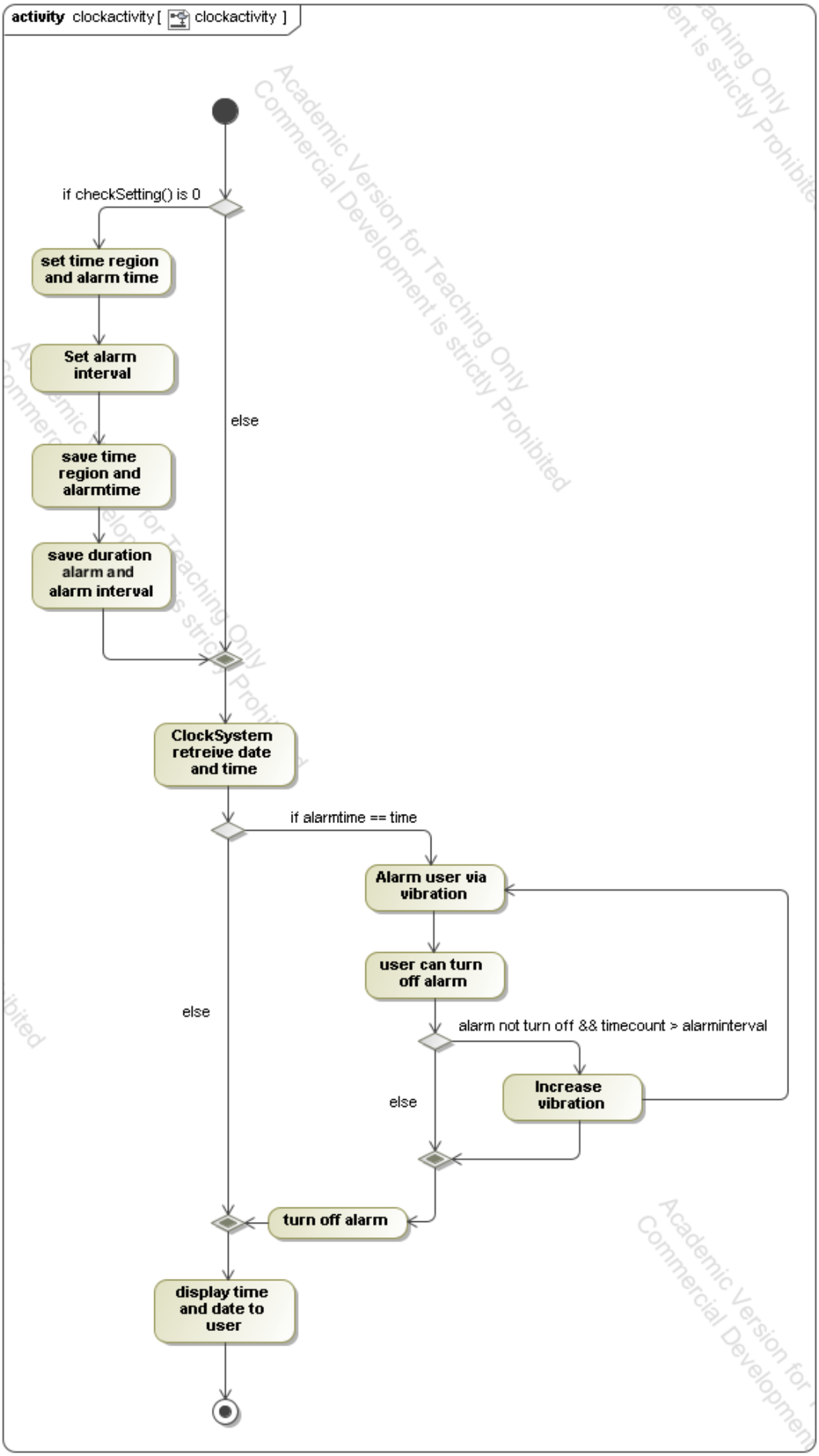

FIGURE 8.3: Time system activity diagram

The activity diagram presents the process of the clock system and time alarm system. The activity flow in this diagram describes how the ClockSystem and its features interact with each other to execute their functions.

- The first if condition: If `checkSetting()` returns 0 (the setting section was not done):

1. The user sets the time zone where the user is currently located and sets the alarm time that the user needs to be alarmed.
2. The user sets the duration of the alarm (number of minutes that the alarm will last) and sets the alarm interval (number of minutes after the alarm starts that the watch will increase the vibration level to wake the user up more efficiently).
3. The clock system saves the time region and alarm time in the setting feature for further usage.
4. The clock system saves the alarm duration and the alarm interval in the setting feature for further usage.

- The first else condition: Else:

5. The date and time data receiving of the `ClockSystem` from the clock system database.

- The second if condition: If the alarm time is equal to the current time:

6. Alarm the user via vibration and screen.

- The third if condition: If the time count parameter reaches the alarm interval parameter:

7. The watch will increase the vibration level. The flow continue to move back to step 6: Alarm user.

- The third else condition: Else: (the user choose to cancel the alarm and the time count did not reach the alarm interval.)

8. The system will remove the alarm when the user turns off the alarm.

- the second else condition: Else:

9. Display the time and date to the user.

8.2.4 Class Diagram

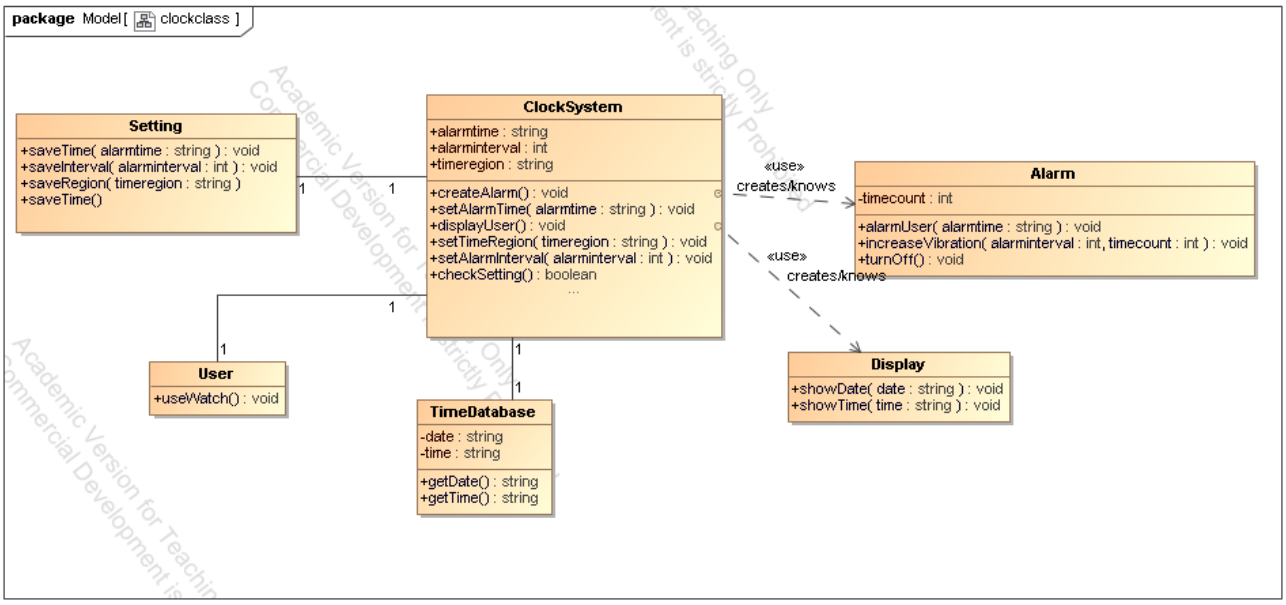

FIGURE 8.4: Time system class diagram

The clock class diagram depicts the classes and their relationships that constitute the clock system and time alarm system feature of a smartwatch. The fundamental classes involved are:

ClockSystem Class

Attributes:

- `alarmtime: string`: String set by the user to indicate the exact time for the alarm.
- `alarminterval: int`: Integer determining the minutes before the watch vibrates more loudly after the alarm sounds.
- `timeregion: string`: String indicating the user's current time zone.

Operations:

- `createAlarm() : void`: Creates an alarm when the current time and the alarm time are similar.
- `setAlarmTime(alarmtime : string) : void`: Sets the time for the user to be alarmed.
- `displayUser() : void`: Shows the user the present time and date on the screen when opening the clock widget.
- `setTimeRegion(timeRegion : string): void`: Sets the time zone where the user is located.
- `setAlarmInterval() : void`: Sets the number of minutes after the after-alarm time to increase the vibration level.
- `checkSetting() : boolean`: Checks if all setting parameters are set.

TimeDatabase Class

Attributes:

- `date : string`: String representing the current date.
- `time : string`: String representing the recent time.

Operations:

- `getDate() : string`: Returns the current date.
- `getTime() : string`: Returns the recent time.

Alarm Class

Attributes:

- `timecount : int`: Watch raises vibration level when `timecount` reaches `alarminterval`.

Operations:

- `alertUser( alarmtime: string): void`: Turns on the alarm, shows it on the screen, and vibrates to alert the user.
- `increaseVibration(alarminterval : int, timeout : int) : void`: Increases the vibration level when `timecount` reaches `alarminterval`.
- `turnOff() : void`: Removes the alarm when the user turns it off.

Display Class

Operations:

- `showDate(date : string) : void`: displays the current date on the screen when the user opens the clock widget.
- `showTime(time : string) :void`: displays the current time on the screen when the user opens the clock widget.

User Class

Operations:

- `useWatch() : void`: User wears and uses the watch.

Setting Class

Operations:

- `saveDuration(durationalarm : int) : void`: Saves the alarm duration into the setting section.

- `saveTime(alarmtime : string) : void`: Saves the alarm time into the setting section.
- `saveInterval(alarminterval : int) : void`: Saves the alarm interval into the setting section.
- `saveRegion(timeregion : string): void`: Saves the time zone into the setting section.

Relationships of Classes:

- The `ClockSystem` class creates, knows, and uses the `Alarm` class and `Display` class.
- A `ClockSystem` relates to only one `Setting` class, `User` class, `TimeDatabase` class, and vice versa.

8.2.5 UI Prototype

Default UI

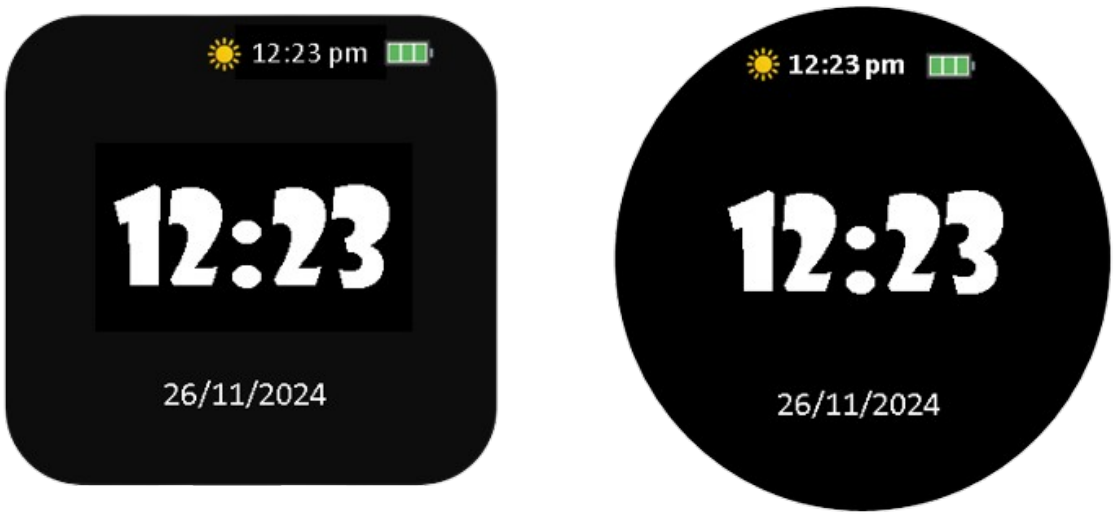

FIGURE 8.5: Time system clock UI prototype

The default UI of the Time System Application provides the user the date and time at that moment. To achieve that purpose, the features involve in will be:

- A small clock appears on the left corner of the UI, it is showed even on other applications for user to keep track of the time while working with the others.
- A big time on the middle of the clock when the user enters the app.
- Date is shown right under the time,

Alarm UI

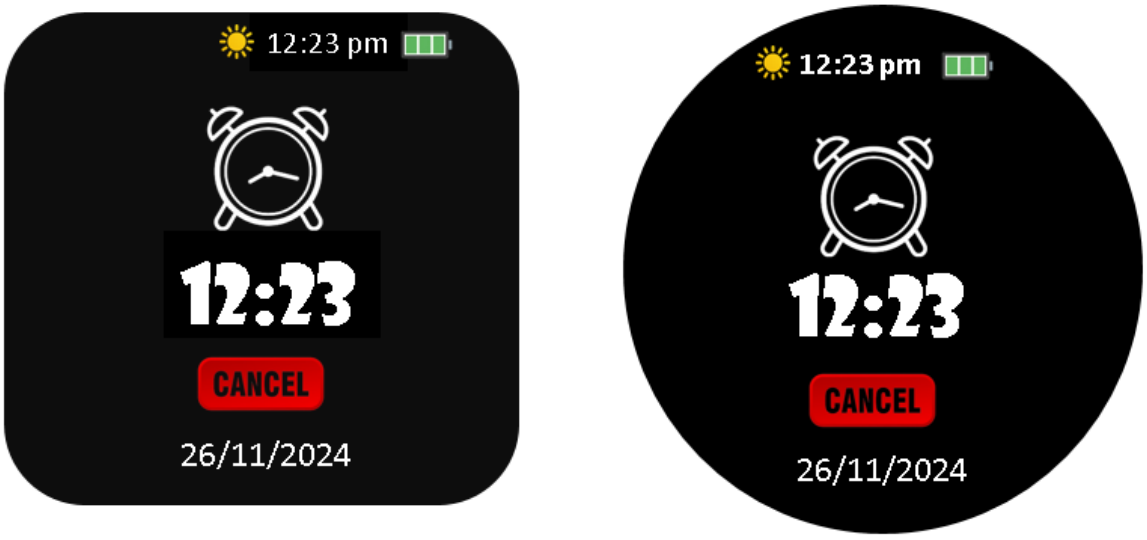

FIGURE 8.6: Time system alarm UI prototype

The Alarm UI will pop up when the time chosen before has met to notify the user. To accomplish the task, several features are used to built it:

- A clock icon to show that the alarm is up.
- A time to show the time that has been set before to be alarmed.
- A cancel button, the user can click onto it the alarm will stop.
- The date is shown as normal

There are also system that run beneath like the time interval check for enhancing the vibration, the vibration and ringing itself which can not be illustrated.

Chapter 9

Meeting Protocols and All Meetings

9.1 Meeting Protocols

The flow of each 15 minutes meeting will be like this:

1. Check attendance for every team members (1-2 minutes)
2. Leaders list out all pre-existing backlogs (1/2 - 1 minutes)
3. Each member choose which backlogs to work on for the week (5 minutes)
4. If there is a problem with the new task, all members discuss for a solution (5 minutes)
5. Leaders will record these down in trello board (1-2 minutes)

There are also certain rules that should be followed:

1. Stay focussed on the meeting
2. Always take notes of TODO tasks
3. Feel free to ask questions
4. Feel free to present your opinion
5. Respect other team members

9.2 Meeting I

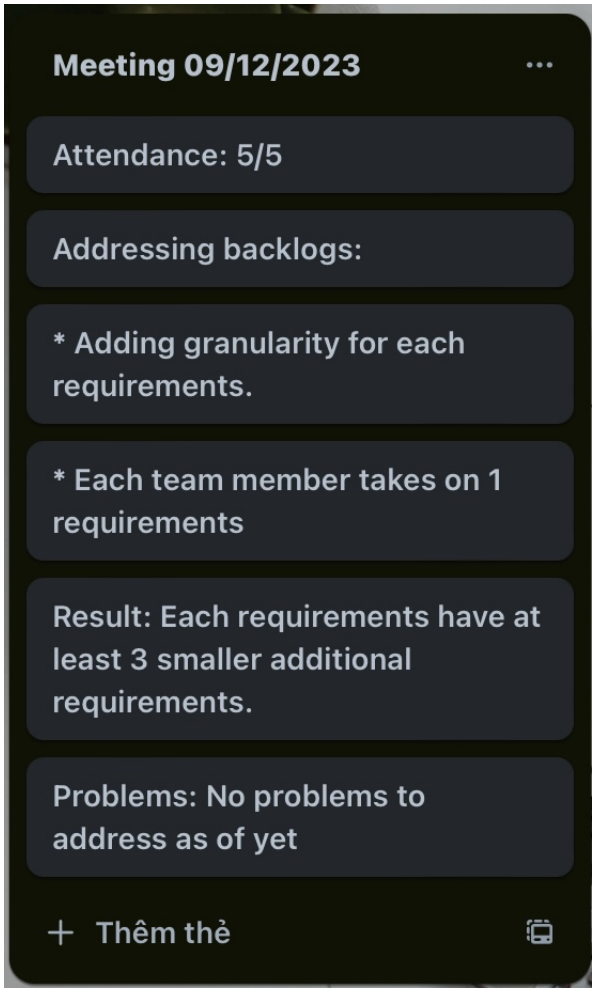

FIGURE 9.1: Meeting I

The first meeting covered these contents:

- The meeting was fully attended by every team members
- Each team members would take on 1 requirements to add granularity:
  - Dai Minh would work on Falling Alert
  - Nhut Thanh would work on Quick Payment
  - Anh Thu would work on Step Count and Exercise
  - Quy Nam would work on UV Exposure
  - Every team member will cooperate on additional 2 functions.
- After the works, all requirements had more than 3 smaller requirements
- Few minor problems relating to what was to be added but no problem worth taking notes
- Set target for the next week: draw use case diagram for all the requirements

9.3 Meeting II

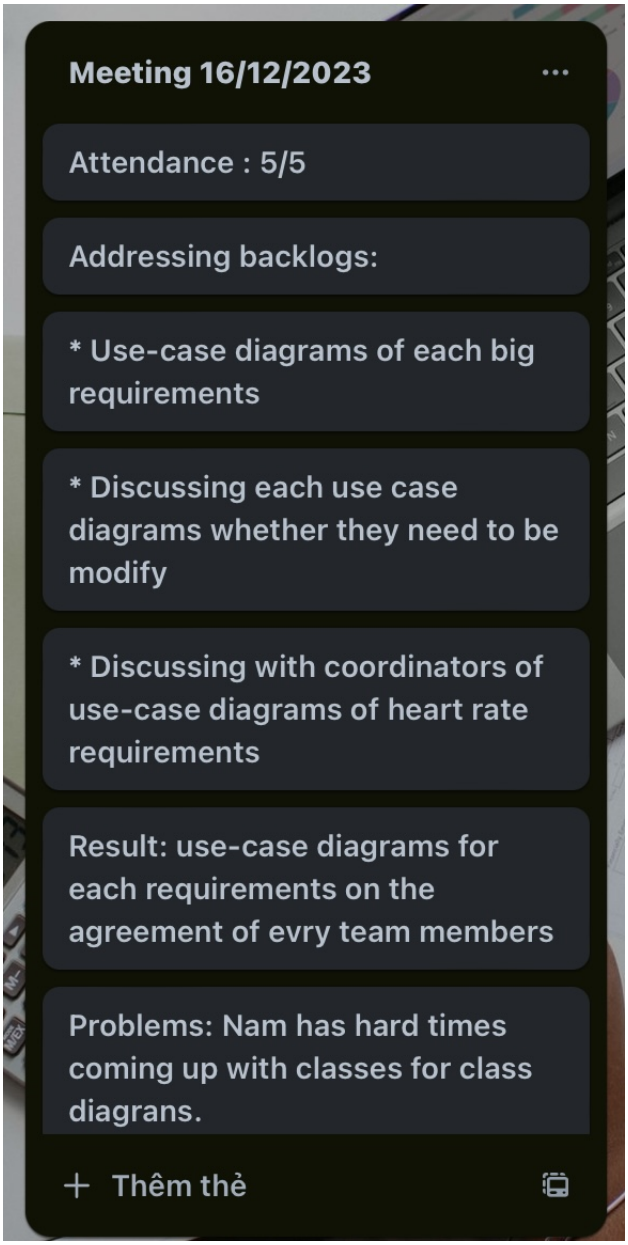

FIGURE 9.2: Meeting II

The second meeting covered these contents:

- The meeting was also attended by all members
- The task throughout the last week will be drawing all the use case diagram for all the available requirements and ensuring that each diagrams have more than 3 main use cases
- Throughout last the week, discuss with the professors on the various use case diagram
- Fixing and modifying the diagrams throughout the week, in particular the heart rate diagrams, removing redundant use case and adding some setter function

- After the week, all the use case diagrams were ready
- Nam have a hard time coming up with some use case for the UV exposure diagram so we add an additional 10 minutes brainstorm section
- Set target for the next week: draw class diagram for all the requirements

9.4 Meeting III

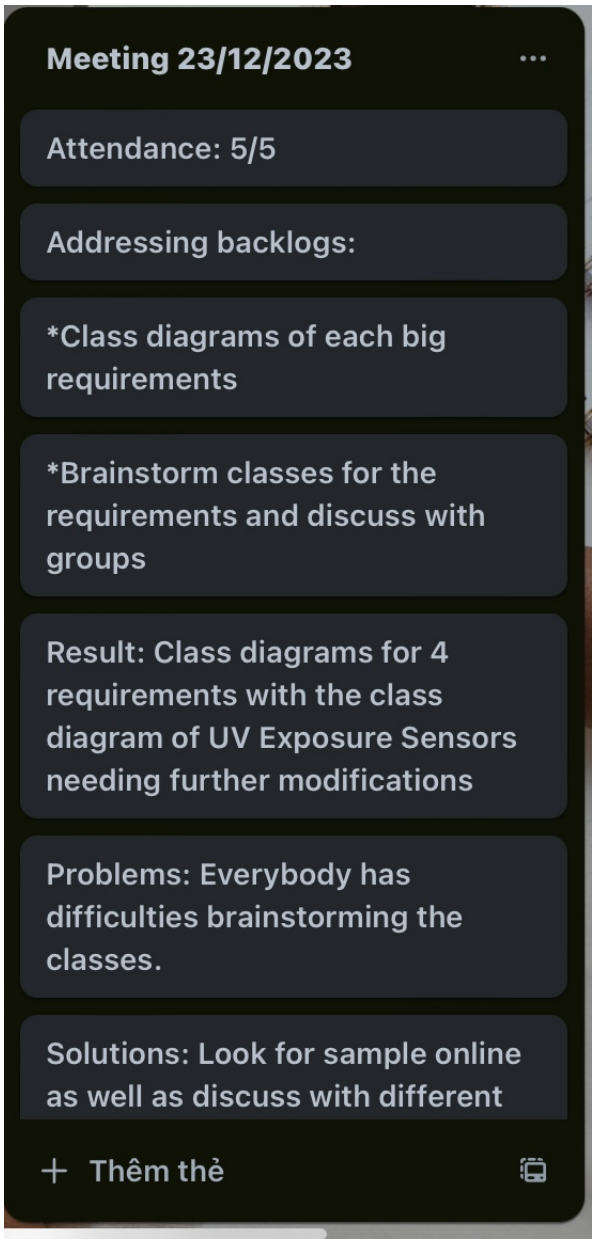

FIGURE 9.3: Meeting III

The third meeting covered these contents:

- The team fully attended the meeting
- The last week task was to create class diagrams for each of the current requirements, ensuring that each of the class diagram have to have at least 5 or more

classes as well as implement fully the function that appeared in the use case diagrams

- Throughout the weeks, the team had brainstormed classes and also discussed the classes with the whole team for everyone opinion
- All class diagrams were fairly close to finished with the exception of the UV exposure sensor class diagram, both having trouble understanding the structure of the class diagrams.
- All team members have minor inconveniences regarding the classes and we add an additional 5 minutes session for the scrum master to explain the structure of the class and how classes should be form based on the use case diagrams as well as looking for inspiration from online source
- Set target for next week: draw sequence diagram for all requirements and polished the class diagram

9.5 Meeting IV

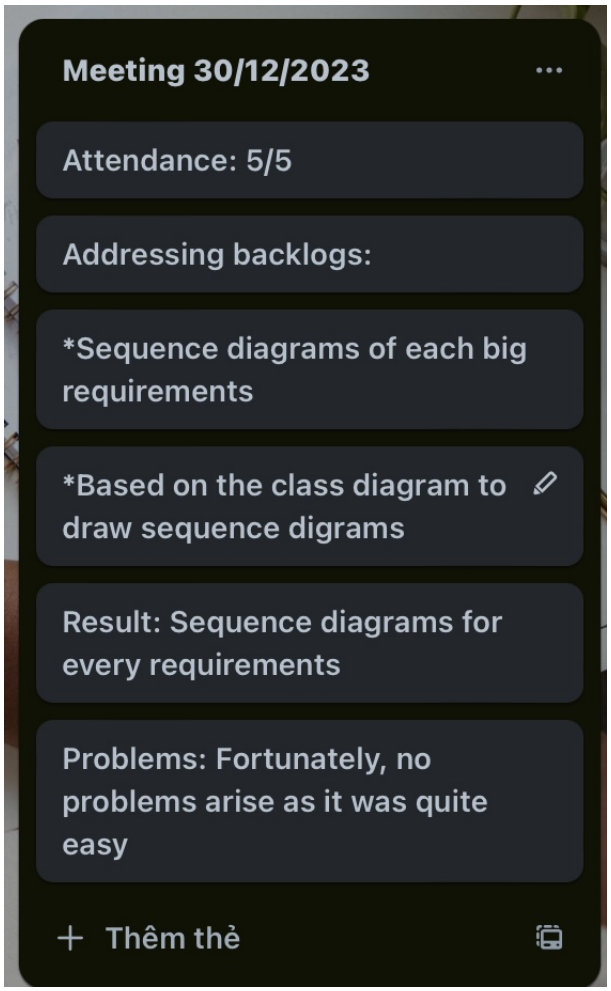

FIGURE 9.4: Meeting IV

The fourth meeting covered these contents:

- The meeting was fully attended by all members

- The last week task was to create sequence diagrams for every requirements, making sure that the sequence diagrams are correctly implemented based on the class diagrams and polished the class diagrams
- In the week, members have discussed with each others on their opinion of the class diagrams to further complete and modify them, members also finished the sequence diagrams for the requirements.
- Fortunately, there were no real problems worth mentioning when it comes to drawing the sequence diagrams
- Set target for next week: draw activity diagrams for each requirements

9.6 Meeting V

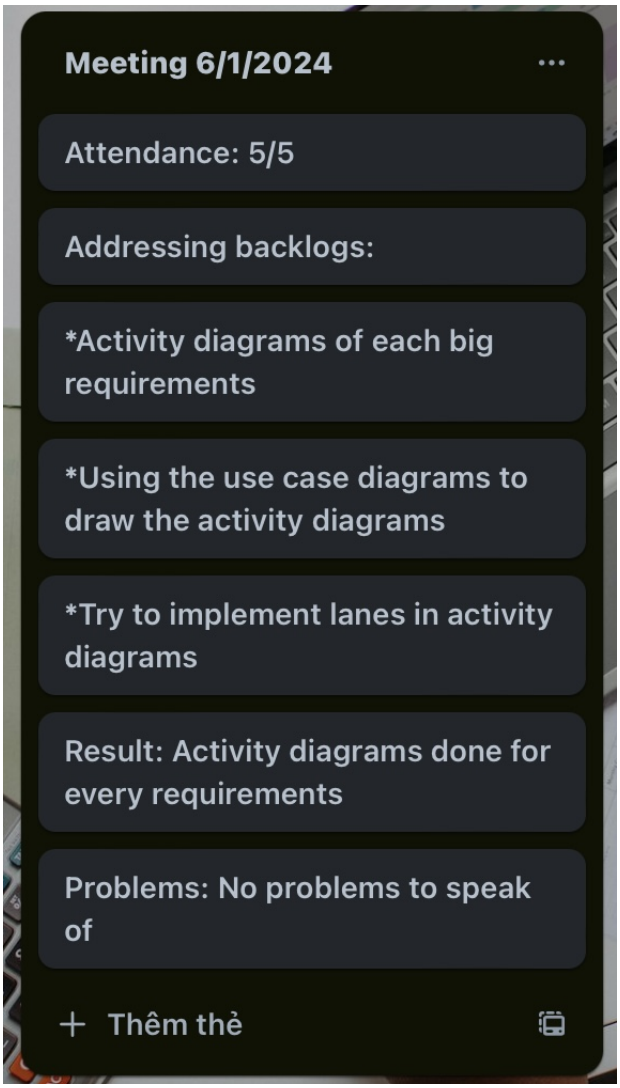

FIGURE 9.5: Meeting V

The fifth meeting covered these contents:

- The meeting was fully attended by the team
- The last week task was to create activity diagrams for every available requirements.

- In the week, members use the use case diagrams as a base to think of activity flows for the activity diagrams and also create those diagrams as well. Activity diagrams that was to be created have to have at least 1 branch or more to make the diagrams more details.
- The task was relatively simple so no problems appears as every team members have gotten used to these diagrams from lectures and exercises.
- Final week's target: write description for all diagrams and the leader finish proofreading the report

9.7 Meeting VI

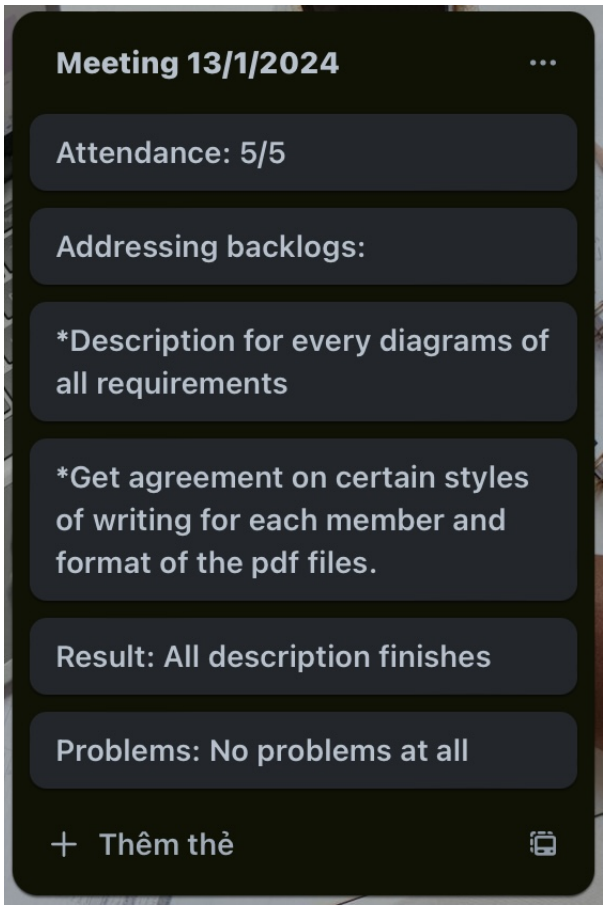

FIGURE 9.6: Meeting VI

The final meeting covered these contents:

- The team fully attended the final meeting
- The last task for every team members was to finish writing description for the final report
- Team members all settle for different styles of writing to increases diversity
- Description for the diagrams can be long or short but it must be understandable and comprehensible for the reader
- Team members also add additional UI prototype as a side tasks

- Scrum Master will continually proofread and modify the report until the report is due to submission

9.8 Work assignment table

Name	Work Assignment
(Developer) Phan Khanh Nghi	• Write all Chapter 2 (100%) • Write description and draw UI Chapter 7 (15%) • Draw UI, write descriptions for Chapter 8 (15%) • Write chapter 1 (20%) • Write chapter 10 (100%)
(Scrum Master) Nguyen Dai Minh	• Arrange tasks for team members • Set up and document progress on trello • Write all chapter 6 (100%) • Write all chapter 9 (100%) • Draw diagrams chapter 7 (40%) • Write chapter 1 (30%) • Draw diagrams chapter 8 (40%)
(Developer) Nguyen Nhut Thanh	• Write all chapter 3 (100%) • Write description chapter 7 (15%) • Write description chapter 8 (15%) • Write chapter 1 (15%)
(Developer) Pham Nguyen Quy Nam	• Write all chapter 4 (100%) • Write description chapter 7 (15%) • Write description chapter 8 (15%) • Write chapter 1 (15%)
(Developer) Phan Huynh Anh Thu	• Write all chapter 5 (100%) • Write chapter 1 (20%) • Write description chapter 7 (15%) • Write description chapter 8 (15%)

Chapter 10

Project Summary

10.1 Learning Experience

Working on this smartwatch project has given our team the chance to go deeply into software development analysis, create UML diagrams, and work with the SCRUM model, which has given me an in-depth understanding. Using Magic Systems with Systems Architect software enhances our understanding of software analysis tools and how to translate concepts into practical solutions. In addition, our smartwatch project presented us with several challenges that improved our problem-solving skills and expanded our understanding of the difficulties associated with software development in practice. Throughout this project, our teamwork skills also developed. We learned how to work well together, compensate for one another's drawbacks, and build on each other's strengths. Furthermore, our group gained a variety of insights, expertise, and experiences from our smartwatch project development.

10.2 Future Development

Potential growth:

Phone notifications recieving:

Additional functionality of this watch is the ability to receive phone notifications. This is helpful since, in addition to operating independently, our smartwatch can link to other devices, such smartphones, to get notifications such as SMS messages and phone calls.

IoT device controling:

Using this capability, users can operate other IoT devices like the thermostat, smart bridge, and television from a distance.

Multiple key:

This function enables users to enter automobile doors, safe deposit boxes, and house doors using their watch as a key.

UI design and smart watch design improvement

As users will be interacting with the user interface directly, it is essential that it be improved. An interface that is simple to use and practical to use will probably enhance the smartwatch's capacity to process user data. The initial stage in upgrading the user interface should be to update design software and design-related expertise.

The design of the watch is also important, the person will have to wear it every day. In order for people to wear a watch for all activities and weather situations, it should feel robust, fashionable and comfortable to wear. The watch should be made of materials that are safe for the user's health and the environment. In the future, we will learn about hardware, material design knowledge and to create a good design.