AFEKA
The Academic College of Engineering in Tel Aviv

SoldiCare

INTEGRATIVE 2025

Semester B
Project Requirements Document - SoldiCare

10143

Department: Software Engineer

Lecturers: Deganit Armon, Tom Cohen

Students:

- Itay Chabra
  Role: Team, Team Lead, DBA
- Idan Noyshul
  Role: Team, Devops
- Sapir Gilany
  Role: Team, Scrum Master, Tech Writer
- Neta Ben Mordechai
  Role: Team, Product Owner, System Architecture
- Roi Dor
  Role: Team, QA Engineer, UI/UX

Table of Contents

1. Introduction

In modern combat environments, soldiers face immense physical and psychological challenges that can impact their health and operational effectiveness. Current methods for assessing soldiers' medical status in the field rely heavily on verbal communication and subjective observation, which may lead to delays in identifying critical health conditions.

SoldiCare is a smart real-time monitoring system designed to enhance medical decision-making for soldiers during combat by leveraging wearable sensors and cutting-edge Ambient Invisible Intelligence technology.

The concept of Ambient Invisible Intelligence, first defined by Gartner in late 2024, refers to the seamless integration of intelligent sensors that unobtrusively collect and analyze real-time data. In the context of SoldiCare, this technology ensures continuous health monitoring without interfering with soldiers' operational performance. By utilizing non-intrusive sensors, crucial biometric data such as blood pressure, heart rate, and oxygen levels can be gathered effortlessly, providing essential insights into soldiers' well-being in real time.

The development of SoldiCare aims to provide military units with access to accurate and timely medical data, supporting the early detection of health issues and enabling faster response times in combat scenarios. By integrating real-time health monitoring into operational decision-making, the system has the potential to enhance medical readiness and improve soldiers' well-being in the field.

1.1 Purpose of System

SoldiCare is designed to provide real-time health monitoring for soldiers in combat environments, enabling early detection of medical issues and data-driven decision-making. The system addresses the critical need for accurate, immediate medical insights that support operational effectiveness and soldier safety.

Beyond individual monitoring, SoldiCare also aims to provide a broader situational overview of the health status across entire military units. By analyzing aggregated health data, commanders can assess the combat readiness of their forces and determine whether adjustments are needed for mission planning. Instead of relying on "whoever shouts the loudest on the radio," this system will assist in prioritizing medical resource allocation, ensuring that aid is dispatched efficiently to the areas where it is needed most.

The primary goal of SoldiCare is to enhance battlefield medical readiness, improve resource allocation for emergency response, and support proactive intervention to prevent medical crises. By delivering objective, real-time health data, the system empowers decision-makers to prioritize treatment efficiently and respond effectively to evolving medical situations.

1.2 Scope of System

SoldiCare operates as a medical monitoring system tailored for battlefield environments, providing continuous assessment of soldiers' health status. The system's scope includes:

1.2.1 Core Functionality

- Collection of biometric data – Monitoring heart rate, blood pressure, oxygen saturation, stress indicators, and additional physiological metrics.
- Real-time data transmission – Displaying collected health data and unit readiness on the system's statistics dashboard, providing immediate visibility for medical units and command centers.
- Automated alerts for abnormal health patterns – Detecting irregularities and triggering real-time warnings, supporting quick decision-making in emergency scenarios.
- User and admin role management – Defining access levels to ensure proper handling and monitoring of medical data across different system users.

1.2.2 Security and Data Protection

- No Guaranteed Encryption for Transmitted Data – The system does not provide full encryption for transmitted biometric data, and there is no assurance of secure end-to-end data protection.
- Limited Access Control Mechanisms – While user roles are defined, there are no advanced authentication or authorization layers to restrict access to sensitive information.
- No Dedicated Cybersecurity Monitoring – The system does not include active security monitoring or intrusion detection to identify unauthorized access or potential threats.

1.2.3 Operational Constraints

- Designed for combat conditions but relies on stable network connectivity for real-time updates.
- The system does not replace direct medical evaluations but supplements them with objective health indicators.
- The system assumes that wearable sensors maintain constant connectivity under all operational conditions. Therefore, scenarios involving connectivity failures are not considered in the system's workflows.

1.2.4 Limitations

- SoldiCare does not provide medical diagnoses but rather supports decision-making with health trends.
- Environmental factors, such as extreme weather or signal disruptions, may affect data transmission.
- The scope of this project does not include predictive AI-based medical recommendations.

2. Actors and goals

2.1 Soldiers (Primary Actor)

Soldiers are equipped with wearable health monitoring sensors. Their goal is to track their personal health status, receive alerts for abnormal conditions, and enhance their awareness of their physical state during combat operations.

2.2 Military Medics (Primary Actor)

Field medics responsible for assessing and treating injured soldiers. Their goal is to access up-to-date medical data, quickly identify emergencies, and provide effective treatment based on accurate real-time information rather than subjective reports.

2.3 Commanders (Primary Actor)

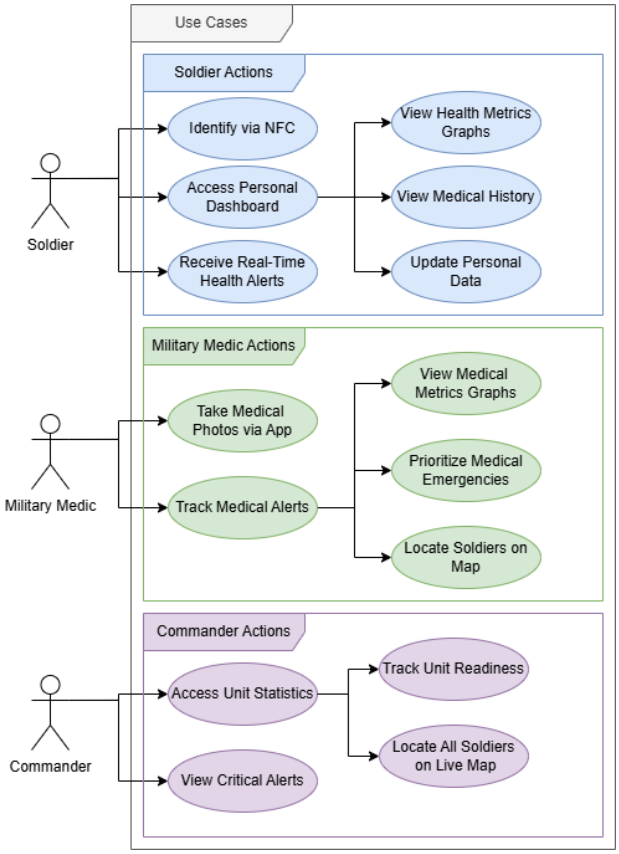
Unit commanders overseeing the combat readiness of their soldiers. Their goal is to gain a broader picture of the overall physical condition of their unit, analyze trends, and prioritize medical assistance where it is most needed. Commanders do not have access to individual medical details but can view general fitness indicators for operational planning.

2.4 Medical Command Center (Support Actor)

A centralized medical authority responsible for monitoring soldier health trends across multiple units. Their goal is to analyze aggregated health data, identify large-scale medical concerns, and optimize the allocation of medical resources based on battlefield conditions.

3. Functional Requirements

3.1 Use Case Diagram



3.2 Use Case Details

3.2.1 Use Case 1

Name: Real-Time Health Monitoring & Alerts

Goals: Provide real-time tracking of soldiers' vital signs and generate alerts for medical abnormalities.

Participating Actors: Soldiers (Primary), Military Medics (Primary), Commanders (Primary), Medical Command Center (Support).

Basic Workflow:

1. Soldier's wearable sensors continuously collect and log biometric data, including heart rate, blood pressure, oxygen levels, and stress indicators.
2. Data is transmitted in real-time to the medical monitoring system.
3. The system stores all received medical metrics in the database over time and continuously processed real-time data.
4. If a critical deviation is detected (e.g., sudden drop in oxygen saturation), the system alerts the abnormality for the specific soldier.
5. Medics receive a detailed overview, including real-time graphs, previous health records, and the soldier's location for rapid intervention.

Alternative Workflow A: False Alarm Prevention

A.1 At Step 4, the system detects a sudden abnormal heart rate spike but notices stabilization within seconds.

A.2 Instead of triggering an immediate alert, the system continues monitoring to confirm the trend.

A.3 If the soldier's health remains stable for 30 seconds, no alert is triggered, and workflow continues from Step 4.

A.4 If further abnormalities occur, an alert is sent to the medics for investigation.

3.2.2 Use Case 2

Name: Real-Time Image Transfer

Goals: Enable real-time transmission of images of injured soldiers in the field by military medics.

Participating Actors: Military Medics (Primary), Soldiers (Primary)

Basic Workflow:

1. A soldier is injured in the field.
2. A medic scans the soldier's NFC tag to quickly retrieve identification details.
3. The medic captures an image of the wounded soldier, which is immediately transmitted to the system.
4. The system displays the image alongside the relevant medical alert, ensuring a rapid and informed response.

Alternative Workflow A: Image Transmission Failure Handling

A.1 At Step 3, multiple If the image capture fails due to network issues or device malfunction, the system automatically attempts a secondary transmission using a backup connection.

A.2 If the transmission continues to fail, the medic manually inputs key injury details into the system to ensure essential medical information is recorded.

A.3 Once communication is restored, the stored image is reprocessed and added to the soldier's medical record.

3.2.3 Use Case 3

Name: Combat Readiness & Health Assessment

Goals: Provide commanders with an overview of soldiers' physical readiness for missions.

Participating Actors: Soldiers (Primary), Commanders (Primary), Medical Command Center (Support).

Basic Workflow:

1. Before an intensive mission, the commander conducts an individual and unit-wide health assessment, ensuring all soldiers are fit for deployment.
2. Soldiers report any recent injuries, fatigue, or medical concerns that might affect their combat performance.
3. The system displays each soldier's recorded medical history, allowing the commander to evaluate potential risks.
4. If necessary, the commander consults medical personnel to determine whether specific soldiers require additional rest or medical clearance.
5. Based on the assessment, final combat readiness decisions are made, ensuring optimal deployment strategy.

Alternative Workflow A: Extreme Physical Degradation

A.1 At Step 5, system analytics indicate a sudden decline in overall unit readiness beyond acceptable thresholds.

A.2 Commanders are alerted with predictive health deterioration metrics.

A.3 If additional rest or medical intervention is scheduled, soldier fitness levels gradually improve.

A.4 Otherwise, the unit continues monitoring trends, and workflow resumes from Step 5.

4. Non-Functional Requirements

A table with the following structure:

Requirement number	Requirement Type	Requirement Description
1	U - Usability	Visual Data Representation for Enhanced Usability – The system must display graphs and medical indicators alongside alerts to provide a clear and intuitive visualization of health trends, aiding quick decision-making.
2	U - Usability	Responsive System Design – The system must provide a fully responsive interface, ensuring seamless accessibility and proper display across various devices, including desktops, tablets.
3	R - Reliability	Medical-Grade Alert Classification – The system must classify and prioritize alerts based on medical urgency standards.
4	P - Performance	Concurrent Session Handling – The system must support at least 10 active sessions simultaneously without performance degradation, ensuring stable operation under typical workload conditions.
5	S - Supportability	Extensibility & Scalability – The system must support future expansion, allowing seamless integration of additional components as needed.
6	S - Supportability	Logging & Debugging Mechanisms – The system must provide comprehensive logging and error-tracking capabilities, enabling efficient fault analysis and operational maintenance.

5. Appendix - Technologies

5.1 Server Side

- Spring Boot: Used to build a modular and scalable RESTful API in Java.
- MongoDB: NoSQL database chosen for flexible document-based storage of complex, nested data structures (e.g., soldiers, sensors, alerts).
- YAML Configuration: Used to define a local development environment, including pulling a MongoDB Docker image for testing and runtime.
- Testing: Implemented behavior-driven scenarios written in Gherkin syntax and executed with JUnit.
- Render & CI/CD: Configured a CI/CD pipeline that deploys the backend to Render on each pull request, enabling seamless cloud deployment using Docker file.

5.2 Frontend Web

- React + TypeScript: Implemented a component-based single-page application with type safety.
- Material UI: Applied as the primary design and layout framework for responsive, accessible interfaces.
- React Hooks & Features: Utilized useState, useEffect, useContext, local storage, and .env files for state and configuration management.
- Custom Hooks: Created reusable logic abstractions to improve code maintainability and readability.
- Leaflet API: Integrated for real-time map visualization of soldier positions and alert zones.
- Recharts Library: Used to display dynamic sensor data in time-series graphs.
- Firebase API: Connected to retrieve metadata and URLs of images associated with soldier profiles.

5.3 Image Storage

- Firebase Storage - Cloud file storage and management
- Firebase BOM 33.15.0 - Bill of Materials for version management
- Google Services Plugin - Firebase integration

5.4 Sensor Simulation

- Python Script: Implemented a simulation tool using `requests` to send real-time health data to the backend.
- Spring Integration: The simulator interacts directly with the backend REST API, mimicking real-world sensor behavior.

5.5 Android Application

5.5.1 Programming Languages

- Java - Primary application logic and Android development
- Kotlin - Build configuration (Gradle build scripts)
- XML - UI layouts, resources, and Android manifest configuration

5.5.2 Camera & Media

- CameraX 1.3.0 - Modern camera API for image capture
- camera-camera2 - Camera2 API backend
- camera-lifecycle - Lifecycle-aware camera operations
- camera-view - Camera preview UI components
- ExifInterface 1.3.7 - Image metadata and orientation handling
- Bitmap & Graphics APIs - Image processing and compression
- FileProvider - Secure file sharing between app components

5.5.3 NFC Technology

- Android NFC API - Near Field Communication integration
- NDEF (NFC Data Exchange Format) - Standard for NFC data structure
- Foreground Dispatch System - Priority NFC event handling
- Multiple NFC Technologies Support:
  - NfcA (ISO 14443-3A)
  - NfcB (ISO 14443-3B)
  - NfcF (JIS 6319-4)
  - NfcV (ISO 15693)
  - NDEF (NFC Data Exchange Format)

5.5.4 UI & User Experience

- Material Design Components - Modern Android UI patterns.
- ConstraintLayout - Flexible and efficient layouts.
- Animation APIs - Visual feedback and transitions.
- Toast & Progress Indicators - User feedback systems.

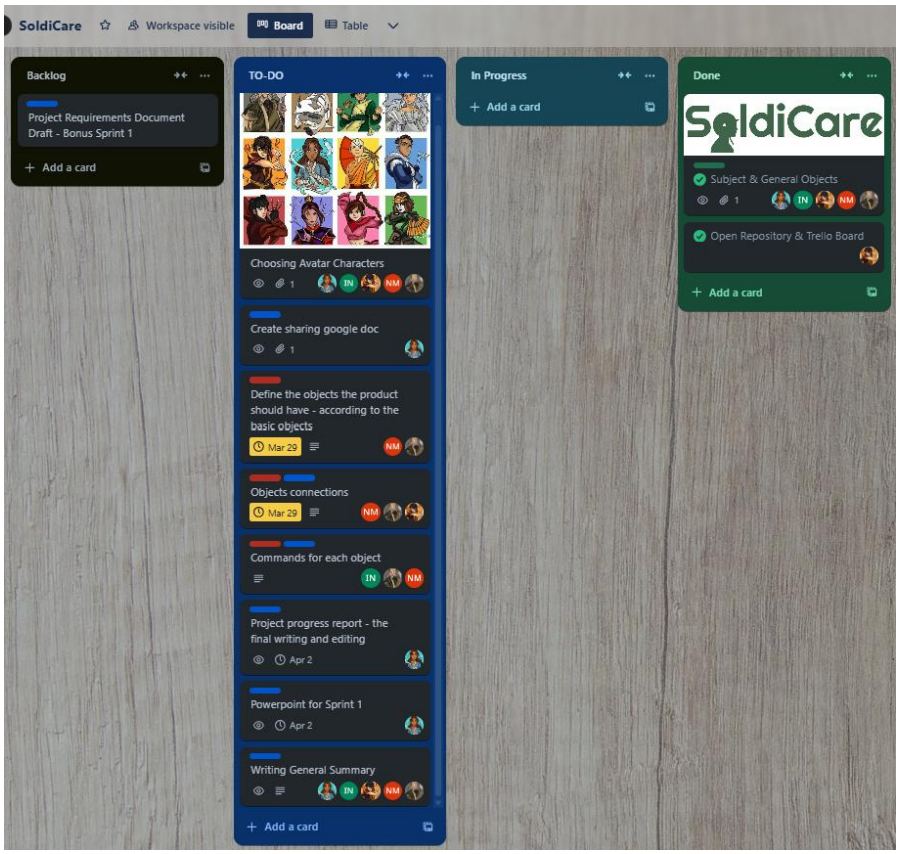
6. Appendix – Project Summary

6.1 Kanban Board Samples

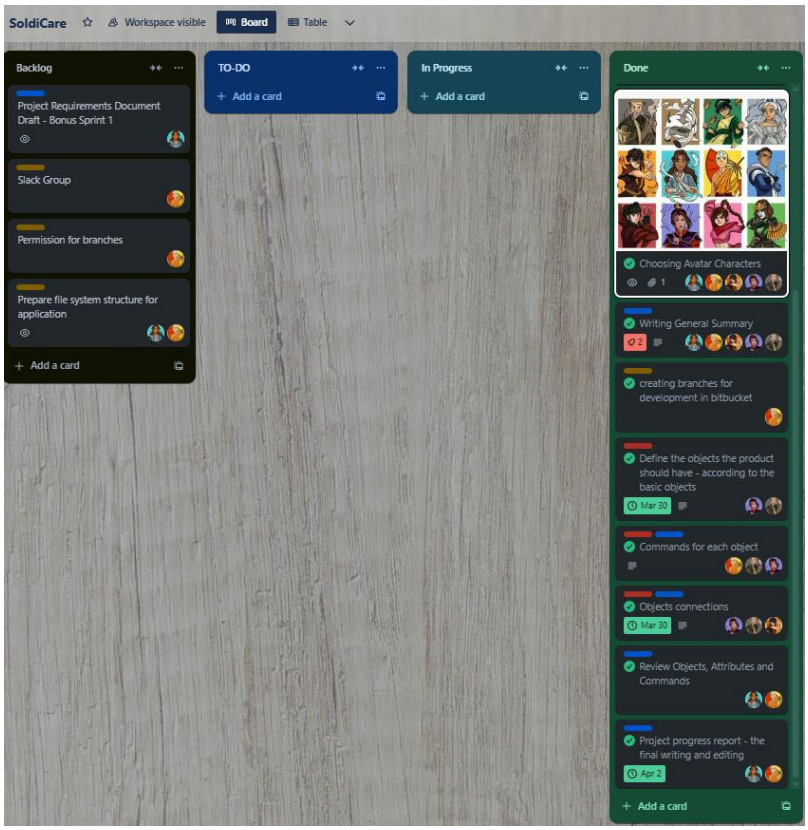
Samples collected throughout the semester are organized from the oldest to most recent, categorized by sprint.

6.1.1 Sprint 1

Date: 29/03/2025

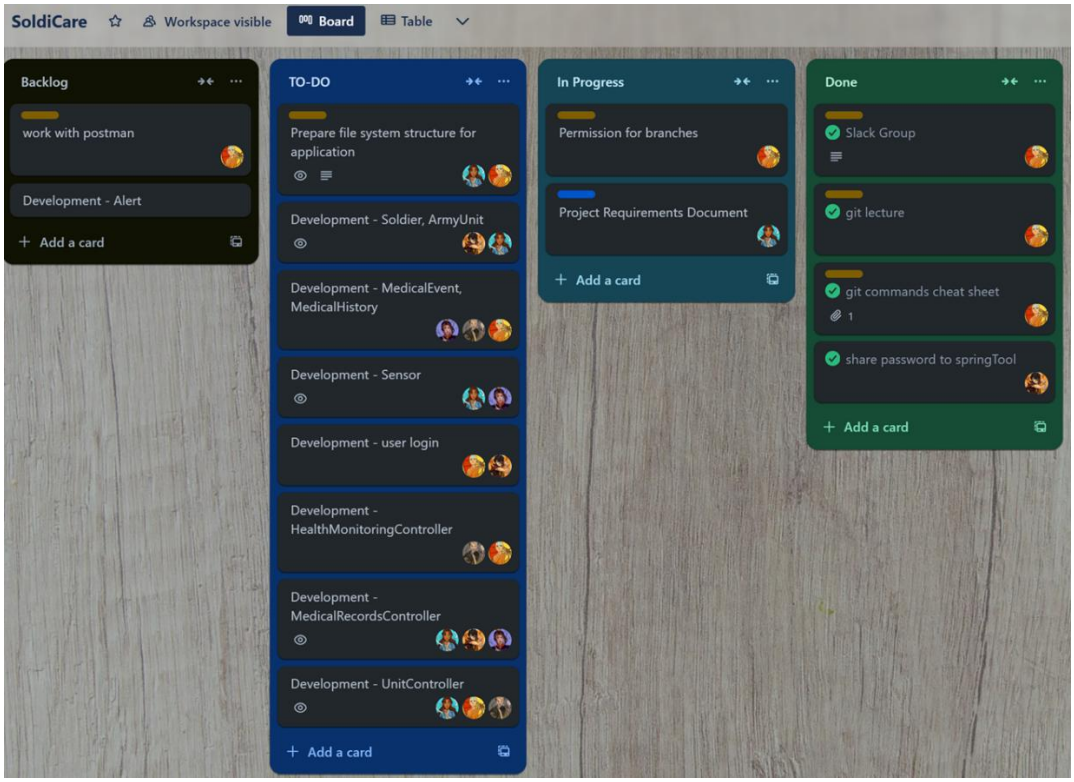


Date: 01/04/2025

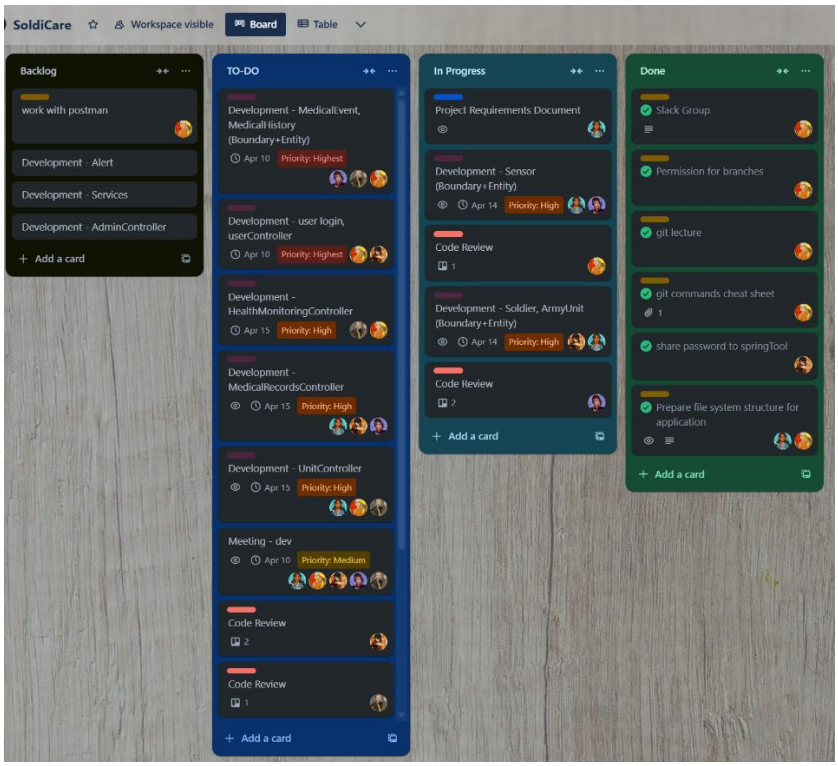


6.1.2 Sprint 2

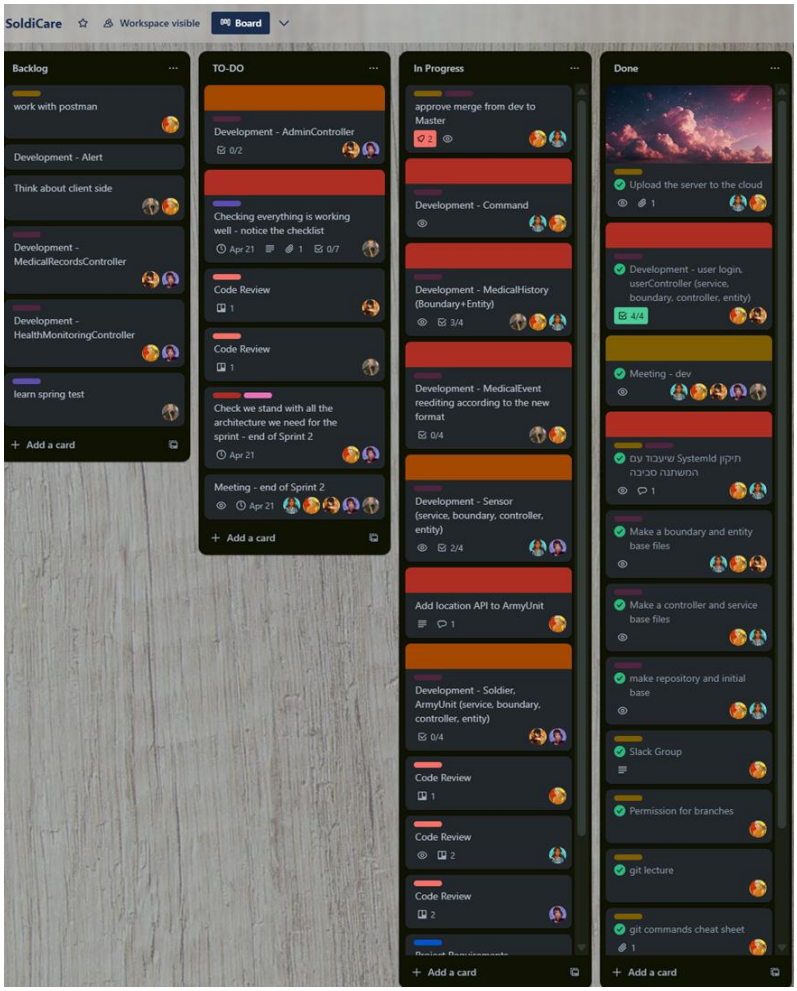
Date: 05/04/2024



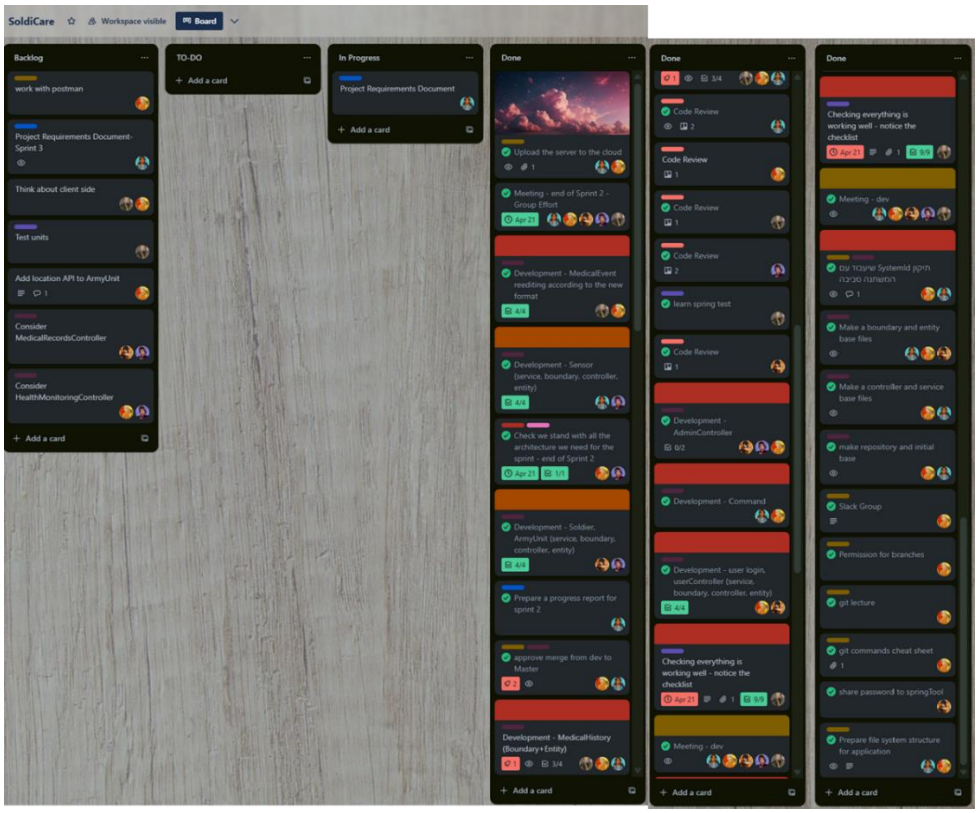
Date: 09/04/2025



Date: 15/04/2025

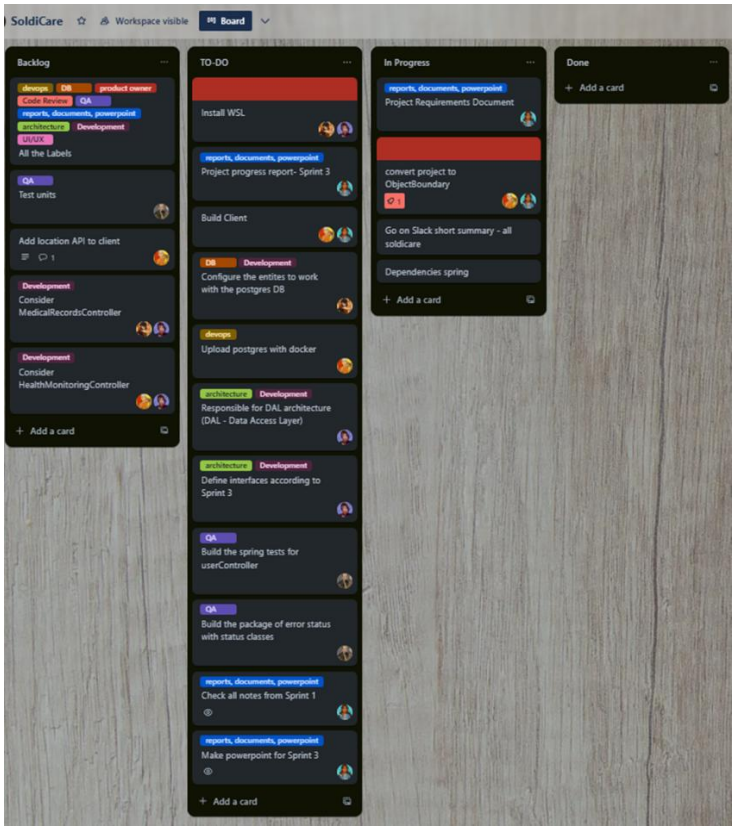


Date: 23/04/2025

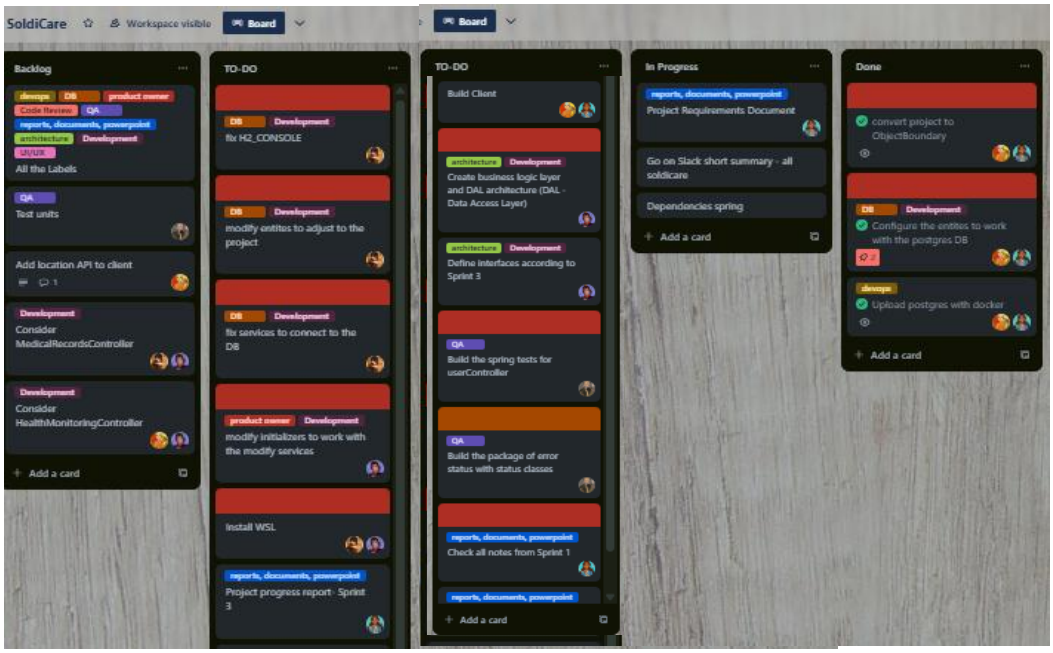


6.1.3 Sprint 3

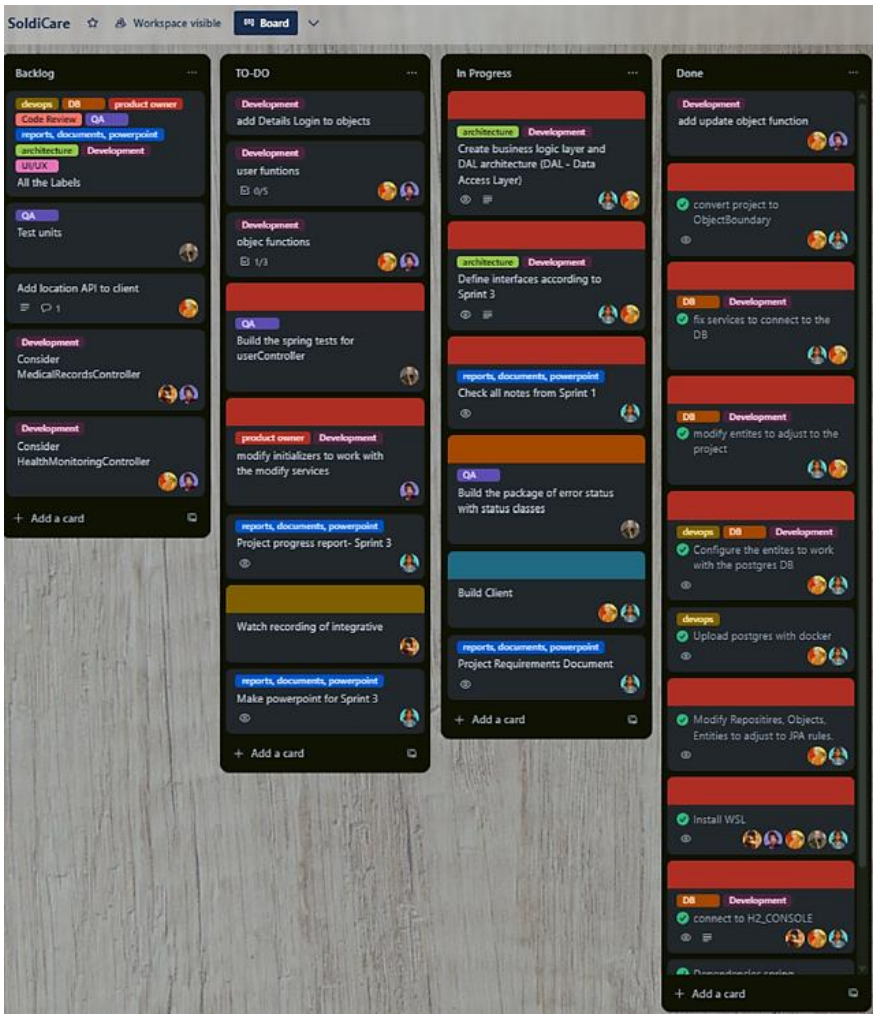
Date: 03/05/2025



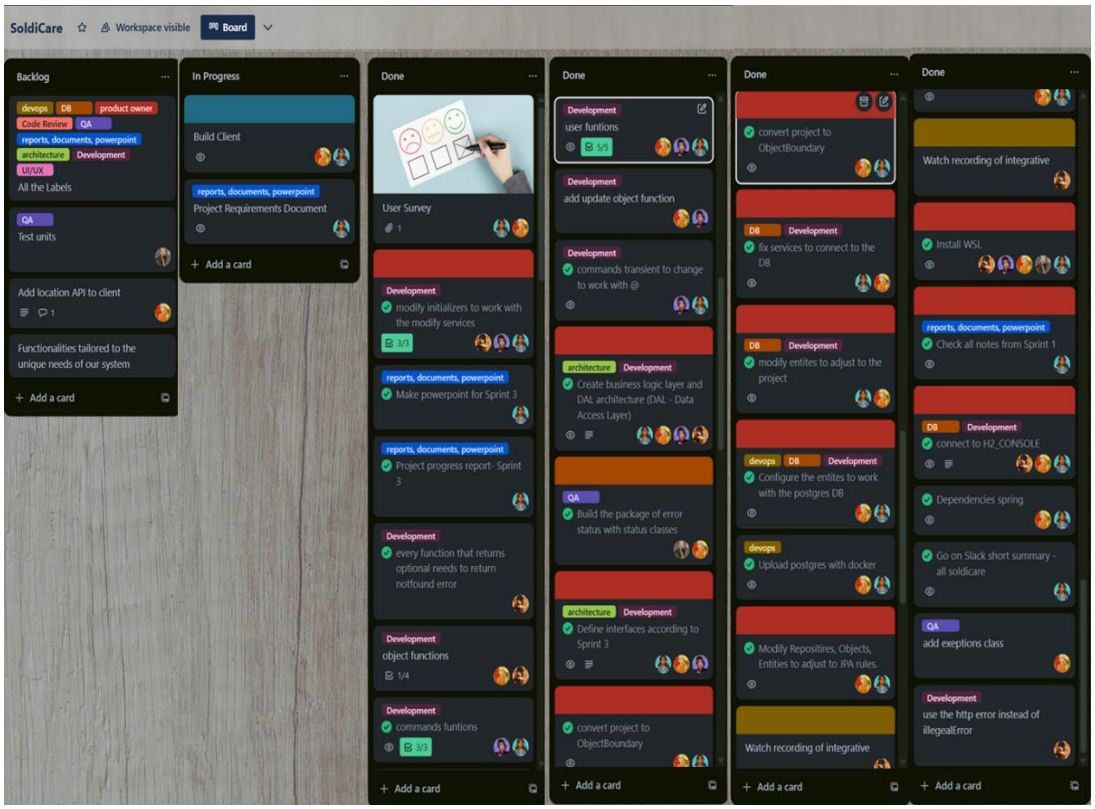
Date: 05/05/2025



Date: 11/05/2025

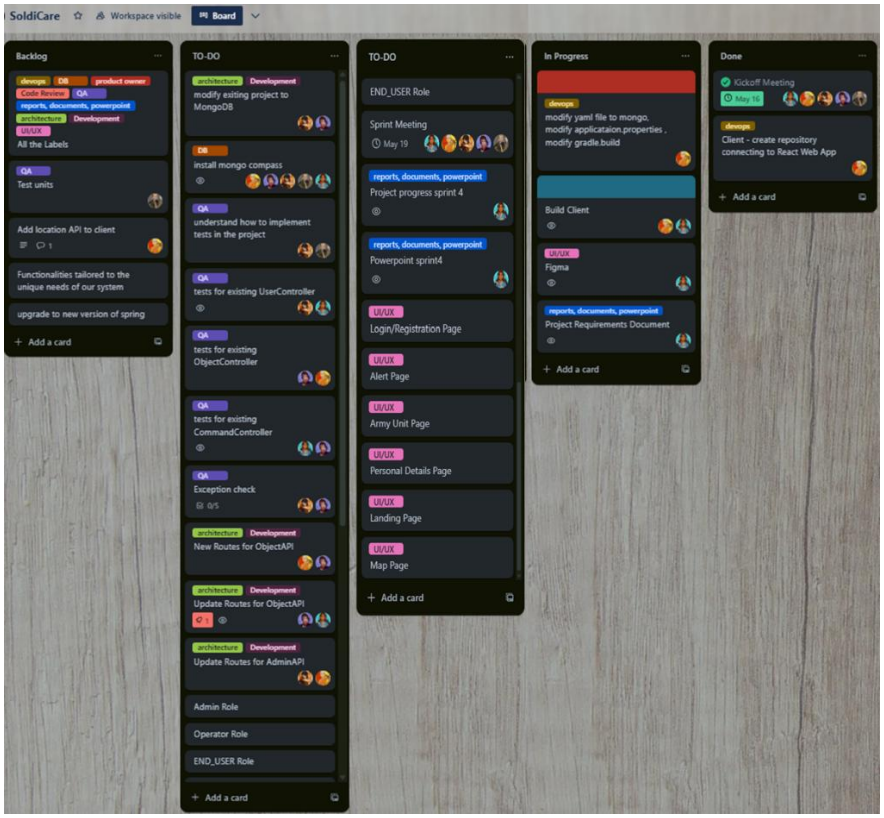


Date: 13/05/2025

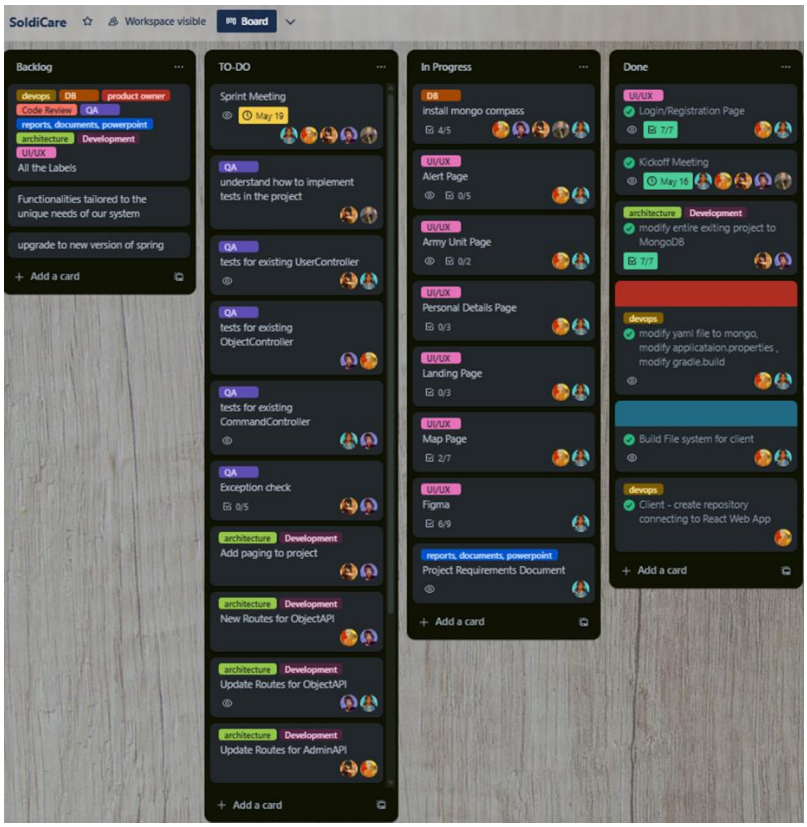


6.1.4 Sprint 4

Date: 16/05/2025

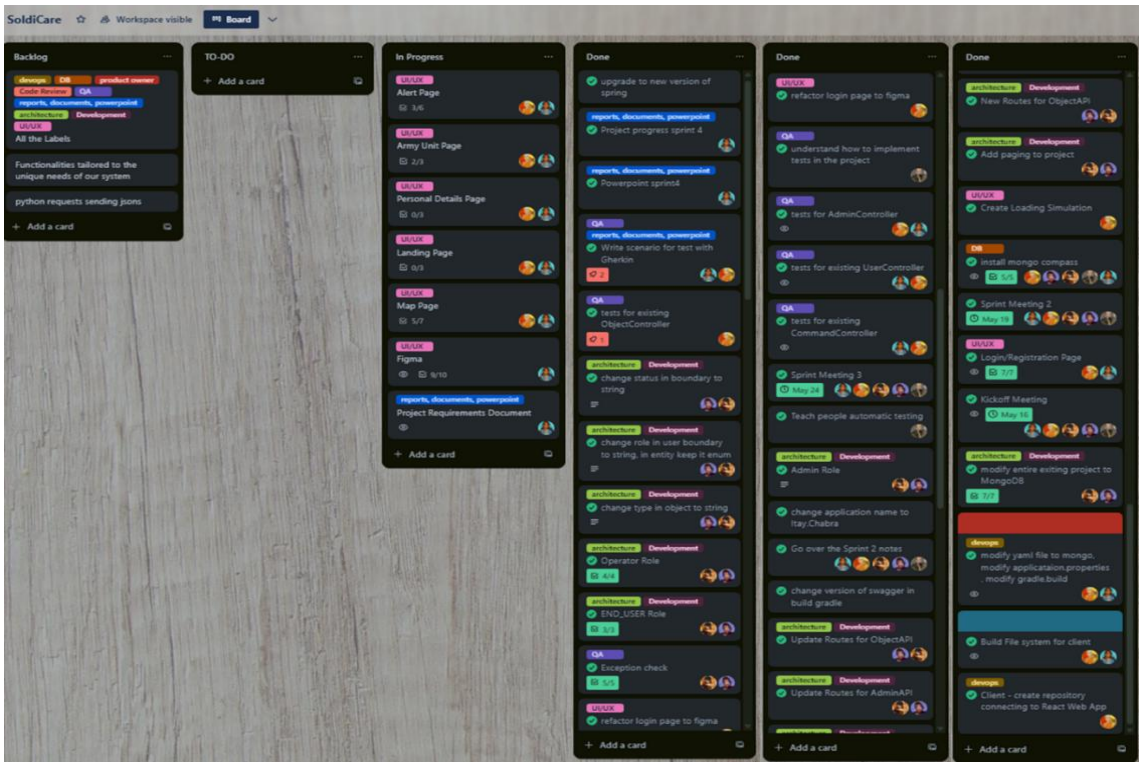


Date: 19/05/2025



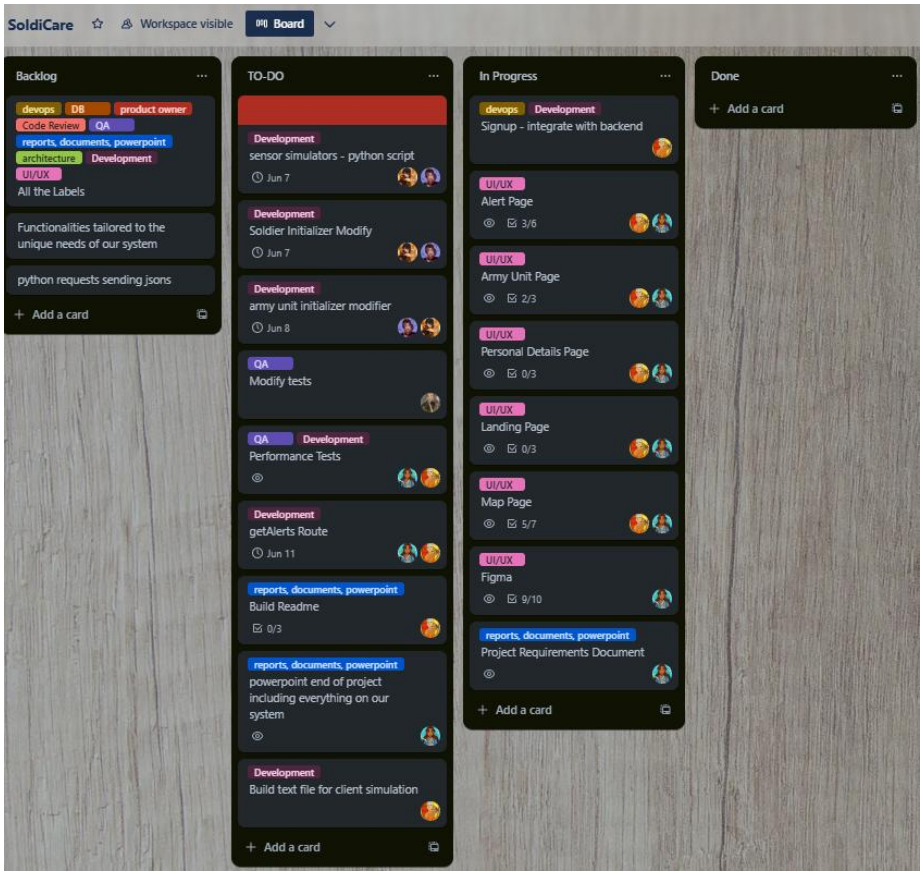
Date: 24/05/2025

Date: 27/05/2025

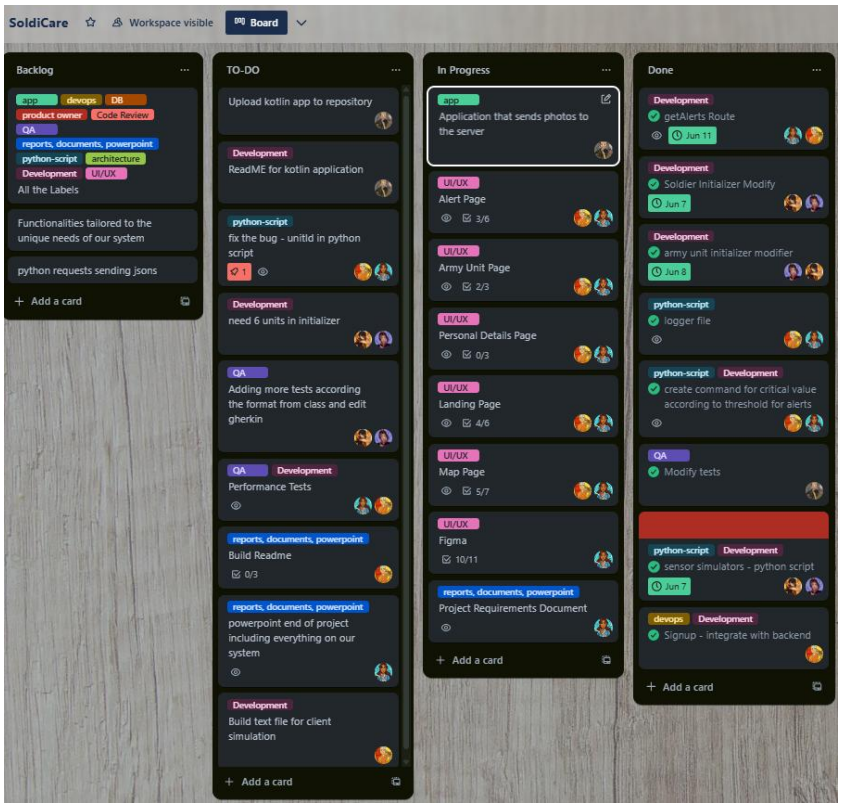


6.1.5 Sprint 5

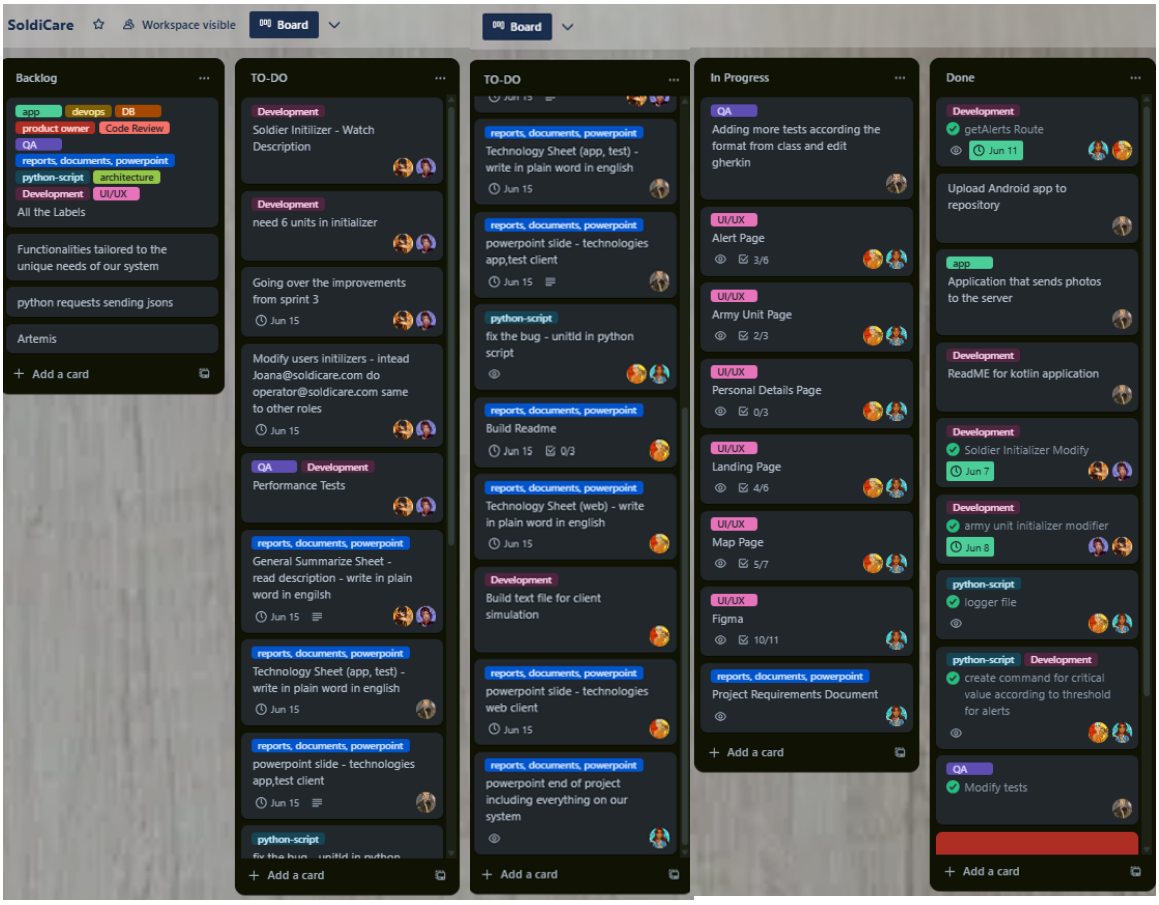
Date: 03/06/2025

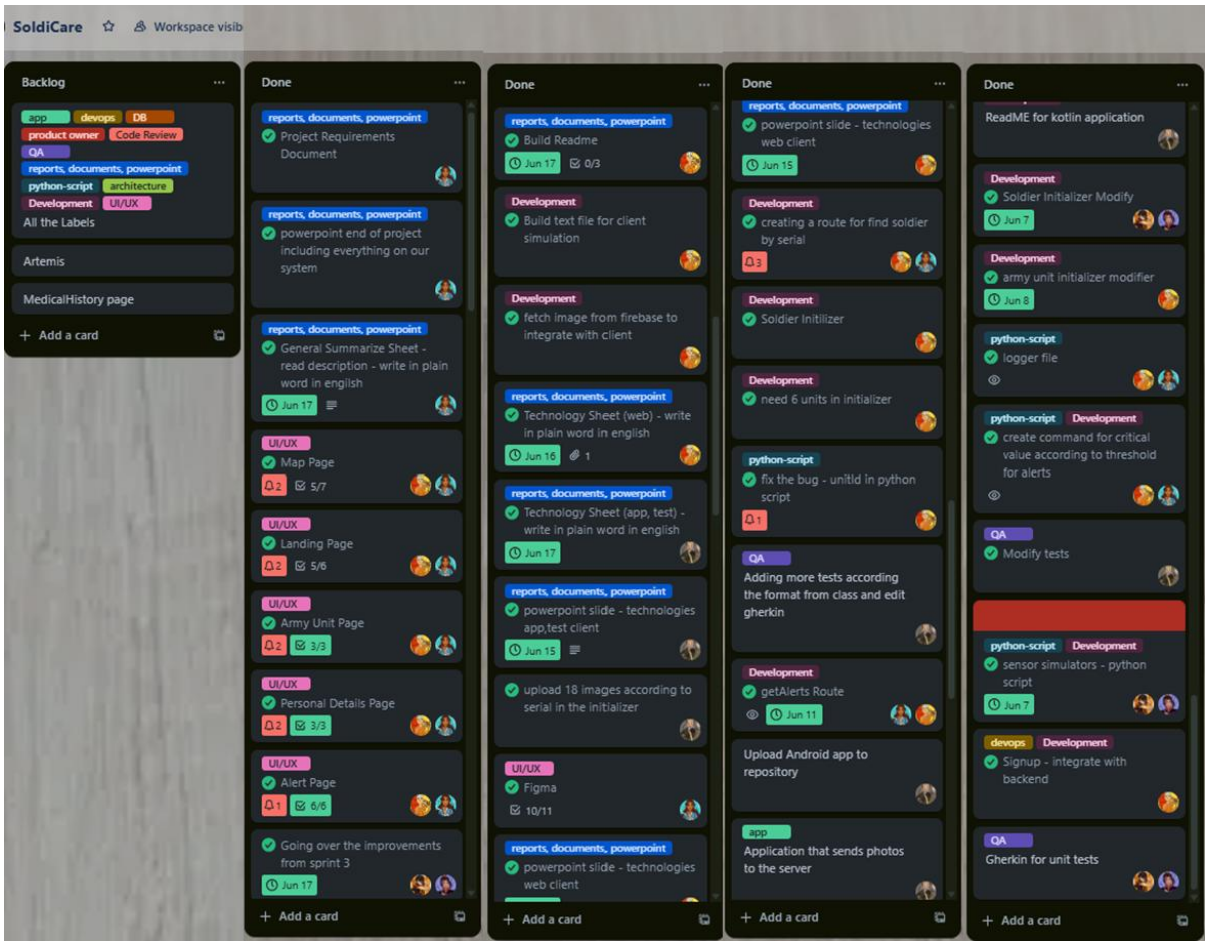


Date: 10/06/2025



Date: 14/06/2025





6.2 Project Retrospective

6.2.1 Team Strengths to Maintain

- Effective Kickoff Meetings – Held initial meetings to clarify objectives and distribute tasks efficiently.
- Structured Git workflow – Maintained organized branching and thorough PR reviews for code quality.
- Dividing tasks into sub-teams – Assigned focused tasks to small groups for better efficiency.
- Commitment to deadlines and quality – Met sprint goals while ensuring high development standards.
- Thorough code review processes – Conducted mandatory reviews before merging PRs to prevent issues.

6.2.2 Areas for Improvement

- Strengthening team communication and synchronization – Holding regular meetings to improve coordination and workflow.
- Encouraging proactive engagement and responsibility – Promoting initiative and ownership in task execution.
- Enhancing feedback culture – Creating an open space for constructive discussions and shared insights.
- Improving task execution within timelines – Maintaining steady progress on individual and sub-team tasks, ensuring that all assigned sprint activities are completed on time.
- Fostering innovation and new initiatives – Supporting creative thinking and exploring new approaches for efficiency.

6.2.3 Most Enjoyable Aspects

- Hands-On Learning Experience – Gaining practical experience through a real project, applying theoretical knowledge, and developing new skills.
- Creative Problem-Solving – Tackling technical challenges and finding innovative solutions that made the work engaging and rewarding.
- Building a Functional Product – Seeing a fully operational system designed, developed, and successfully tested by the team.

6.2.4 What We Would Do Differently

- Assigning roles based on prior knowledge and individual expertise.
- Earlier planning of all system components to be implemented – Establishing a comprehensive development roadmap at the project's outset to ensure smoother execution.

6.2.5 Remote Collaboration

- Tools Used
    - Microsoft Teams – Used for team meetings and remote discussions.
    - Trello – Managed task distribution among team members efficiently.
    - Bitbucket & Git – Maintained structured workflow with Pull Requests, branches, and code reviews for each task, alongside separate repositories for different system components.

- Slack – Shared key points from lessons, explanation documents, and technical guides (Git commands, Docker integration, etc.), while receiving notifications on PRs from Git.
- Google Docs – Facilitated collaborative document sharing and editing.

- Impact of Remote Work – Remote collaboration allowed for flexible communication and task management, ensuring smooth progress despite physical distance. While digital tools streamlined workflow, real-time coordination was sometimes challenging, emphasizing the importance of maintaining structured sync meetings and clear documentation.

7. Appendix – External Links

7.1 Figma

Figma Link

7.2 System Video

System Video Link

7.3 Bitbucket

Bitbucket Link

SoldiCare

AFEKA
The Academic College of Engineering in Tel Aviv

Project Progress Report – Sprint 5

INTEGRATIVE 2025 Semester B (10143)

June 18th, 2025

Students

Itay Chabra

Role: Team, Team Lead & DBA

Idan Noyshul

Role: Team, Devops

Sapir Gilany

Role: Team, Scrum Master & Tech Writer

Neta Ben Mordechai

Role: Team, Product Owner, System Architecture

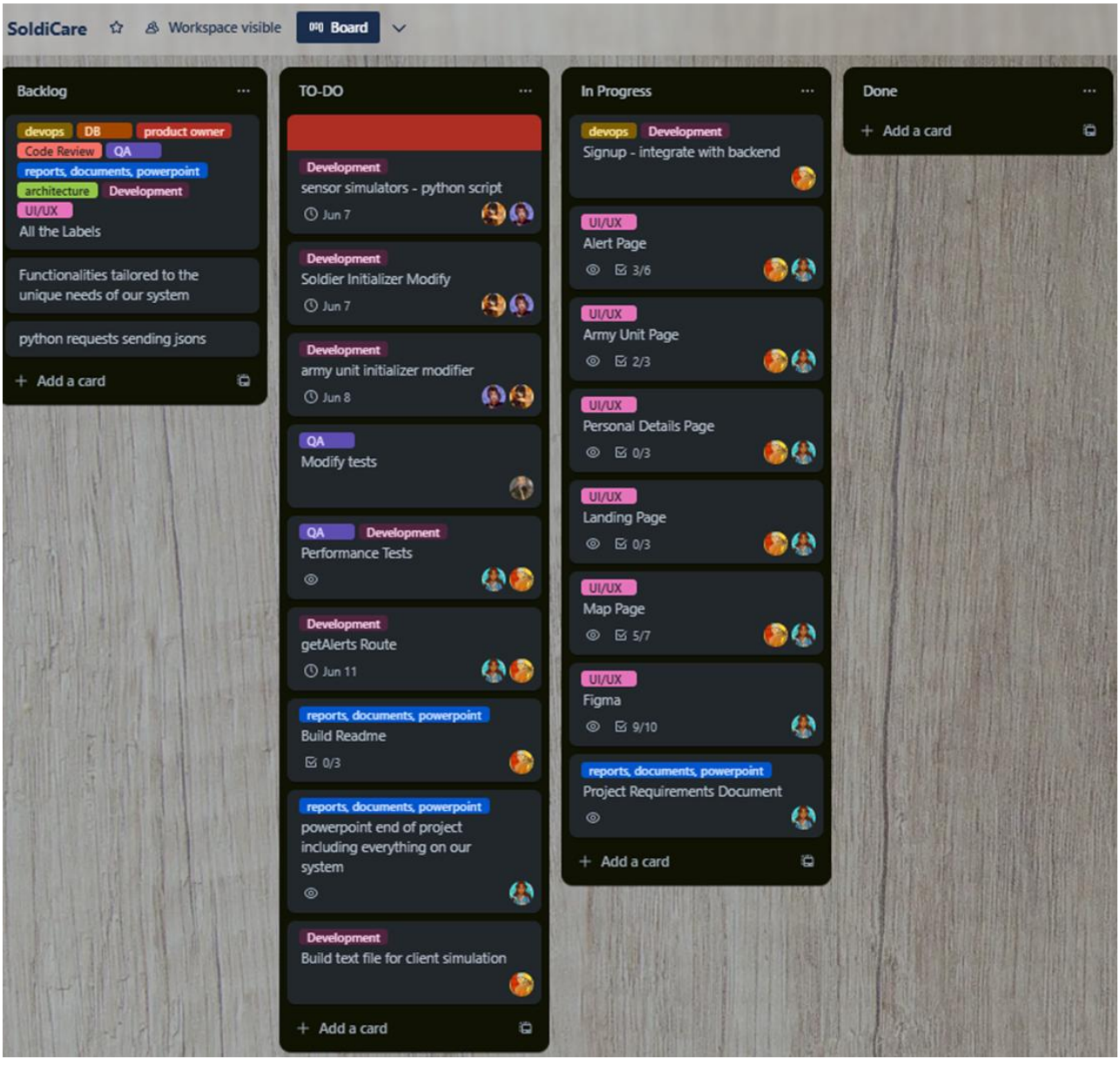
Roi Dor

Role: Team, QA Engineer, UI/UX

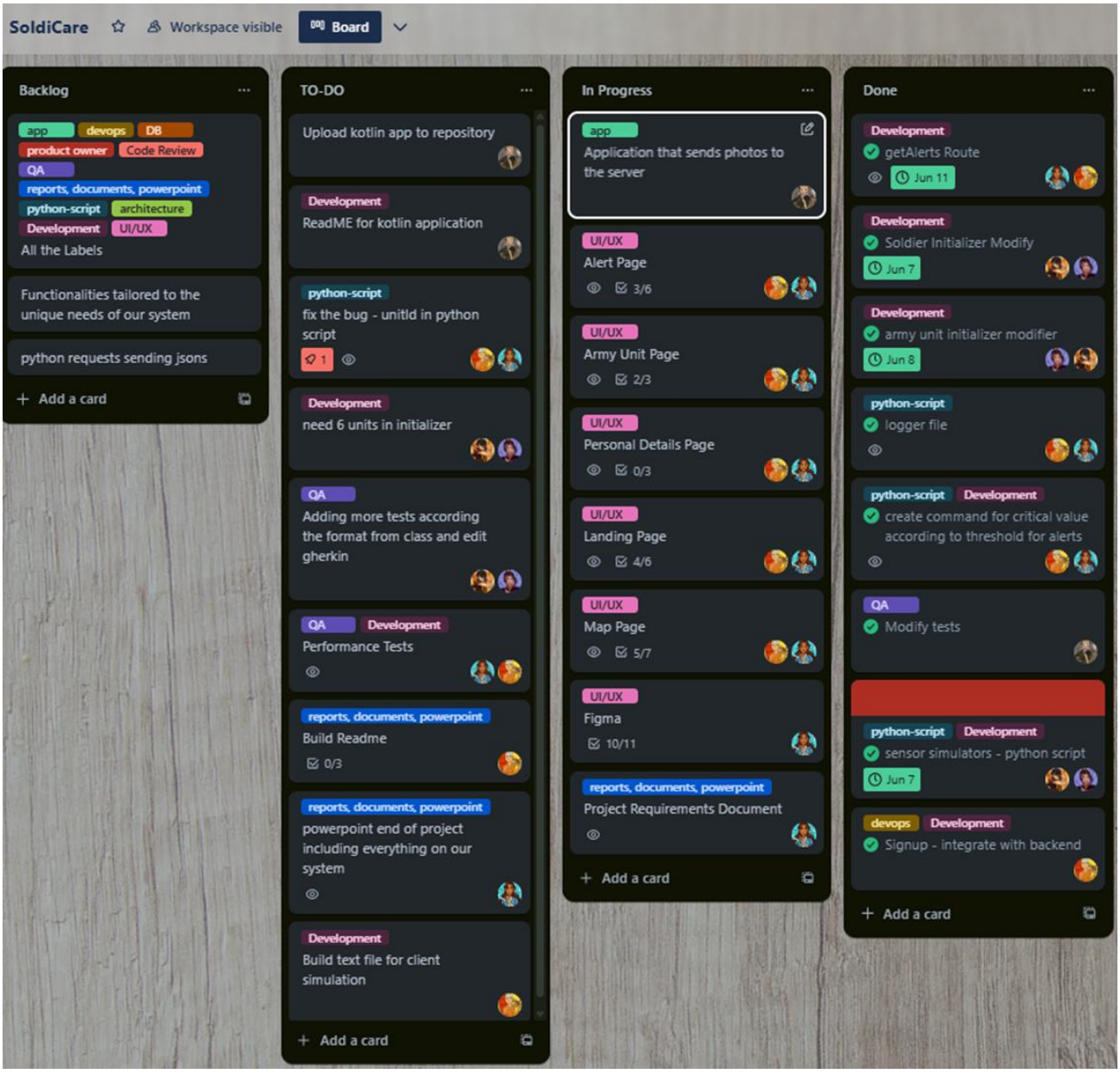
Contents

Kanban Boards

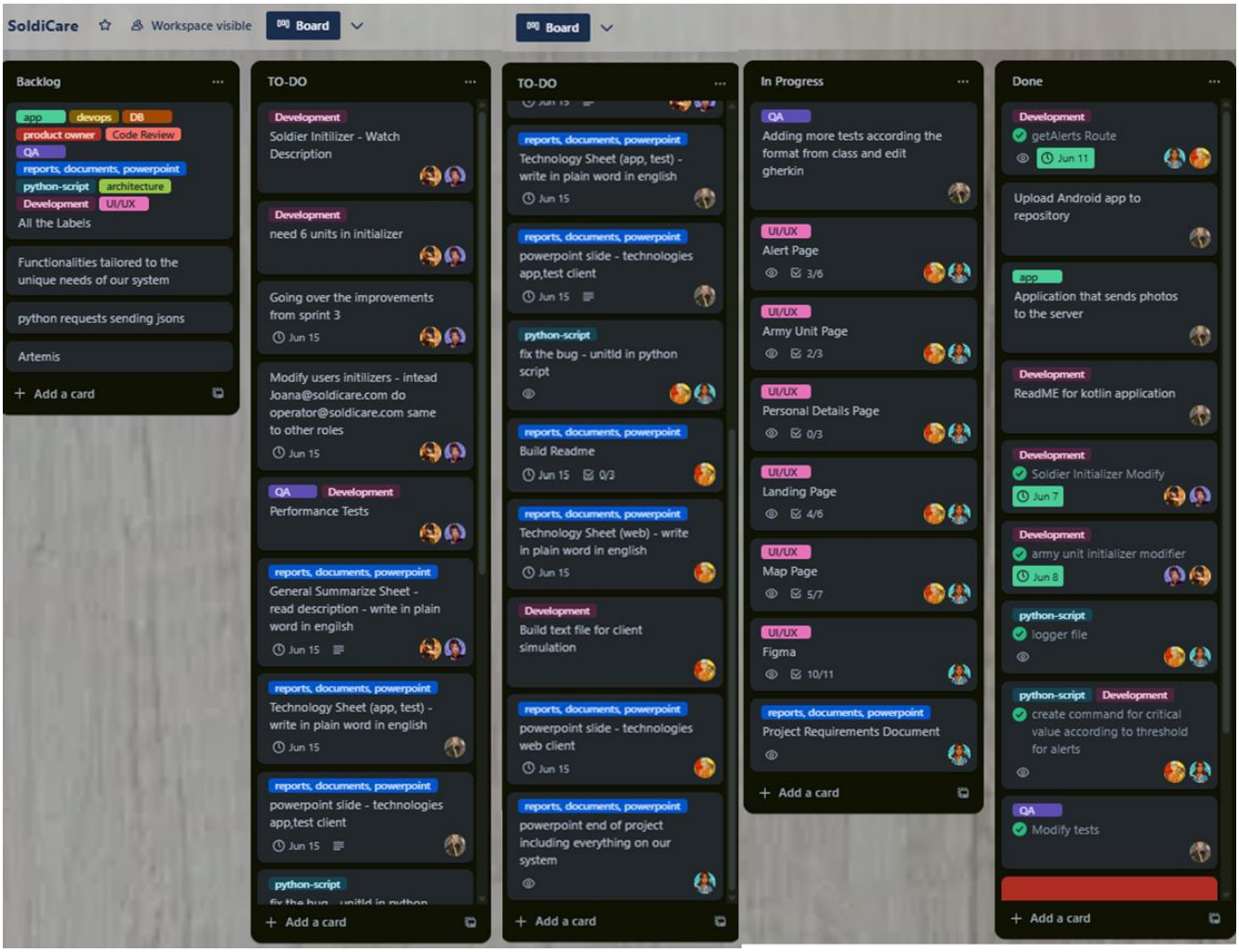
First State (03/06/2025)



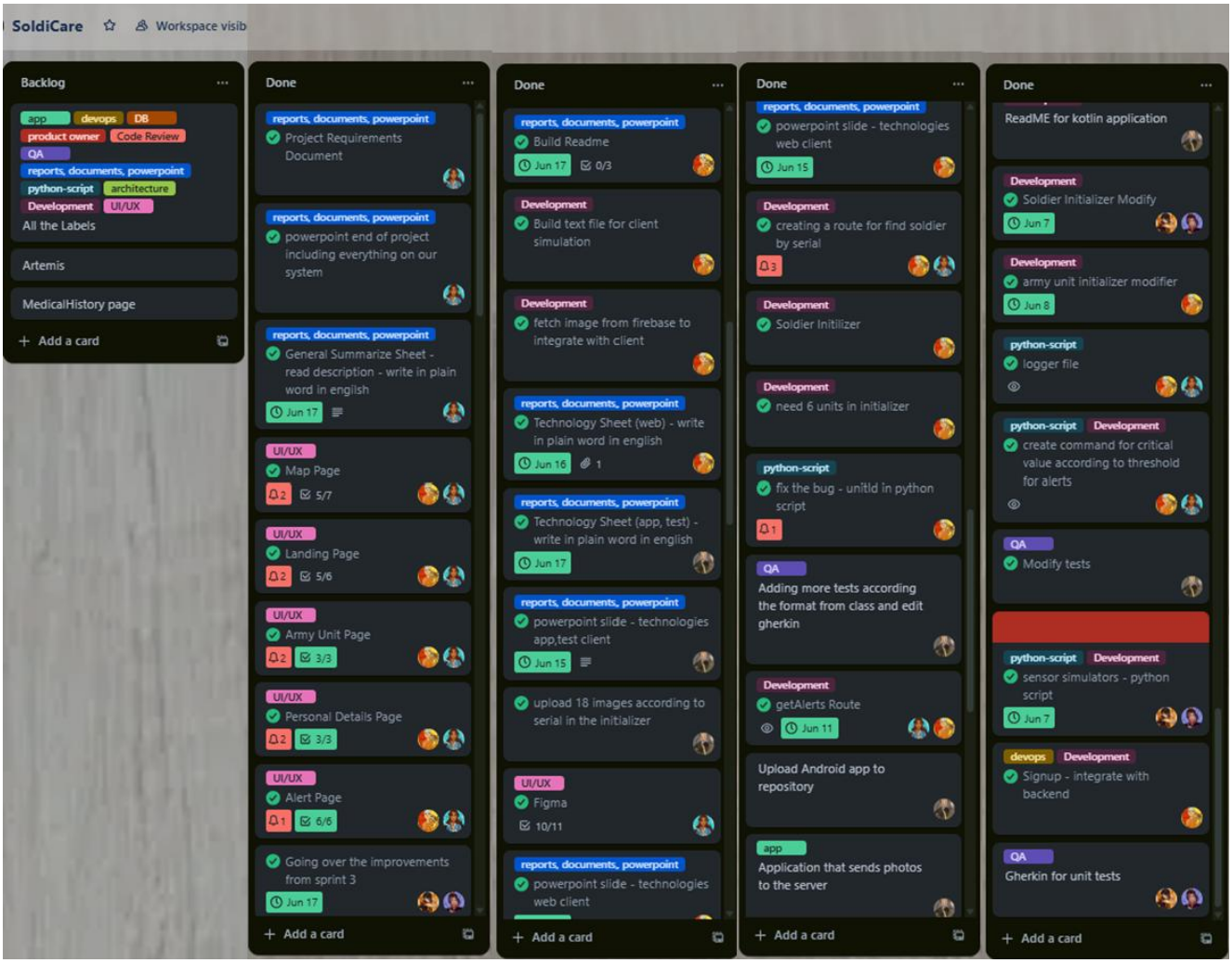
Second State (10/06/2025)



Third State (14/06/2025)



Last State (17/06/2025)



Summary

Points to preserve

- Architectural planning for a system integrating various technologies and cloud-based image storage.
- UI/UX team integration.

Points to improve

- There was a lack of group meetings, which affected communication and coordination among team members.
- Several tasks were not completed on time, leading to delays in development.
- Task distribution was not balanced across the team, resulting in an uneven workload.

Problems encountered

- Not all team members were consistently available, which impacted the progress of certain tasks.

Why did we not complete all the planned work

The following tasks were not completed due to an overload of prior assignments and the outbreak of the war, which significantly affected our timeline and workflow:

- The implementation of Artemis was not completed within this sprint.
- The development of the medical history page was not finalized.

AFEKA
The Academic College of Engineering in Tel Aviv

June 18th, 2025

Integrative semester B, 2025
(10143)

SoldiCare

Ambient Invisible Intelligence

Lecturers:
Deganit Armon, Tom Cohen

System Video Figma

Meet the teem

Itay Chabra
Team Lead & DBA

Sapir Gilany
Scrum Master
& Tech Writer

Team

Neta Ben Mordechai
Product Owner &
System Architecture

Idan Noyshul
Devops

Roi Dor
QA Engineer
& UI/UX

System Overview

Problem Statement & General System Goal

Problem

Lack of real-time medical monitoring in combat zones.

Challenge

Soldiers are not always aware of the deterioration in their health condition.

Consequences

Reduced operational readiness and increased risk to human life.

Objective

Ensure the health and combat readiness of soldiers in the field.

Solution

Advanced monitoring with an intuitive smart interface and instant alerts.

Scope of System

Core Functionality

Real-time health monitoring and alert system

Security and Data Protection

No guaranteed encryption or advanced access control

Operational Constraints

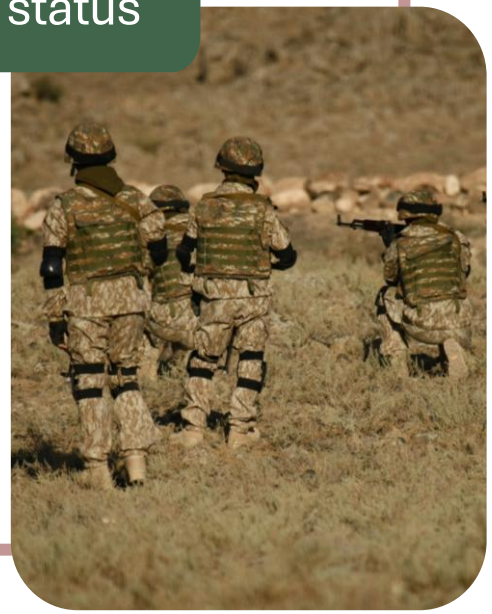
Requires stable connectivity, supplements medical evaluations

Limitations

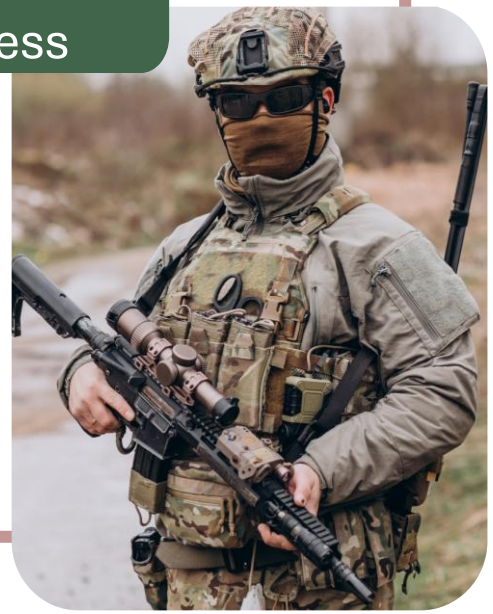
No medical diagnoses, data transmission may be affected by external factors

Understanding the Users

Soldiers
Monitoring their physical and health status



Commanders
Field force management and unit readiness



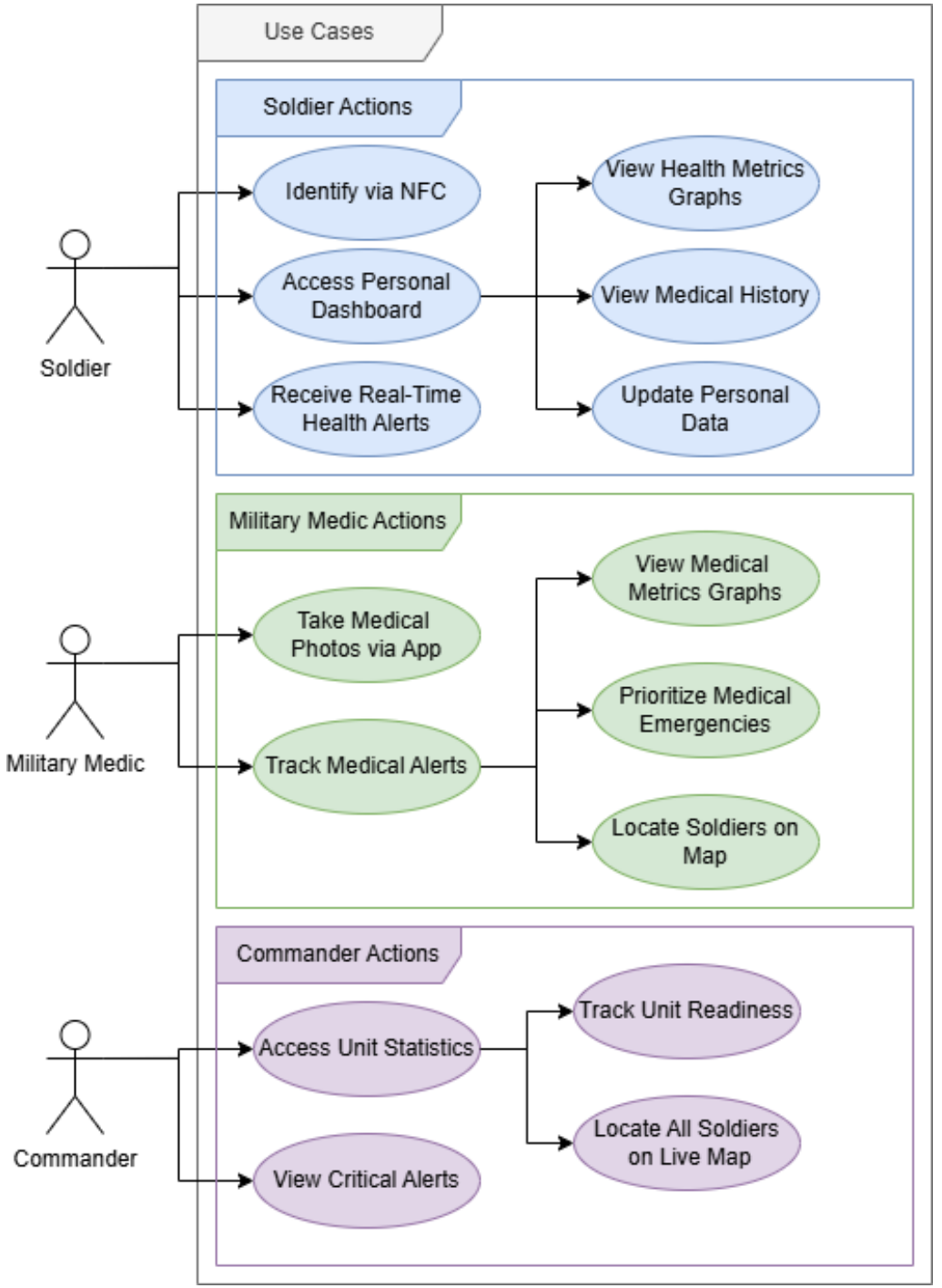
Military doctors
Prioritization of field treatment

Research & Conceptual Planning

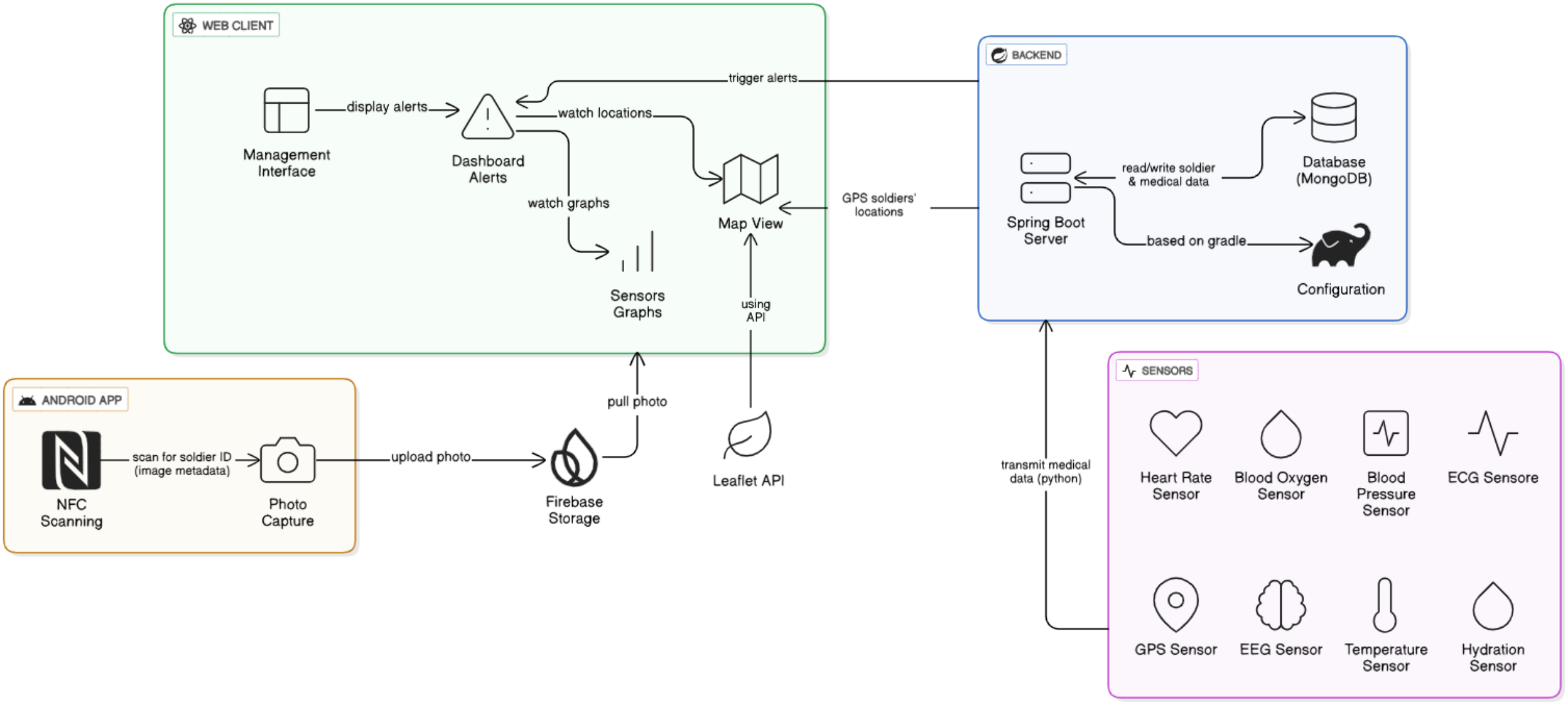
Market Research & Industry Insights

CriterionSystem	IoMT	HRS	Equivital
Type of medical monitoring	Pulse, blood pressure, oxygen saturation, stress	Blood pressure, oxygen saturation, cardiac metrics	Pulse, respiration, skin temperature, movement
Suitability for military use	✓	✗	✓
Data analysis capability	AI	Basic medical	Advanced physiological
Integration with command systems	✓	✗	✓
User interface	Adapted to military terminology	Intuitive for medical treatment	Adapted to field conditions

Use Case Analysis & System Workflow



Core Functionality & System Architecture



Non-Functional Requirements

Usability

- ✓ Visual Data Representation
- ✓ Responsive System Design

Reliability

- ✓ Medical-Grade Alert Classification

Performance

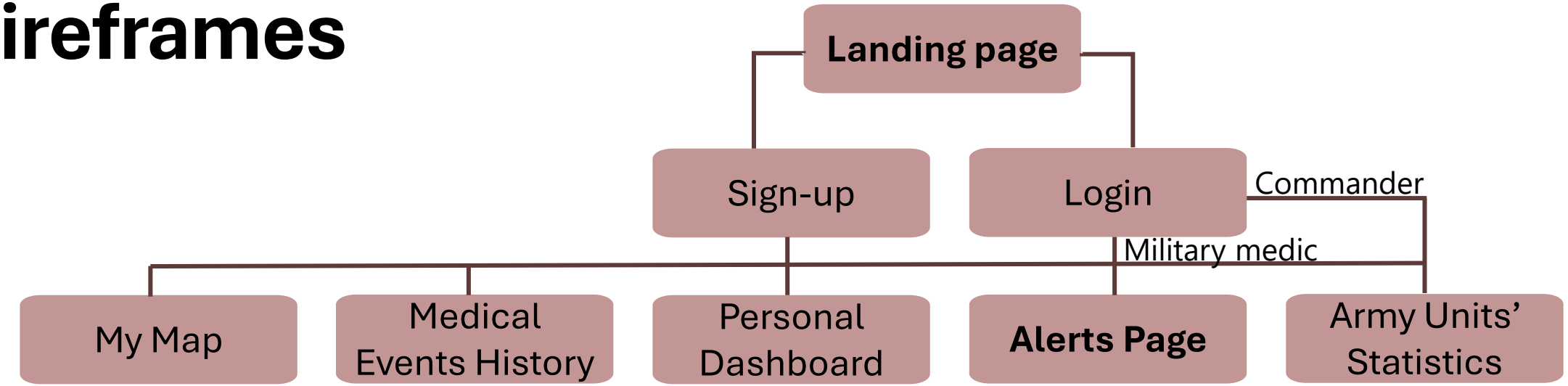
- ✓ Concurrent Session Handling

Supportability

- ✓ Extensibility & Scalability
- ✓ Logging & Debugging Mechanisms

Frontend Development & System Interfaces

User Interface Design & Conceptual Wireframes

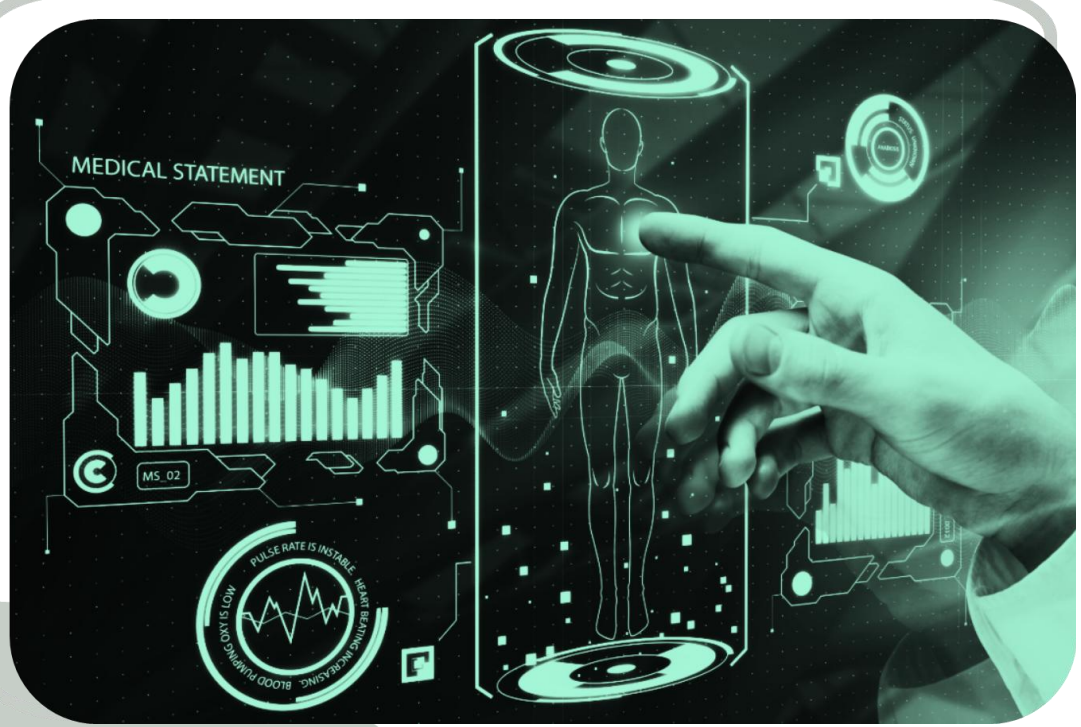


Design Goal

- ✓ Critical medical data monitoring
- ✓ No cumbersome navigation – localized and centralized data
- ✓ Real-time decision support
- ✓ Separation between historical data and current information

Client-Side Functionalities

- ✓ Real-time medical monitoring for soldiers
- ✓ Receives vitals from biometric sensors
- ✓ Smart alerts, graphs, and live map
- ✓ View history and generate unit stats



Client-Side Code Highlights

Custom Hook

Efficiently manages soldier health data for quick access to medical history and alerts

- ✓ Manages soldier and unit state
- ✓ Auto-fetch on data change
- ✓ Parallel fetching using Promise.all
- ✓ Returns structured data object

```
export const useSoldierData = (
  soldiers: Soldier[],
  selectedId: string | null,
  systemID: string,
  email: string
) => {
  const [activeSoldier, setActiveSoldier] = useState<Soldier | null>(null);
  const [soldierUnit, setSoldierUnit] = useState<UnitData | null>(null);
  const [snapshotHistory, setSnapshotHistory] = useState<Snapshot[]>([]);
  const [loading, setLoading] = useState<boolean>(false);

  useEffect(() => {
    const fetchSoldierData = async () => {
      setLoading(true);

      const soldier =      Idan Noyshul, yesterday • custom Hook
        soldiers.find((s) => s.id.objectId === selectedId) ??
        soldiers[0] ??
        null;

      setActiveSoldier(soldier);

      if (soldier) {
        try {
          const [unitRaw, snapshots] = await Promise.all([
            getSoldierUnit(
              soldier.id.systemID,
              soldier.id.objectId,
              systemID,
              email
            ),
            getSoldierSnapshot(
              soldier.id.systemID,
              soldier.id.objectId,
              systemID,
              email
            ),
          ]);
```

Client-Side Code Highlights

Map View

Provides a real-time unit overview with health statuses and hotspot markers.

- ✓ Fetches unit data from API
- ✓ Renders polygons and hotspot markers
- ✓ Handles map clicks (shows coordinates)
- ✓ Integrates unit stats modal

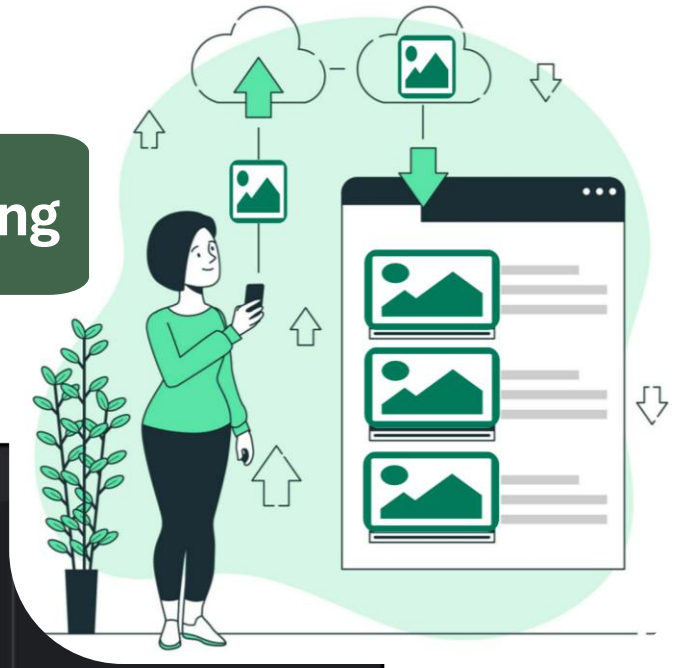
```
33  export default function MapView() {
34    const { systemID, email } = useAppContext();
35    const [clickedPos, setClickedPos] = useState<LatLngTuple | null>(null);
36    const [open, setOpen] = React.useState<boolean>(true);
37    const [allUnits, setAllUnits] = useState<UnitData[]>([]);
38    const [unitsToDisplay, setUnitsToDisplay] = useState<UnitData[]>([]);
39    const [isLoading, setLoading] = useState<boolean>(false);
40    const handleOpen = () => setOpen(true);
41    const handleClose = () => setOpen(false);
42
43    const fetchUnits = async () => {
44      try {
45        setLoading(true);
46        const rawUnits = await getArmyUnits(systemID, email);
47
48        const formattedUnits: UnitData[] = rawUnits.map((unit: any) => ({
49          id: unit.id,       Idan Noyshul, 5 days ago • firebase and integrat
50          name: unit.objectDetails.name,
51          color: unit.objectDetails.color,
52          polygon: unit.objectDetails.polygon,
53          hotspots: [],
54          status: unit.status,
55          severelyWounded: unit.objectDetails.severelyWounded,
56          lightlyWounded: unit.objectDetails.lightlyWounded,
57          immobileUnconscious: unit.objectDetails.immobileUnconscious,
58          dead: unit.objectDetails.dead,
59          combatReadyTroops: unit.objectDetails.combatReadyTroops,
60        }));
61
62        setAllUnits(formattedUnits);
63        setUnitsToDisplay(formattedUnits);
64      } catch (err) {
65        console.error("Error fetching units:", err);
66      } finally {
67        setLoading(false);
68      }
69    };
```

Field Medical Photo Capture App

- ✓ NFC scans to link photos to soldiers
- ✓ Real-time injury documentation by field medics
- ✓ Image storage in Firebase cloud
- ✓ Instant medical team access to uploaded photos

App Code Highlights

Metadata handling



```
StorageMetadata metadata = new StorageMetadata.Builder()
        .setContentType("image/jpeg")
        .setCustomMetadata("SoldierID", scannedNfcId)
        .setCustomMetadata("SystemID", "2025b.Itay.Chabra")
        .setCustomMetadata("captureDate", new java.text.SimpleDateFormat(
                pattern: "yyyy-MM-dd", java.util.Locale.getDefault()).format(new java.util.Date()))
        .setCustomMetadata("captureTime", new java.text.SimpleDateFormat(
                pattern: "HH:mm:ss", java.util.Locale.getDefault()).format(new java.util.Date()))
        .setCustomMetadata("captureDateTime", new java.text.SimpleDateFormat(
                pattern: "yyyy-MM-dd HH:mm:ss", java.util.Locale.getDefault()).format(new java.util.Date()))
        .setCustomMetadata("UserID", "1234567")
        .setCustomMetadata("deviceModel", android.os.Build.MODEL)
        .build();
```

App Code Highlights

Compression with Orientation Correction

Image compression that preserves correct orientation based on metadata.

- ✓ EXIF metadata preservation and manipulation
- ✓ Memory-efficient bitmap processing
- ✓ Configurable compression with quality control

```
private File compressImage(File originalFile) {
    try {
        // Get the original image orientation
        ExifInterface exif = new ExifInterface(originalFile.getAbsolutePath());
        int orientation = exif.getAttributeInt(ExifInterface.TAG_ORIENTATION, ExifInterface.ORIENTATION_NORMAL);

        Log.d(TAG, msg: "Original image orientation: " + orientation);

        // Load the image
        Bitmap bitmap = BitmapFactory.decodeFile(originalFile.getAbsolutePath());

        if (bitmap == null) {...}

        // Rotate the bitmap if needed
        Bitmap rotatedBitmap = rotateBitmapIfNeeded(bitmap, orientation);

        // If rotation created a new bitmap, recycle the original
        if (rotatedBitmap != bitmap) {...}

        // Create compressed file
        File compressedDir = new File(getExternalFilesDir(Environment.DIRECTORY_PICTURES), child: "compressed");
        if (!compressedDir.exists() && !compressedDir.mkdirs()) {...}

        File compressedFile = new File(compressedDir, child: "compressed_" + originalFile.getName());

        // Compress and save
        FileOutputStream fos = new FileOutputStream(compressedFile);

        // Compress with 60% quality (adjust as needed: 0-100)
        boolean success = rotatedBitmap.compress(Bitmap.CompressFormat.JPEG, COMPRESSION_QUALITY, fos);

        fos.flush();
        fos.close();
        rotatedBitmap.recycle(); // Free memory

        if (!success) {...}

        // Set the correct orientation in the compressed file's EXIF data
        ExifInterface compressedExif = new ExifInterface(compressedFile.getAbsolutePath());
        compressedExif.setAttribute(ExifInterface.TAG_ORIENTATION, String.valueOf(ExifInterface.ORIENTATION_NORMAL));
        compressedExif.saveAttributes();

        Log.d(TAG, msg: "Image compressed successfully with correct orientation");
        return compressedFile;

    } catch (Exception e) {...}
}
```

NFC Code Highlight

NDEF tag reading

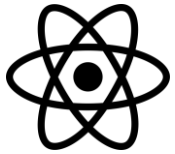
Reads and parses NDEF tags to retrieve encoded soldier identification data.

- ✓ Proper NDEF record parsing with language code handling
- ✓ Understanding of NFC data format specifications
- ✓ Robust error handling without crashing


```
private String readNfcTag(Tag tag) {
    Log.d(TAG, msg: "Reading NFC tag...");

    try {
        // First try to read NDEF data
        Ndef ndef = Ndef.get(tag);
        if (ndef != null) {
            Log.d(TAG, msg: "NDEF tag detected");
            ndef.connect();

            if (ndef.isConnected()) {
                NdefMessage ndefMessage = ndef.getNdefMessage();
                if (ndefMessage != null) {
                    NdefRecord[] records = ndefMessage.getRecords();
                    Log.d(TAG, msg: "Found " + records.length + " NDEF records");

                    for (NdefRecord record : records) {
                        Log.d(TAG, msg: "Processing NDEF record, TNF: " + record.getTnf());

                        if (record.getTnf() == NdefRecord.TNF_WELL_KNOWN &&
                                java.util.Arrays.equals(record.getType(), NdefRecord.RTD_TEXT)) {...}
                        // Also try reading as UTF-8 string directly for some tag types
                        else if (record.getTnf() == NdefRecord.TNF_WELL_KNOWN) {
                            try {
                                String directText = new String(record.getPayload(), StandardCharsets.UTF_8);
                                if (directText.length() > 0 && !directText.contains("\u0000")) {
                                    Log.d(TAG, msg: "Read as direct UTF-8: " + directText);
                                    ndef.close();
                                    return directText;
                                }
                            } catch (Exception e) {
                                Log.d(TAG, msg: "Could not read as direct UTF-8");
                            }
                        }
                    }
                }
                ndef.close();
                Log.d(TAG, msg: "No readable text records found in NDEF message");
            } else {
                Log.d(TAG, msg: "Not an NDEF tag, trying to read tag ID");
            }

            // Fallback: return tag ID as hex string
            byte[] tagId = tag.getId();
            if (tagId != null && tagId.length > 0) {
                String hexId = bytesToHex(tagId);
                Log.d(TAG, msg: "Using tag ID as fallback: " + hexId);
                return hexId;
            }

        } catch (Exception e) {...}

        Log.w(TAG, msg: "Could not read any data from NFC tag");
        return null;
    }
```

Technologies Behind the Frontend

Frontend Web



React + TypeScript

Material UI

React Hooks

Leaflet API

Recharts Library

Image Storage & Metadata Handling

Firebase Storage

ExifInterface

Bitmap & Graphics APIs

NFC Technology

Android NFC API

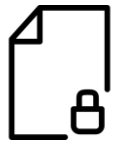
NDEF (NFC Data Format)

Foreground Dispatch System

Android Application

Kotlin

CameraX



FileProvider

Material Design Components

Constraint Layout

Backend Development & System Operations

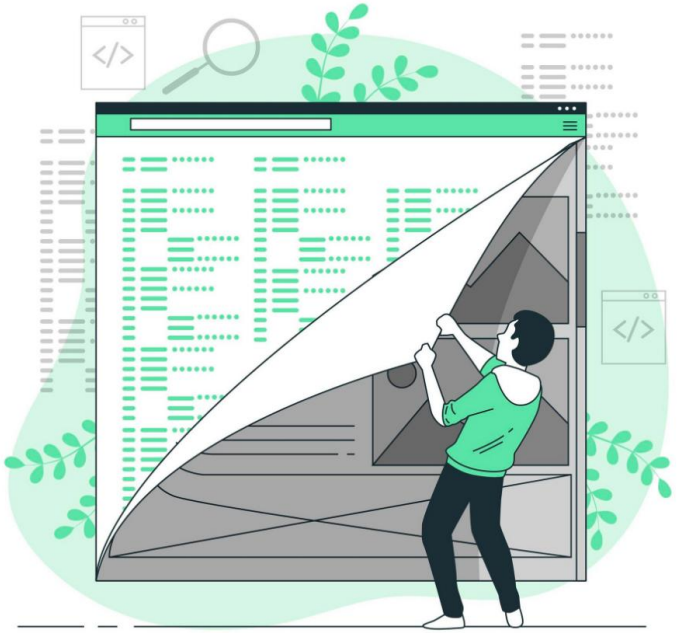
Python-Based Sensor Data Simulation

(for development and POC purposes)

- ✓ Simulates biometric sensor data with randomized values
- ✓ Automatic soldier identification & unit linking
- ✓ Send critical health alerts via command
- ✓ Utilizes Python modules: requests, JSON, logging
- ✓ Runs continuously for real-time simulation testing

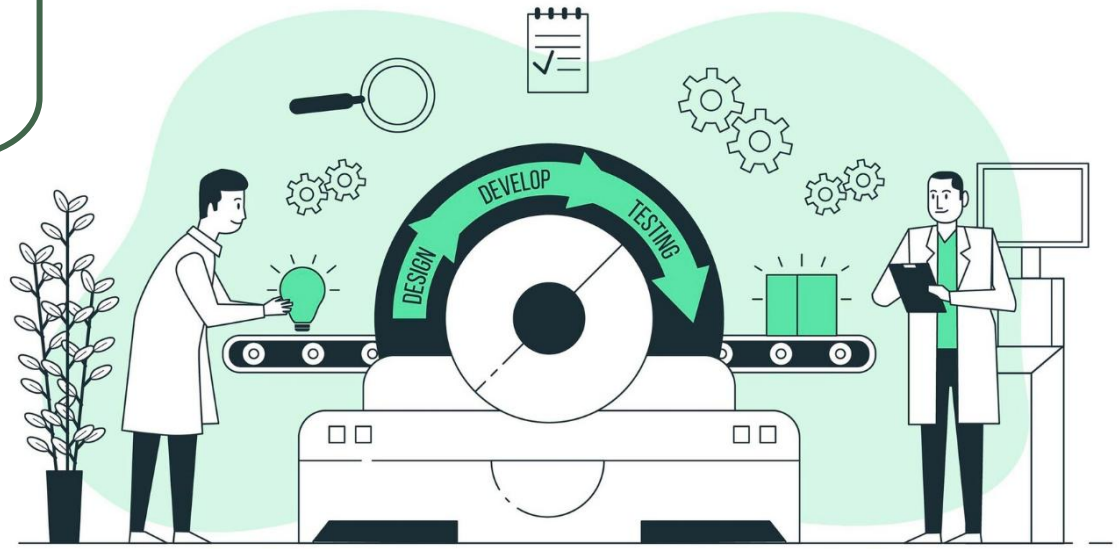
Custom Backend Enhancements for Our System

- ✓ AlertController – RESTful endpoint for managing alerts, enabling retrieval based on user and pagination settings.
- ✓ AlertService – Handles alert retrieval, interacting with CommandService to filter critical commands.
- ✓ Snapshot History – Maintains historical alert records for context and accessibility.
- ✓ Related Soldiers Retrieval – Extracts data on associated soldiers to provide a comprehensive situational overview.



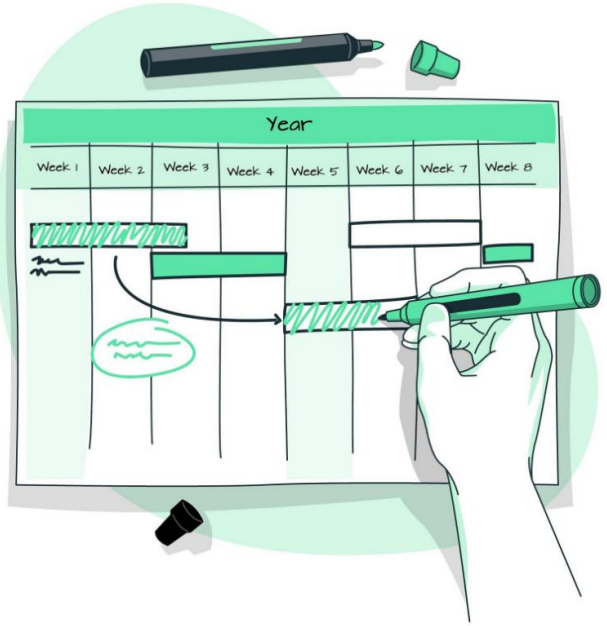
Unit Tests & Quality Assurance

- ✓ Unit & Integration Testing for System Components
- ✓ Mock-based Testing Using Mockito
- ✓ Spring Boot & REST API Validation
- ✓ Structured Gherkin Test Writing (BDD – Behavior-Driven Development)
- ✓ Automated Test Execution & Continuous Validation



Final Review

Kanban Board #1
03/06/2025



SoldiCare | Workspace visible | Board

Backlog

devops DB product owner
Code Review QA
reports, documents, powerpoint
architecture Development
UI/UX
All the Labels

Functionalities tailored to the unique needs of our system

python requests sending jsons

+ Add a card

TO-DO

Development
sensor simulators - python script
Jun 7

Development
Soldier Initializer Modify
Jun 7

Development
army unit initializer modifier
Jun 8

QA
Modify tests

QA Development
Performance Tests

Development
getAlerts Route
Jun 11

reports, documents, powerpoint
Build Readme
0/3

reports, documents, powerpoint
powerpoint end of project including everything on our system

Development
Build text file for client simulation

+ Add a card

In Progress

devops Development
Signup - integrate with backend

UI/UX
Alert Page
3/6

UI/UX
Army Unit Page
2/3

UI/UX
Personal Details Page
0/3

UI/UX
Landing Page
0/3

UI/UX
Map Page
5/7

UI/UX
Figma
9/10

reports, documents, powerpoint
Project Requirements Document

+ Add a card

Done

+ Add a card

Kanban Board #2
10/06/2025

SoldiCare | Workspace visible | Board

Backlog

- app, devops, DB, product owner, Code Review, QA, reports, documents, powerpoint, python-script, architecture, Development, UI/UX — All the Labels
- Functionalities tailored to the unique needs of our system
- python requests sending jsons

+ Add a card

TO-DO

- Upload kotlin app to repository
- Development — ReadME for kotlin application
- python-script — fix the bug - unitId in python script (1)
- Development — need 6 units in initializer
- QA — Adding more tests according the format from class and edit gherkin
- QA, Development — Performance Tests
- reports, documents, powerpoint — Build Readme (0/3)
- reports, documents, powerpoint — powerpoint end of project including everything on our system
- Development — Build text file for client simulation

+ Add a card

In Progress

- app — Application that sends photos to the server
- UI/UX — Alert Page (3/6)
- UI/UX — Army Unit Page (2/3)
- UI/UX — Personal Details Page (0/3)
- UI/UX — Landing Page (4/6)
- UI/UX — Map Page (5/7)
- UI/UX — Figma (10/11)
- reports, documents, powerpoint — Project Requirements Document

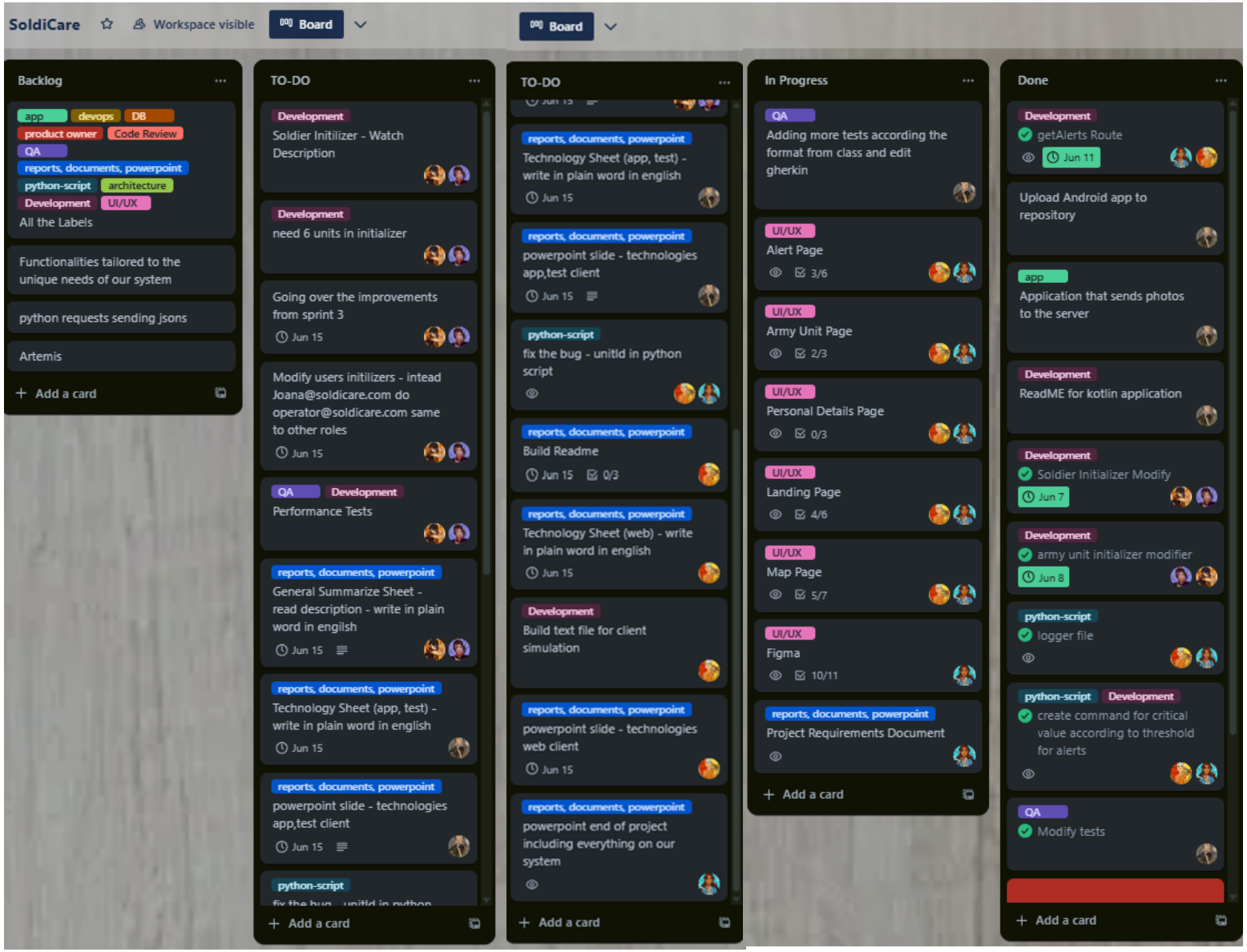
+ Add a card

Done

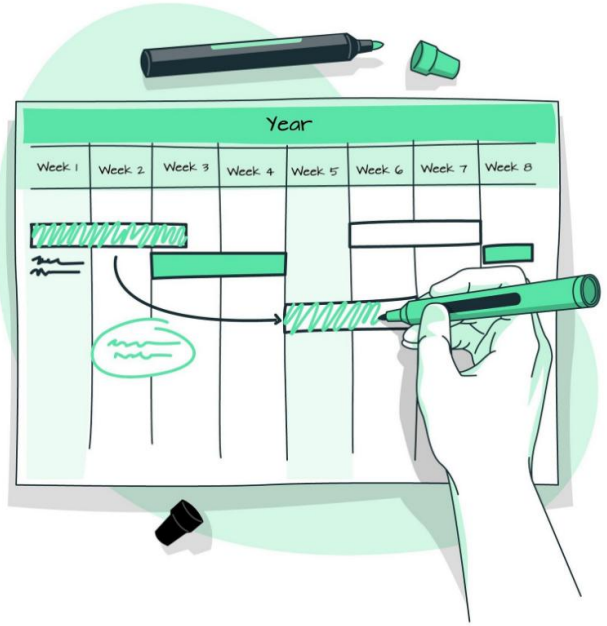
- Development — getAlerts Route (Jun 11)
- Development — Soldier Initializer Modify (Jun 7)
- Development — army unit initializer modifier (Jun 8)
- python-script — logger file
- python-script, Development — create command for critical value according to threshold for alerts
- QA — Modify tests
- python-script, Development — sensor simulators - python script (Jun 7)
- devops, Development — Signup - integrate with backend

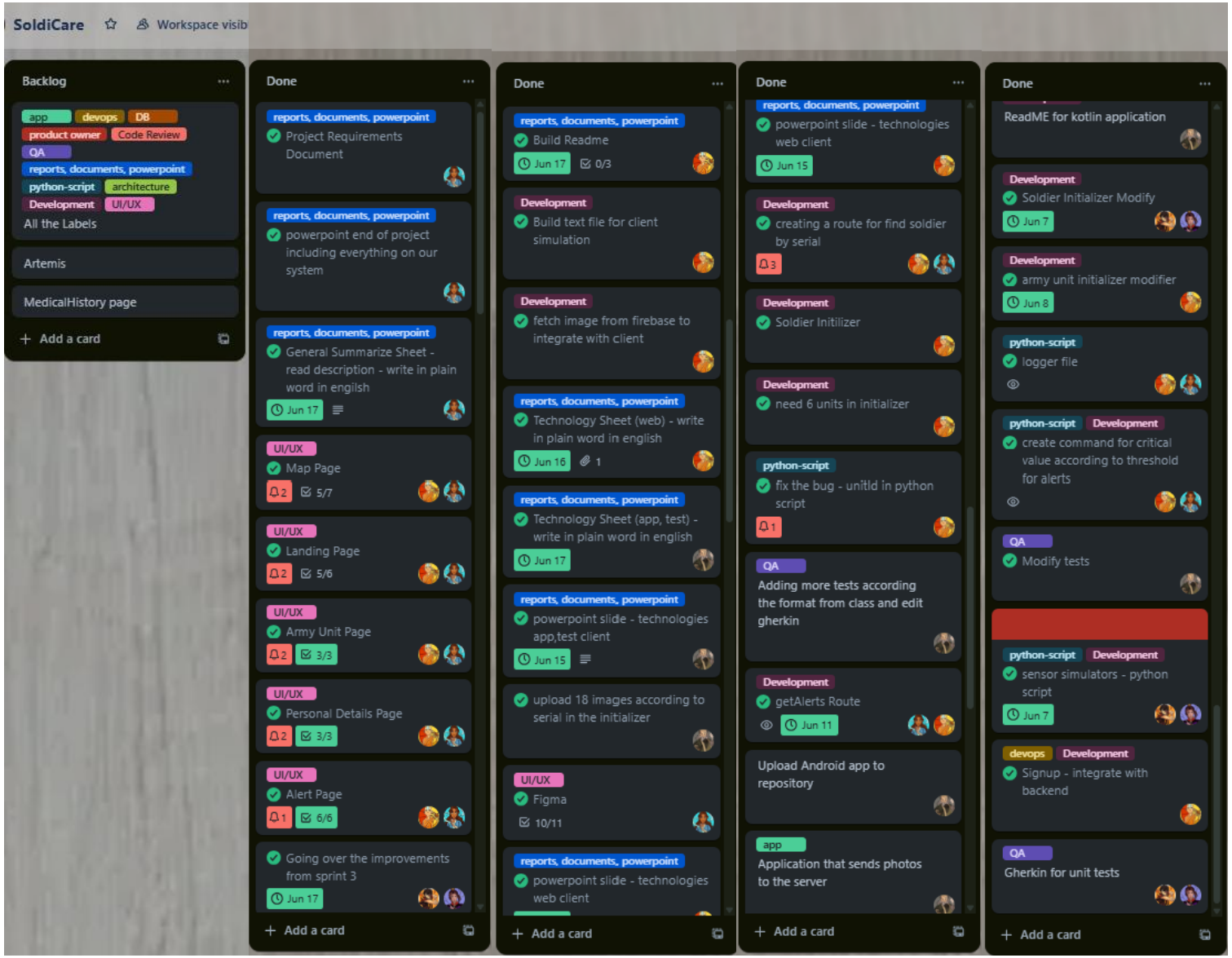
+ Add a card

Kanban Board #3
14/06/2025



Kanban Board #4
17/06/2025





Project Retrospective: Achievements & Improvements

Achievements

- ✓ Effective Kickoff Meetings
- ✓ Utilizing multiple collaboration platforms
- ✓ Structured Git workflow
- ✓ Dividing tasks into small sub-teams
- ✓ Commitment to deadlines and quality
- ✓ Thorough code review processes

Improvements

- ✓ Strengthening team communication and synchronization
- ✓ Encouraging proactive engagement and responsibility
- ✓ Enhancing feedback culture
- ✓ Improving task execution within timelines
- ✓ Fostering innovation and new initiatives

AFEKA
The Academic College of Engineering in Tel Aviv

SoldiCare

Thanks for your attention