Design

Use Cases

Use Case 1: Create Profile

Primary Actor(s): RaDoTech app user.

Stakeholders:
Users - want to create their own profile to track health measures from scans.

Pre-condition(s):
User is logged in and currently in the app's "Main menu" page.

Success guarantee(s):
The user(s) can see their newly-added profile in the list on the app's "Profiles" page.

Main success scenario:

1. User clicks the "Profiles" option.
2. User is moved to the "Profiles" page.
3. User clicks the "Add Profile" button to add a new profile.
4. User enters data in fields representing their Profile and clicks the "Save and continue" button.
5. App checks that no profile exists with the full name entered.
6. New profile gets created and stored by the app.
7. User is prompted back to the "Profiles" page.

Extensions:
1a. User clicks another option by mistake.
    1a1. The user can return to the "Main menu" page by clicking on a "Back" button.
5a. Profile with the entered name for the new one is found.
    5a1. App prevents account creation. A "Validating Error" pop-up appears, informing users that Profile full name should be unique. They click the "Okay" button within the pop-up and alter the profile's name before proceeding to change their full name to something unique.

Use Case 2: Update Profile

Primary Actor(s): RaDoTech app user.

Stakeholders:
Users - want to change their profile information.

Pre-condition(s):
User is logged in and currently in the app’s “Main menu” page.

Success guarantee(s) (equivalently Post-conditions)
User(s) can open their augmented profile within “Profiles” to see changes persist, and upon taking scans will be able to see the augmented profile among profile options.

Main success scenario:
1. User clicks the “Profiles” option.
2. User is moved to the “Profiles” page.
3. User presses the profile they wish to alter.
4. The app moves them to their Profile’s page, which contains all pre-existing information loaded into its fields, empty if none was provided.
5. User clicks on any input text field.
6. User alters the value stored in the field.
7. User returns to their Profile’s page.
8. User clicks the “Save” button.
9. A “Success” pop-up appears in the app, informing the user that Profile is saved.
10. App returns the user to the “Profiles” page.

Extensions:
8a. User didn’t click “Save and continue”, but instead pressed the “Back” button.
    8a1. No changes are saved to the corresponding profile.

Use Case 3: Delete Profile

Primary Actor(s): RaDoTech app user.

Stakeholders:
Users - want to change their profile information.

Pre-condition(s):
User is logged in and currently in the app’s “Main menu” page.

Success guarantee(s) (equivalently Post-conditions)
The RaDoTech app no longer stores the Profile deleted, so user(s) can no longer see it within the list on the “Profiles” page, or when selecting the profile for scans.

Main success scenario:
1. User clicks the “Profiles” option.
2. User is moved to the “Profiles” page.
3. User selects the profile they wish to remove.
4. User clicks on the “Delete profile” button.
5. App removes the profile, and its associated snapshots.

6. A “Success” pop-up appears in the app, letting the user know the profile has been deleted.
7. App returns the user to the “Profiles” page, where the corresponding profile disappeared.

Extensions: None.

Use Case 4: Scan Health Hata

Primary Actor(s): RaDoTech users

Stakeholders:
Users - want to scan their health data using the RaDoTech device and save it to analyze their health metrics.
RaDoTech - offers an accessible app for users providing them with various features to track their own health condition.

Pre-condition(s):
A user is logged-in in the RaDoTech app.

Success guarantee(s):
The user can access their newly-scanned health data on their “History” page to track their health data.

Main success scenario:
1. User takes off jewelry, earrings, etc that could interfere with the scan.
2. User clicks “Measure now” in the app’s “Main menu” page.
3. The app presents a drop-down menu with profiles that the user can choose for the scan.
4. The user chooses one profile for their scan.
5. The app asks the user to scan the target point with the scanner device.
6. The app waits for the scanner to make contact with the user's skin.
7. “Skin on/off” Sub Use Case occurs.
8. User repeats steps (5) to (7) for the remainder of (23) scans.
9. User gets moved to the “Notes” page once the scan is complete.
10. User adds some textual description of how they were feeling to the “Notes” input text box, or input data inside other fields on the page.
11. User clicks the “Save” button.
12. An “Attention” pop-up appears on the app, warning users they can specify all fields except tags and notes only once.
13. The user clicks the “Okay” button on the “Attention” pop-up to close it.
14. A “Success” pop-up appears in the app, informing users the scan is complete.
15. User clicks the “Okay” button on the “Success” pop-up to close it.

16. The app saves all the input data into the history of snapshots.
17. The app returns the user to their “Main menu” page.

Extensions:
4a. The user has multiple profiles on the account they’re logged on, but at least one belongs to them.
  4a1. The user chooses their own profile.
4b. User has a profile, but none belongs to them.
  4b1. User cancels the current scan using the “Back” button, so they return to the “Main menu” page. View “Create profile” Use Case.
8a. User clicks “Stop” anytime during the scanning steps.
  8a1. The user is returned to the “Main menu” page and all scanned data is reset.
15a. User doesn’t click the “Okay” button.
  15a1. Pop-up remains open until the user clicks the “Okay” button to confirm they understand the snapshot of scans is done.

Use Case 4a: Skin on/off Sub Use Case (Safety)

Primary Actor(s):
RaDoTech users and the RaDoTech medical device.

Stakeholders:
RaDoTech medical device - used by users to get health readings by skin contact.
Users - use the RaDoTech device for scanning health data to upload to the RaDoTech app.
RaDoTech app - gets readings from the RaDoTech medical device scans.

Pre-condition(s):
The user has chosen a profile for their scan and is now in the scanning stage.

Success guarantee(s) (equivalently Post-conditions)
The medical device uploads the scanned user’s reading to the app and the app resumes with its operation.

Main success scenario:
1. User presses the medical device gently on their skin to scan the current point.
2. Device scans the current point and uploads it to the RaDoTech app.
3. User immediately takes the device off their skin.

Extensions:
1a. Medical device’s node is too far from the skin.
  1a1. The RaDoTech app waits for scan reading to be initiated by the medical device.
2a. Skin contact is lost during the current scan.

2a1. The app then displays a warning pop-up to let the user know to press the medical device properly on their skin, and re-attempts scanning the same point.

3a. User does not take the device off in time for the next scan.

3a1. Device may get an inaccurate reading for the next scan.

Use Case 5: View Historical data

Primary Actor(s): RaDoTech app user.

Stakeholders:
Users - want to be able to view historical data related to their scans to analyze their health condition.

Pre-condition(s):
The user has completed at least one past scan on their corresponding profile. They are currently on the app's "Main menu" page.

Success guarantee(s)
The user has viewed the body, chart and circle components of their historical data to gain understanding of their health, and can move on to other activities.

Main success scenario:

1. User clicks on the "History" option from the "Main menu" page.
2. App moves the user to the "History" page, displaying a table with multiple rows (timestamp of the snapshot, the Profile's full name and snapshot notes) of snapshots of scans from all profiles.
3. User clicks on any scan from the list.
4. App moves the user to the Metering Results page's "Body" view, an overall body shot with which organs are above normal, normal, or below normal.
5. User clicks on the "Chart" view option to view the scanned data as a chart, to understand meaning at organ level.
6. User can click on other options for more options related to their scans.
7. User clicks the "Back" button to return to the "Main menu" page.

Extensions:

6a. User clicks on the "Indicators" option in the footer.

6a1. Refer to "View Indicators Sub Use Case".

6c. User clicks on the "Recommendations" option in the footer.

6c1. Refer to "View Recommendations Sub Use Case".

Use Case 5a: View Indicators Sub Use Case

Primary Actor(s): RaDoTech app user.

Stakeholders:
Users - want to be able to view Indicators within historical data related to their scans to analyze their health condition.

Pre-condition(s):
The user has completed at least one past scan on their corresponding profile. They are currently on the app's “Metering results” page for one of their snapshots of scans.

Success guarantee(s)
The user has viewed the Indicators section for their scan data to get diagnostic results from the app.

Main success scenario:

1. User clicks on the “Indicators” button in footers of the “Metering results” page.
2. App displays a screen with the user's diagnosed Energy level, Immune system, Metabolism, Psycho-emotional state, and Musculoskeletal system, as well as an “Average value” table as a snapshot of how the user's systems are overly functioning.
3. User analyzes the diagnostics to understand their health condition.

Extensions: None.

Use Case 6: View Recommendations Sub Use Case

Primary Actor(s): RaDoTech app user and RaDoTech company.

Stakeholders:
Users - want to be able to view Indicators within historical data related to their scans to analyze their health condition.
RaDoTech advisors - legal professional advisors able to provide users with reliable information and advice based on their metering results.
RaDoTech website - allows users to buy supplements based on the recommendations from their scan data.

Pre-condition(s):
The user has completed at least one past scan on their corresponding profile. They are currently on the app's “Metering results” page for one of their snapshots of scans.

Success guarantee(s)
The user has viewed the Recommendation section for their scan data to get diagnostic results from the app.

Main success scenario:
1. User clicks on the “Recommendation” button in footers on the “Metering results” page
2. App displays a page with some cool insight into their scanned data. It displays textual information going in depth with practical and actionable tips.
3. User uses the recommendations to improve their health.
4. User presses the “Back” button to return to the “Body view” page.

Extensions: None.

Use Case 7: Device out of Battery

Primary Actor(s):
RaDoTech users and the RaDoTech medical device.

Stakeholders:
RaDoTech medical device - runs out of battery as its running and being used by users.
Users - use the RaDoTech device for scanning health data uploaded to the RaDoTech app.
RaDoTech app - requires the RaDoTech medical device to allow users to scan and store their health data.

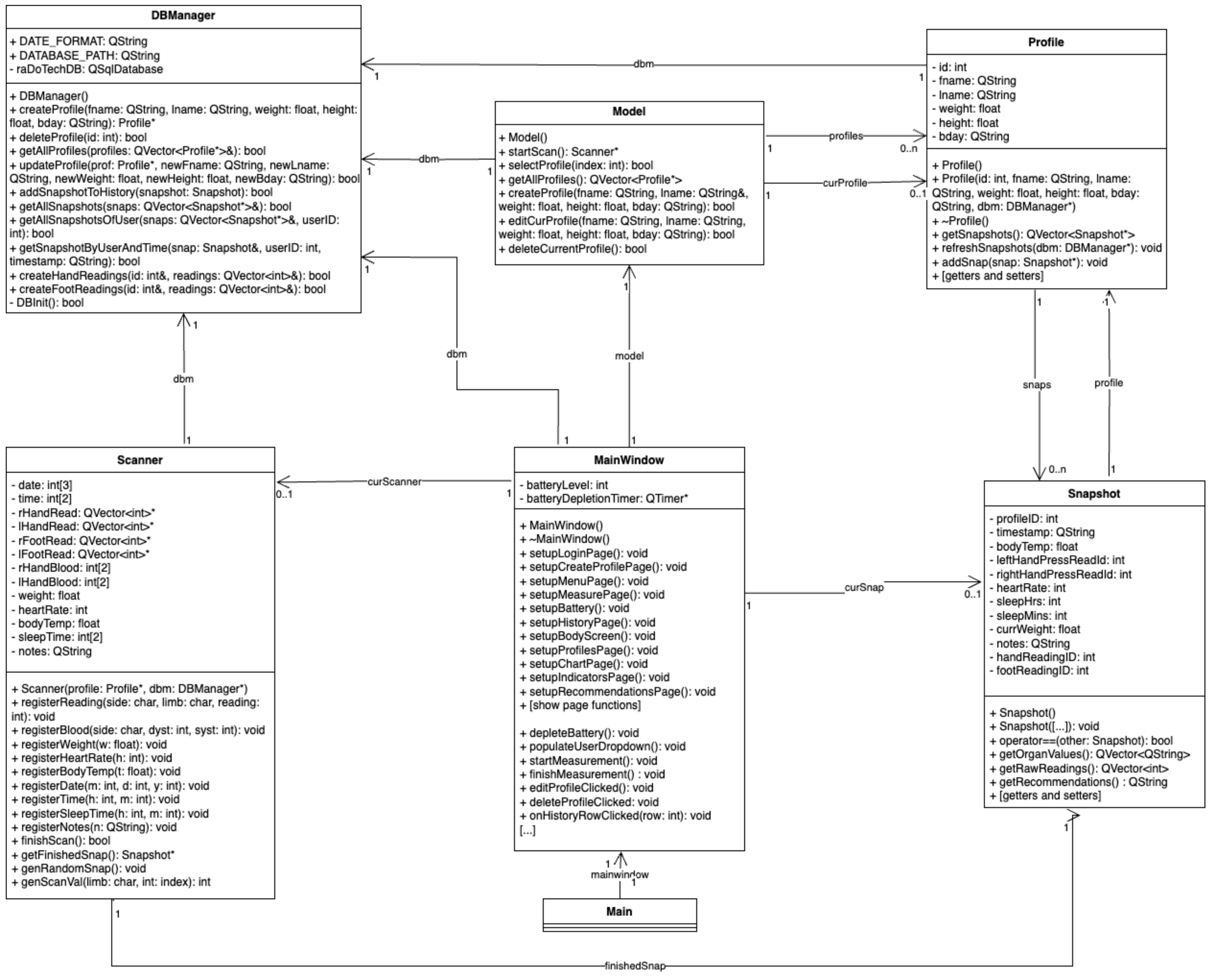
Pre-condition(s):
The battery has been used up to the point it reached 0%.

Success guarantee(s)
The device has gracefully shut down, and users can recharge it to continue using it for scanning.

Main success scenario:
1. User uses the medical device until it reaches 0% Battery Power.
2. A pop up appears in the app, letting the user know the Device is about to turn off.
3. Medical Device disconnects from the app.
4. Medical Device has fully powered off.

Extensions:
3a. User was currently in the middle of a scan.
    3a1. The app returns the user to the “Main menu” page and resets the scan.
    3a2. User recharges the medical device before doing any further scans.

UML Class Diagram


DBManager
+ DATE_FORMAT: QString
+ DATABASE_PATH: QString
- raDoTechDB: QSqlDatabase
+ DBManager()
+ createProfile(fname: QString, lname: QString, weight: float, height: float, bday: QString): Profile*
+ deleteProfile(id: int): bool
+ getAllProfiles(profiles: QVector<Profile*>&): bool
+ updateProfile(prof: Profile*, newFname: QString, newLname: QString, newWeight: float, newHeight: float, newBday: QString): bool
+ addSnapshotToHistory(snapshot: Snapshot): bool
+ getAllSnapshots(snaps: QVector<Snapshot*>&): bool
+ getAllSnapshotsOfUser(snaps: QVector<Snapshot*>&, userID: int): bool
+ getSnapshotByUserAndTime(snap: Snapshot&, userID: int, timestamp: QString): bool
+ createHandReadings(id: int&, readings: QVector<int>&): bool
+ createFootReadings(id: int&, readings: QVector<int>&): bool
- DBInit(): bool
Model
+ Model()
+ startScan(): Scanner*
+ selectProfile(index: int): bool
+ getAllProfiles(): QVector<Profile*>
+ createProfile(fname: QString, lname: QString&, weight: float, height: float, bday: QString): bool
+ editCurProfile(fname: QString, lname: QString, weight: float, height: float, bday: QString): bool
+ deleteCurrentProfile(): bool
Profile
- id: int
- fname: QString
- lname: QString
- weight: float
- height: float
- bday: QString
+ Profile()
+ Profile(id: int, fname: QString, lname: QString, weight: float, height: float, bday: QString, dbm: DBManager*)
+ ~Profile()
+ getSnapshots(): QVector<Snapshot*>
+ refreshSnapshots(dbm: DBManager*): void
+ addSnap(snap: Snapshot*): void
+ [getters and setters]
Scanner
- date: int[3]
- time: int[2]
- rHandRead: QVector<int>*
- lHandRead: QVector<int>*
- rFootRead: QVector<int>*
- lFootRead: QVector<int>*
- rHandBlood: int[2]
- lHandBlood: int[2]
- weight: float
- heartRate: int
- bodyTemp: float
- sleepTime: int[2]
- notes: QString
+ Scanner(profile: Profile*, dbm: DBManager*)
+ registerReading(side: char, limb: char, reading: int): void
+ registerBlood(side: char, dyst: int, syst: int): void
+ registerWeight(w: float): void
+ registerHeartRate(h: int): void
+ registerBodyTemp(t: float): void
+ registerDate(m: int, d: int, y: int): void
+ registerTime(h: int, m: int): void
+ registerSleepTime(h: int, m: int): void
+ registerNotes(n: QString): void
+ finishScan(): bool
+ getFinishedSnap(): Snapshot*
+ genRandomSnap(): void
+ genScanVal(limb: char, int: index): int
MainWindow
- batteryLevel: int
- batteryDepletionTimer: QTimer*
+ MainWindow()
+ ~MainWindow()
+ setupLoginPage(): void
+ setupCreateProfilePage(): void
+ setupMenuPage(): void
+ setupMeasurePage(): void
+ setupBattery(): void
+ setupHistoryPage(): void
+ setupBodyScreen(): void
+ setupProfilesPage(): void
+ setupChartPage(): void
+ setupIndicatorsPage(): void
+ setupRecommendationsPage(): void
+ [show page functions]
+ depleteBattery(): void
+ populateUserDropdown(): void
+ startMeasurement(): void
+ finishMeasurement() : void
+ editProfileClicked(): void
+ deleteProfileClicked(): void
+ onHistoryRowClicked(row: int): void
[...]
Snapshot
- profileID: int
- timestamp: QString
- bodyTemp: float
- leftHandPressReadId: int
- rightHandPressReadId: int
- heartRate: int
- sleepHrs: int
- sleepMins: int
- currWeight: float
- notes: QString
- handReadingID: int
- footReadingID: int
+ Snapshot()
+ Snapshot([...]): void
+ operator==(other: Snapshot): bool
+ getOrganValues(): QVector<QString>
+ getRawReadings(): QVector<int>
+ getRecommendations() : QString
+ [getters and setters]
Main
dbm
profiles
curProfile
model
snaps
profile
curScanner
curSnap
mainwindow
finishedSnap
1
0..n
0..1

Classes:
MainWindow - general UI, responsible for overall state. Access model and scanner frequently to update the state of data, but rarely directly modifies things, preferring to use the intermediaries. Includes several screens (see demo) for various use cases.

DBManager - SQL database for storing profiles and snapshots. Provides functions to easily access and modify the database in specifically designed ways.

Scanner - used to stage a scan so that it can be added to the database, then is the function to generate resultant snapshot objects. Used directly by mainwindow.

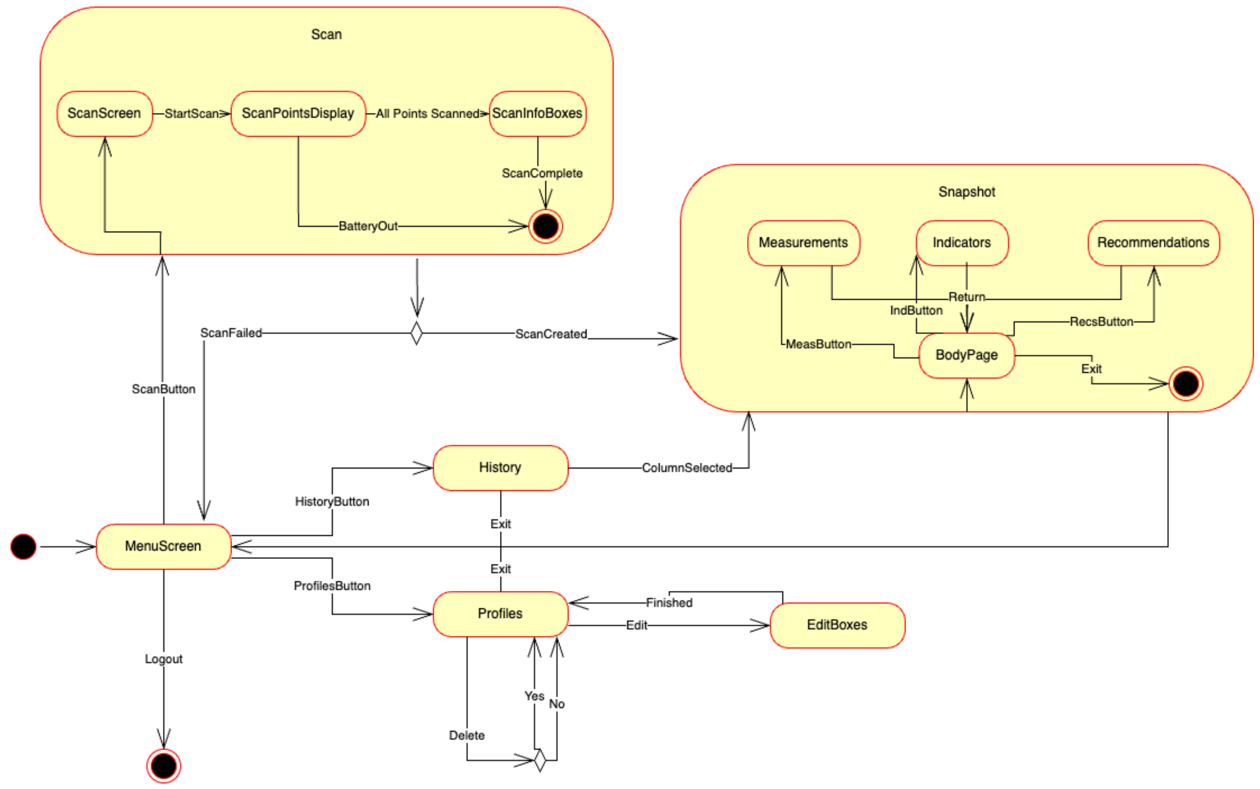
Model - Container class which provides access to certain specific functionality for the mainwindow while abstracting it. This is the class which stores all the profiles - and thus, indirectly, all the snapshots.

Profile - Contains user data and snapshots. Relatively simple; uses many getters and setters.

Snapshot - stores data for a single scan. On top of getters and setters for raw data, includes limited functionality for processing data and returning meaningful results.

Additional Notes:
The project mostly follows an MVC paradigm. Due to the nature of QT, view and controller have somewhat blended together, but the model remains wholly distinct, handling all behaviour requiring the back-end, such as transactions with the DB through the DBManager, and the individual objects representing the system state occupy their own appropriate space.

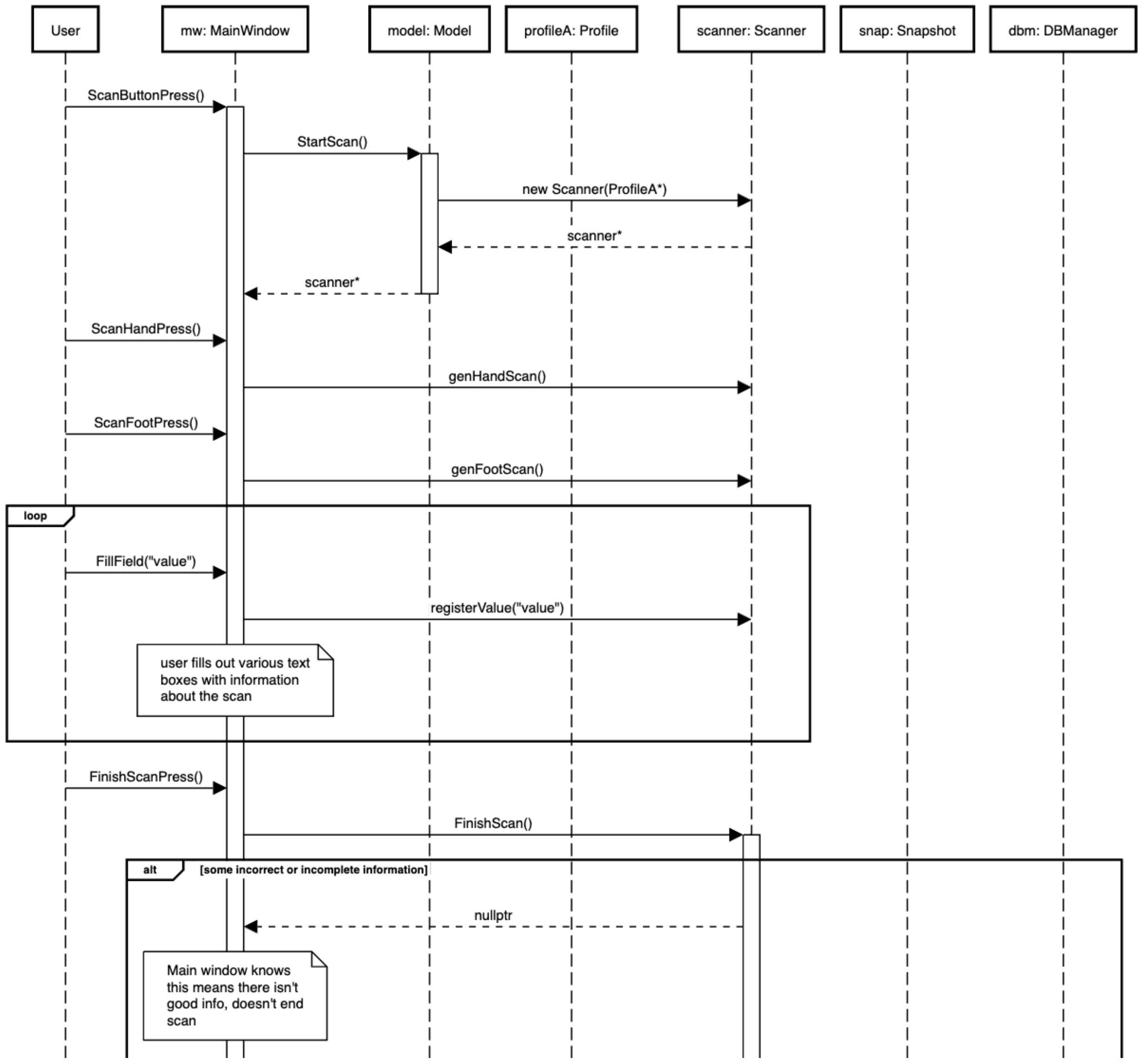
State Machine Diagram

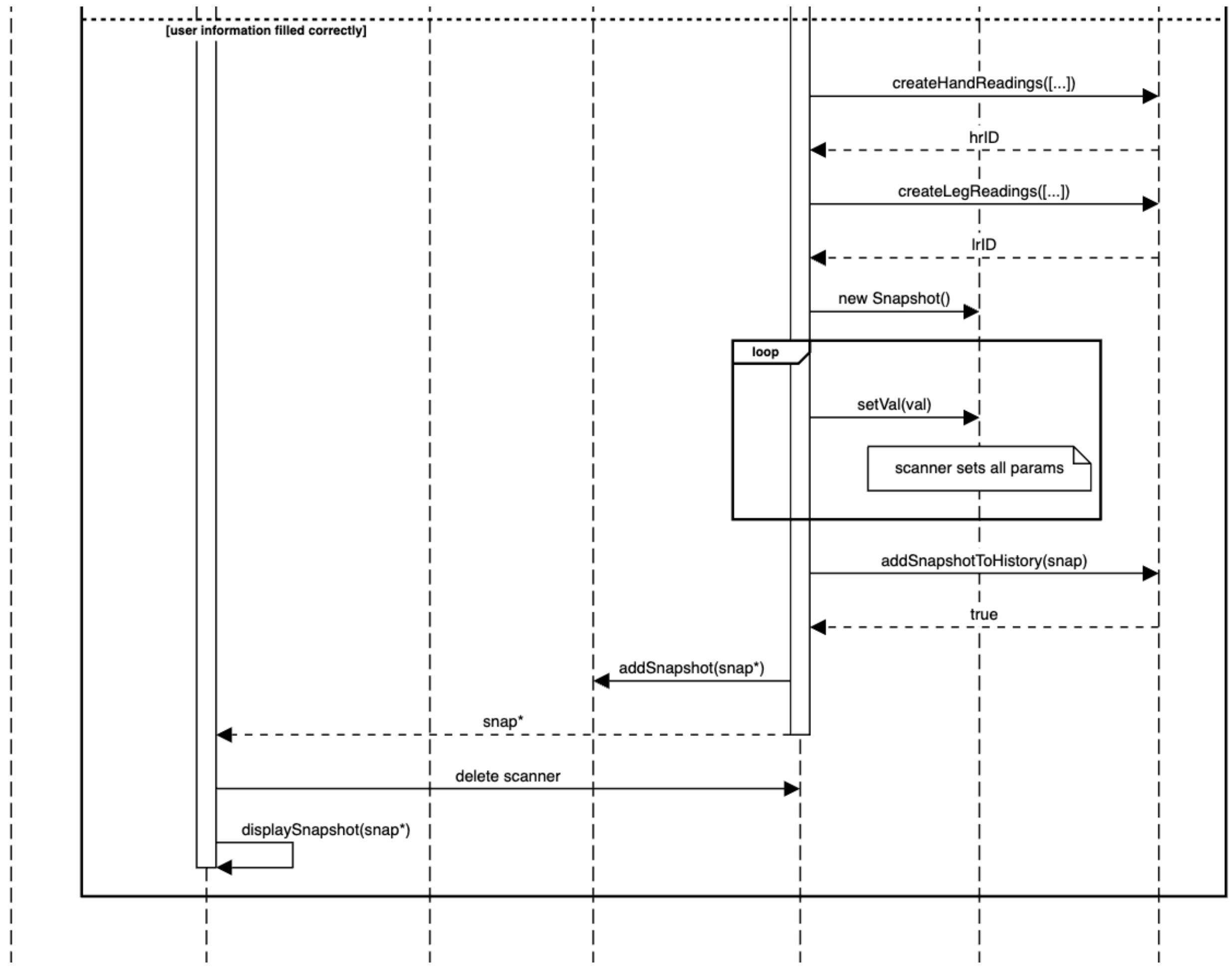


Notes:
Snowcases state machine for GUI and internal processes.

Sequence Diagrams

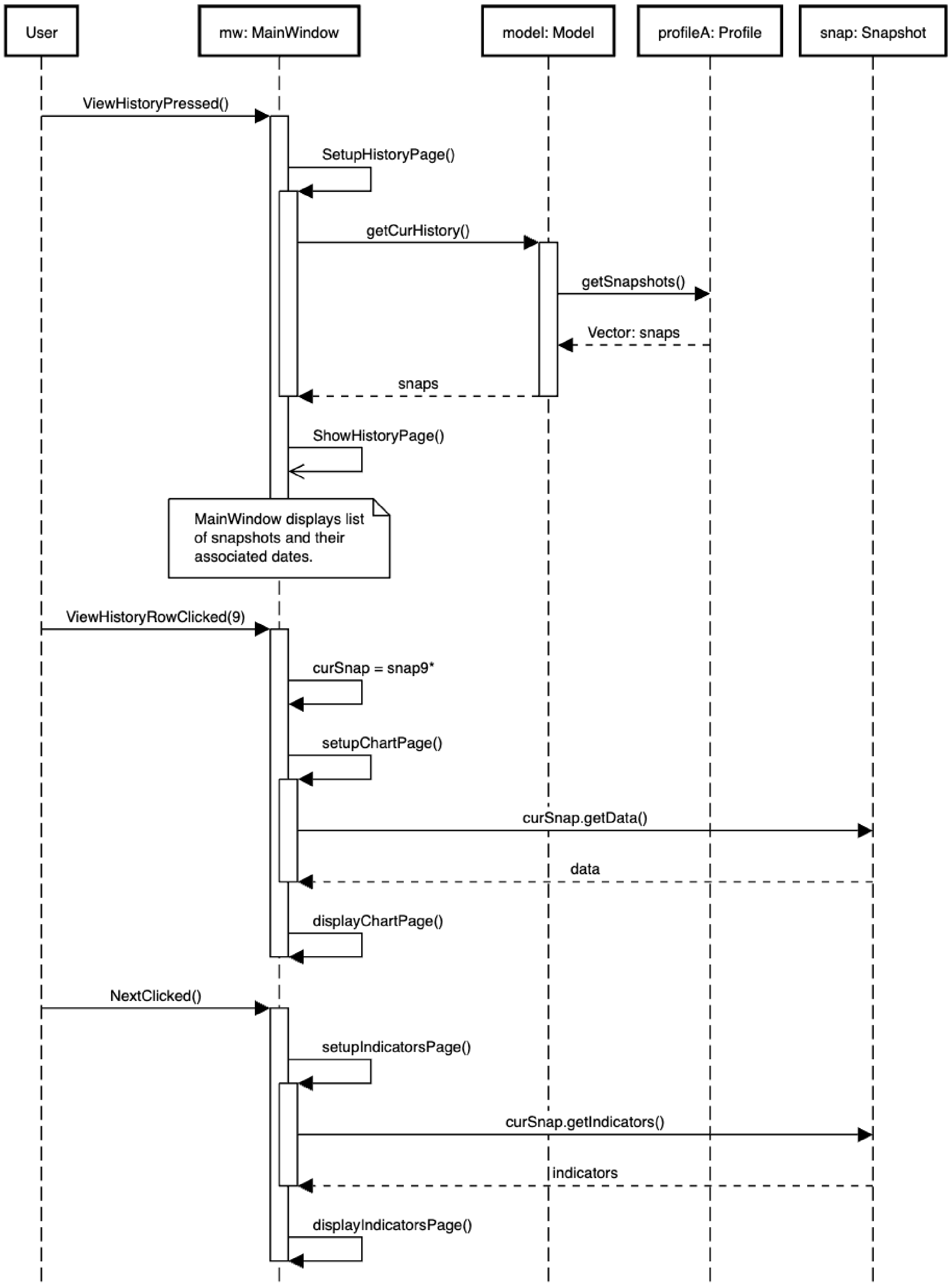
Success 1: User Performs Scan

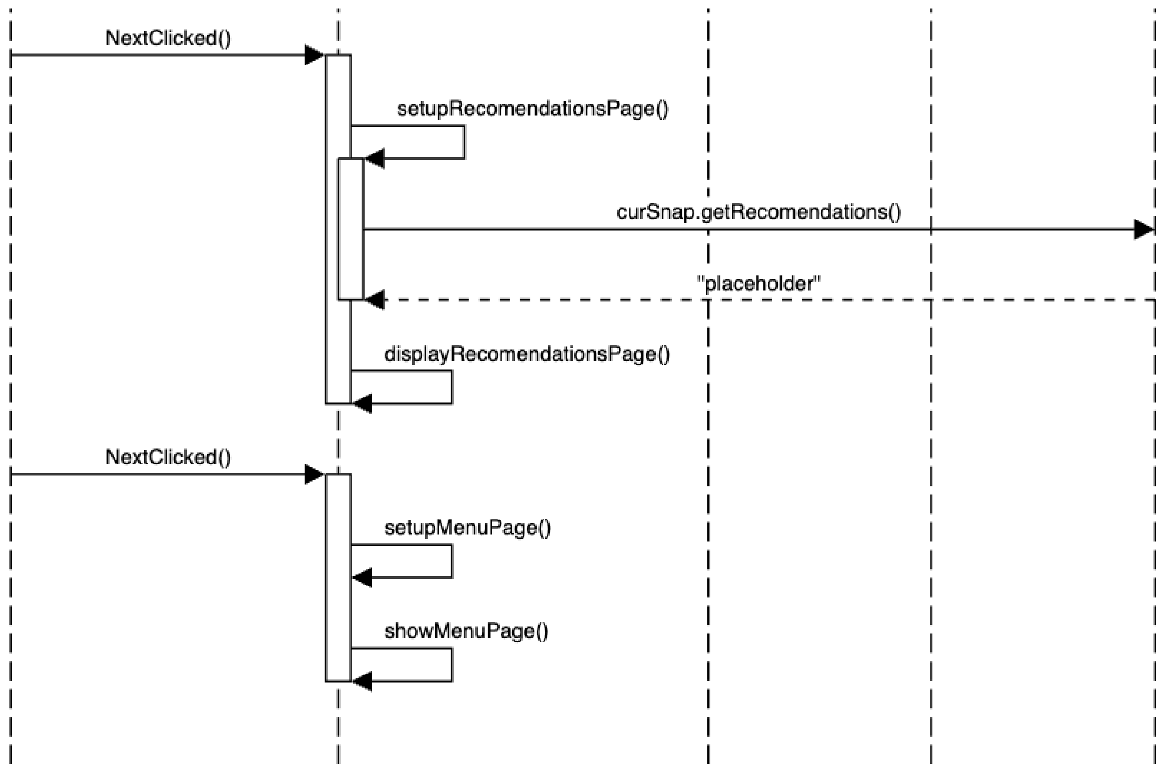




Notes:
Sequence Diagram 1 showcases the standard procedure of a user performing a scan and the system's storage of it in the database. It does not include the MainWindow's actual methods of displaying the scan, which is covered in some detail in Success Scenario 2.

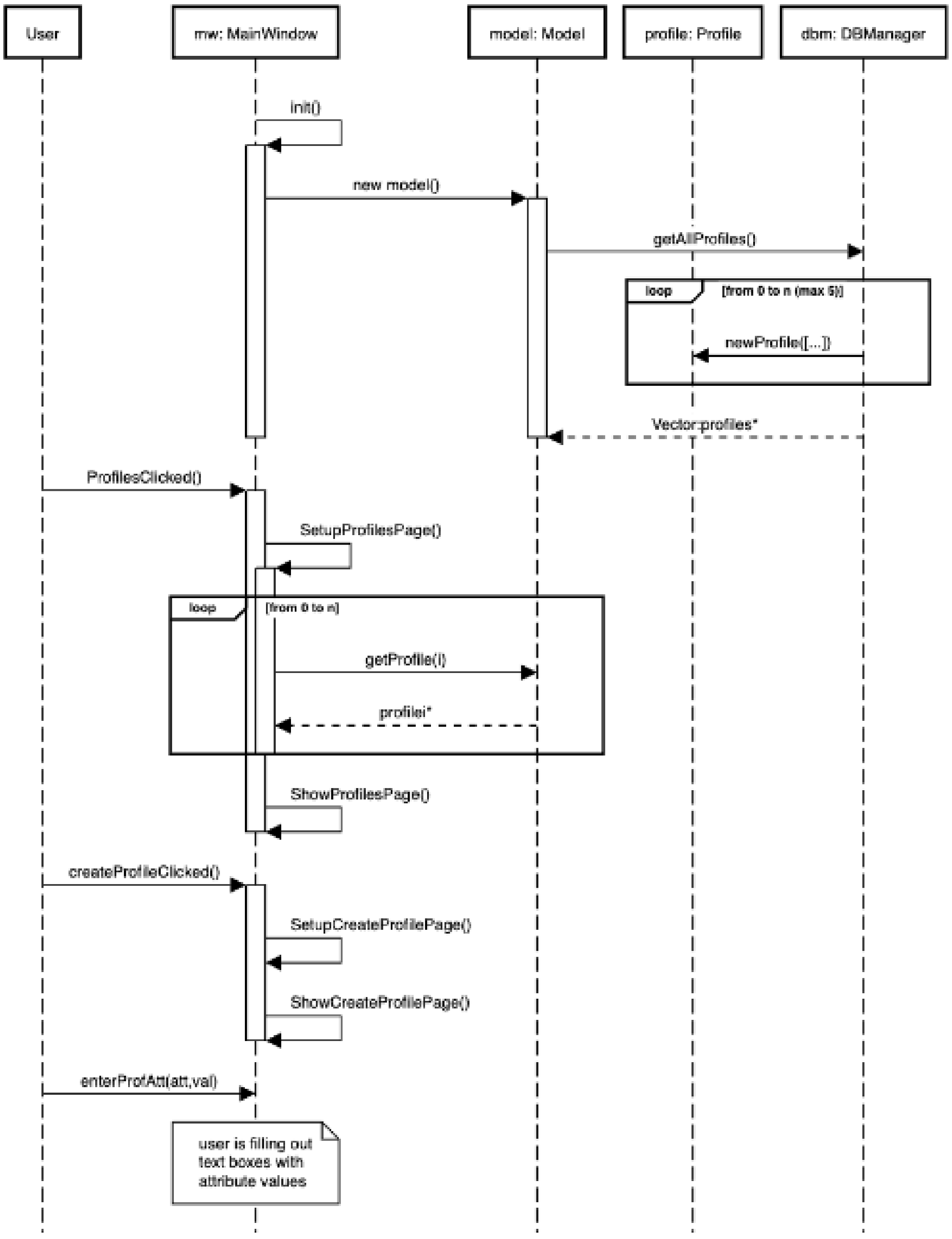
Success 2: User Views Old Scan

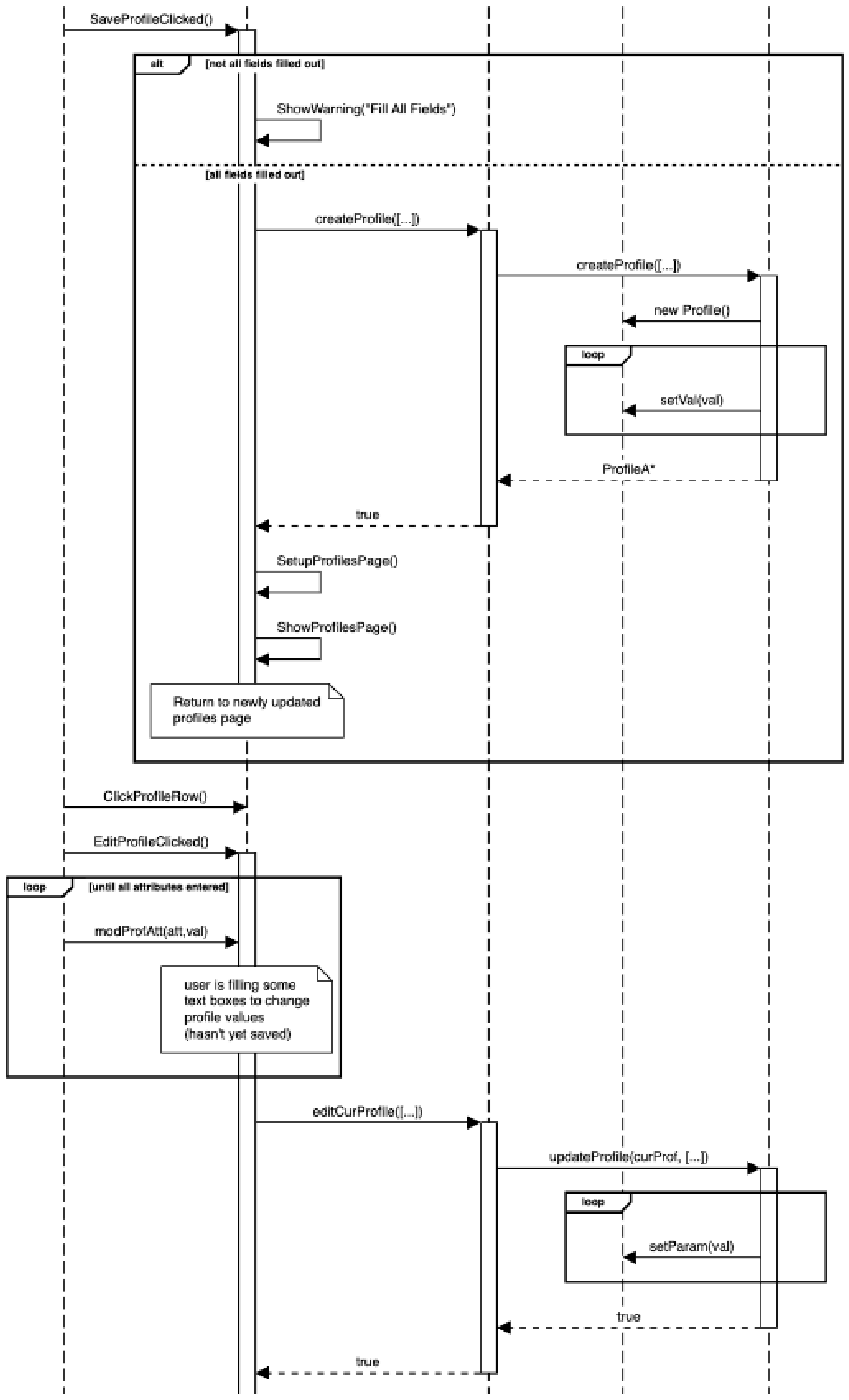


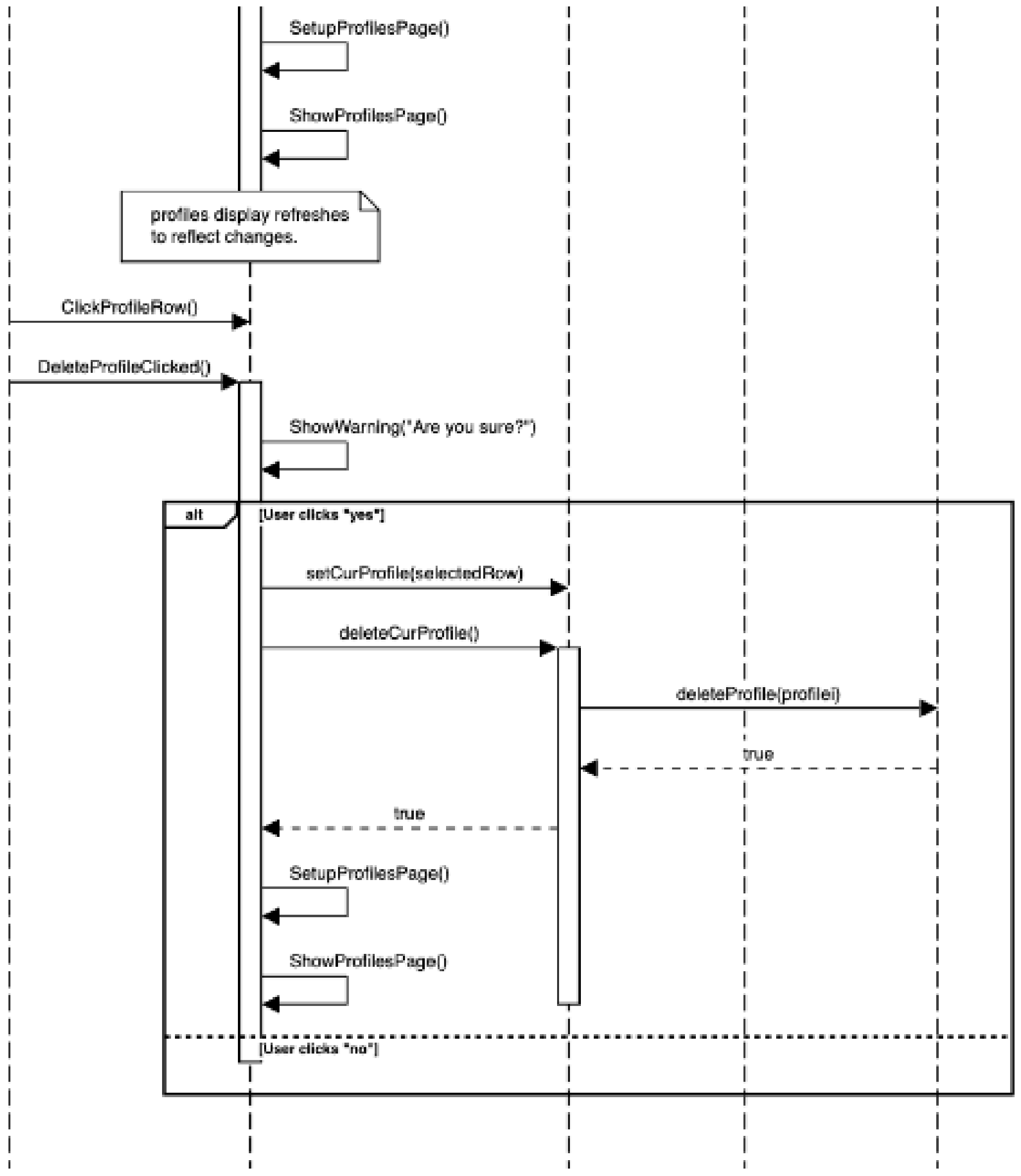

NextClicked()
setupRecomendationsPage()
curSnap.getRecomendations()
"placeholder"
displayRecomendationsPage()
NextClicked()
setupMenuPage()
showMenuPage()

Notes:
Showcases snapshots being pulled from profile history to be displayed, then snapshot being selected from among them and having its results displayed.

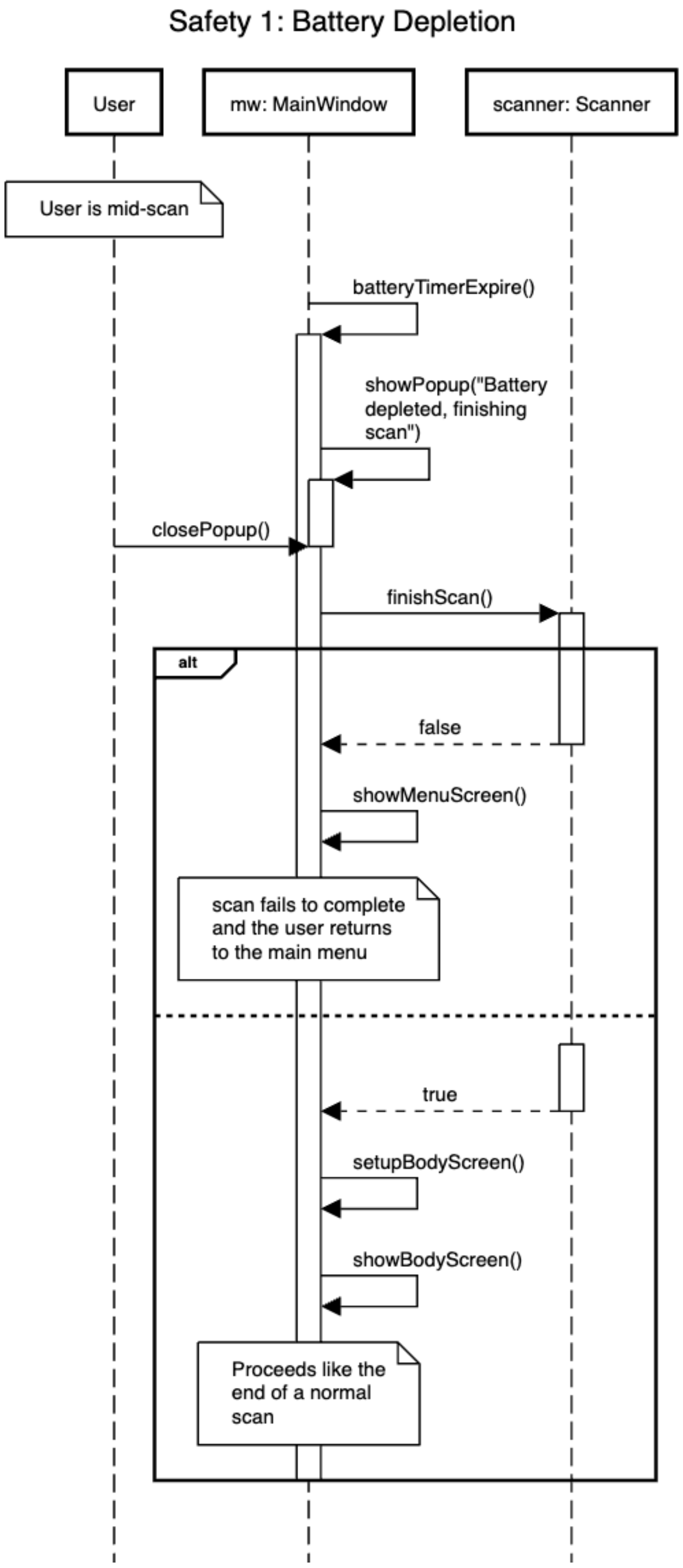
Success Scenario 3: User Creates, Modifies, and Deletes Profile




SaveProfileClicked()
alt
[not all fields filled out]
ShowWarning("Fill All Fields")
[all fields filled out]
createProfile([...])
createProfile([...])
new Profile()
loop
setVal(val)
ProfileA*
true
SetupProfilesPage()
ShowProfilesPage()
Return to newly updated profiles page
ClickProfileRow()
EditProfileClicked()
loop
[until all attributes entered]
modProfAtt(att,val)
user is filling some text boxes to change profile values (hasn't yet saved)
editCurProfile([...])
updateProfile(curProf, [...])
loop
setParam(val)
true
true



Notes: Success Scenario 3 shows profile selection and updating. Deletion happens very similarly to updates, but the model updates curProfile = nullptr, and the mainwindow loads the profile select screen (the one that it starts on). Profile Deletion also deletes any snapshots associated with the deleted profile.


Safety 1: Battery Depletion
User
mw: MainWindow
scanner: Scanner
User is mid-scan
batteryTimerExpire()
showPopup("Battery depleted, finishing scan")
closePopup()
finishScan()
alt
false
showMenuScreen()
scan fails to complete and the user returns to the main menu
true
setupBodyScreen()
showBodyScreen()
Proceeds like the end of a normal scan

Notes: battery depletion is handled very simply, but this suffices. If a scan has enough info to complete, it is registered in the DB and the user is given the normal post-scan breakdown. If not, they are returned to the main menu.

Requirements Traceability Matrix

ID	Requirement	Description	Related Use Case(s)	Fulfilled By	Tested By
1	User Management	Interface allows for the creation, updating, and deletion of up to 5 user profiles.	UC 1, UC 2, UC 3	Model, Profile, MainWindow	Success Scenario 3
2	Hardware Data Collection	Software interfaces with hardware to collect data - namely electric current readings.	UC 4	Scanner	Success Scenario 1
3	Software Data Collection	After each reading, the following data is collected: Body Temperature, Blood Pressure (L and R hands), Heart Rate, Sleep Time, Weight	UC 4	Scanner, MainWindow	Success Scenario 1
4	Data Processing	Processes data using algorithms to generate health metrics	UC 5	Snapshot	Success Scenario 2
5	Data Visualization	Display health metrics in an easy-to-understand format.	UC 5	MainWindow	Success Scenario 2
6	Specialist Recommendations	Provide a placeholder in the GUI.	UC 6	Snapshot, MainWindow	Success Scenario 2
7	Historical Data	Store historical data of past scans and have it be accessible to user. Data is available for trend analysis.	UC 5	Snapshot, Profile, MainWindow	Success Scenario 2
8	Battery Power	Battery power is tracked and displayed. It diminishes over time.	UC 7	MainWindow	Safety Scenario 1