# SmartGarden – Documentație Proiect de Licență

## 1. Introducere generală a proiectului

**SmartGarden** este un sistem inteligent de monitorizare și irigare automată a plantelor, conceput pentru a ușura munca iubitorilor de plante și a asigura condiții optime de creștere pentru acestea. Scopul proiectului este de a automatiza procesul de îngrijire a plantelor de apartament (și nu numai) prin combinarea senzorilor de mediu, a unui microcontroler cu conectivitate Wi-Fi și a unei aplicații mobile intuitive. Astfel, SmartGarden monitorizează constant factorii importanți precum umiditatea solului, temperatura, luminozitatea și nivelul apei din rezervor, și decide **când și cât să ude planta**, fără intervenție umană directă, exceptând configurarea inițială și supravegherea prin aplicație.

**Motivația** proiectului pornește de la problemele comune întâmpinate de persoanele ocupate sau care călătoresc des: uitarea udării plantelor la timp sau, dimpotrivă, udarea excesivă. O soluție automatizată elimină **riscul de uscare sau supra-udare** a plantelor, menținând un nivel optim de umiditate în sol conform necesităților fiecărei specii. În plus, monitorizarea continuă a condițiilor de mediu oferă informații valoroase despre sănătatea plantelor și eficiența îngrijirii.

**Beneficiarii** direcți ai SmartGarden sunt pasionații de plante de apartament, florarii, proprietarii de sere mici sau grădini urbane, precum și orice persoane care doresc să se bucure de plante sănătoase fără efort zilnic. Prin accesul de la distanță oferit de aplicația mobilă, utilizatorii pot verifica oricând starea plantelor și pot primi alerte (de exemplu, când rezervorul de apă este pe terminate) – lucru util mai ales când aceștia sunt plecați de acasă. De asemenea, proiectul are relevanță educațională, demonstrând aplicarea conceptelor de **Internet of Things (IoT)** și automatizare în viața de zi cu zi.

În concluzie, SmartGarden își propune să **faciliteze îngrijirea plantelor** prin tehnologie, asigurând un mediu optim de creștere pentru plante și oferind utilizatorilor confort, timp liber și satisfacția unor plante viguroase. În secțiunile următoare este prezentată în detaliu arhitectura sistemului, componentele hardware și software, modul de funcționare, algoritmica de udare, precum și rezultatele testelor practice și direcțiile de dezvoltare viitoare.

## 2. Arhitectură generală SmartGarden

Sistemul SmartGarden are o arhitectură modulară, formată din patru componente software majore, alături de componenta hardware IoT. Separarea pe straturi și module asigură o dezvoltare mai ușoară, posibilitatea înlocuirii sau extinderii unor părți ale sistemului și o mentenanță îmbunătățită. Componentele principale sunt:

* **SmartGarden.Data** – layer-ul de acces la date, care include modelul bazei de date și configurația ORM (Object-Relational Mapping) folosind Entity Framework Core. Acest proiect definește **tabelele și relațiile** din baza de date SQL (prin intermediul claselor de model și a configurărilor asociate) și gestionează conexiunea la baza de date (contextul EF). De asemenea, conține migrațiile pentru crearea și actualizarea schemelor de tabel. Pe scurt, SmartGarden.Data este responsabil de persistența datelor aplicației (utilizatori, plante, citiri de la senzori, log-uri de udare etc.).
* **SmartGarden.Core** – layer-ul de logică de bază (business logic) și definire a entităților de domeniu. În acest proiect, SmartGarden.Core conține modelele aplicației (clasele care reprezintă conceptele cheie precum Plant, SensorReading, Species, SoilType, User etc.) și interfețele pentru serviciile de business. De asemenea, sunt definite DTO-urile (Data Transfer Objects) folosite pentru a transfera date între client, server și baza de date (ex: CreatePlantDto, CreateSensorReadingDto, WateringLogDto). Acest modul central nu are dependențe față de infrastructură (EF sau API), ceea ce îl face ușor de testat și reutilizat. Practic, **SmartGarden.Core definește logica de business și contractele (interfețele) serviciilor**, fiind nucleul aplicației.
* **SmartGarden.API** – layer-ul de prezentare pe server, adică **backend-ul web** expus printr-un API RESTful construit cu ASP.NET Core Web API. Acest proiect referențiază Core (pentru modele și interfețe) și Data (pentru acces la baza de date) și implementează **controlerele web** care gestionează cererile HTTP. Fiecărui controler îi corespunde o resursă/entitate (de exemplu, PlantController gestionează operațiile legate de plante). Logica de manipulare a datelor este realizată în **servicii** (implementări ale interfețelor din Core, cum ar fi PlantService, SensorService, WateringService, definite în SmartGarden.API/Services), pe care controlerele le apelează. Aceste servicii folosesc contextul EF din SmartGarden.Data pentru a citi/scrie în baza de date. SmartGarden.API expune endpoint-urile prin care atât aplicația mobilă, cât și dispozitivul IoT comunică. Tehnologiile folosite includ ASP.NET Core 8, Entity Framework Core cu SQL Server, iar pentru testare și explorare a fost integrat Swagger (Swashbuckle) pentru a documenta și încerca ușor endpoint-urile. De asemenea, API-ul gestionează și autentificarea utilizatorilor (într-o variantă simplificată, bazată pe validarea email/parolă), asigurând că fiecare utilizator are acces doar la plantele și senzorii săi.
* **SmartGarden.App** – componenta de interfață cu utilizatorul, o aplicație mobilă cross-platform dezvoltată în .NET MAUI (Multi-platform App UI). Aceasta este aplicația pe care utilizatorii o instalează pe telefoane (Android/iOS) sau tablete, permițându-le să interacționeze cu sistemul: să configureze dispozitivul, să adauge plante și să vizualizeze datele senzorilor în timp real, să primească notificări și să controleze udarea (manual sau automat). Aplicația comunică cu backend-ul prin Internet, folosind API-ul REST pentru a trimite sau prelua date în format JSON. Arhitectura internă a aplicației mobile este de tip **MVVM (Model-View-ViewModel)**, separând logica de afișare (View, paginile XAML) de logica de prezentare (ViewModel, implementat în C#) și de accesul la date (servicii care apelează API-ul). Astfel, codul este mai structurat, ușor de testat și de extins cu funcționalități noi. Mai multe detalii despre structura și funcționalitățile aplicației mobile sunt prezentate într-o secțiune dedicată.

Alături de aceste patru componente software, arhitectura generală include și **dispozitivul hardware IoT** (ESP32 + senzori și actuatori), care poate fi privit drept a cincea componentă, deși este de natură diferită (firmware-ul microcontrolerului face legătura între lumea fizică și sistemul informatic). Dispozitivul colectează date și execută comenzi de udare, comunicând cu serverul prin rețeaua Wi-Fi.

În diagrama de mai jos este ilustrat sumar modul în care aceste componente interacționează între ele și principalele entități din baza de date (utilizator, plantă, senzori, log-uri udare etc.):

*Figura 1: Diagrama entităților principale din sistemul SmartGarden și relațiile dintre ele (utilizatorii dețin una sau mai multe plante, fiecare plantă are citiri de senzori și log-uri de udări; există tabele de referință pentru specii de plante și tipuri de sol, iar preferințele utilizatorului includ setări globale precum activarea udării automate și pragul de umiditate).*

După cum se observă în Figura 1, modelul de date este unul relațional, construit în SQL Server. Principalele entități sunt:

* **User** – reprezintă utilizatorul aplicației (cheie primară UserId). Stochează informații precum email, parolă (criptată) și nume. Există o relație de **1 la 1** între User și UserSetting (fiecare utilizator are un set de preferințe).
* **UserSetting** – preferințele utilizatorului (identificate tot prin UserId). Conține setări globale, cum ar fi: dacă udarea automată este activată, pragul de umiditate al solului sub care se udă automat planta, și intervalul de citire a datelor în minute. Implicit, udarea automată este activă (AutoWateringEnabled = true), pragul de umiditate este 30% și senzorii trimit date la fiecare 15 minute.
* **Species** – tabel cu specii de plante (cheie primară SpeciesId), conținând denumirea comună și științifică a plantei, precum și parametri recomandați pentru acea specie (ex. interval de udare recomandat, limite optime de temperatură, umiditate, lumină etc.). De exemplu, Ficus lyrata (ficusul cu frunze liră) poate avea asociați parametri ideali de mediu. Această informație este utilă la configurarea plantei în sistem (preluând praguri implicite adaptate speciei).
* **SoilType** – tipuri de sol (cheie SoilTypeId), cuprinzând caracteristici ale solului în care este planta (ex. sol universal, sol de cactuși, sol argilos etc.). În sistemul nostru, tipul de sol poate influența frecvența și cantitatea de apă necesară. De asemenea, calibrările senzorului de umiditate a solului pot fi stocate per tip de sol (fiindcă un sol nisipos se comportă diferit față de unul argilos în privința retenției de apă).
* **Plant** – reprezintă o plantă reală monitorizată și udată de SmartGarden (cheie PlantId). Fiecare plantă este asociată unui User (relație *multe la 1* către User), unei Species și unui SoilType. În plus, sunt stocate detalii precum: numele plantei (sau porecla dată de utilizator), camera sau locul unde se află planta (RoomName), dacă este plasată indoor sau outdoor (IsOutdoor) și data la care a fost adăugată planta (sau data plantării). Aceste informații ajută la organizarea plantelor în aplicație (filtrare după cameră sau interior/exterior) și la adaptarea deciziilor (o plantă de exterior poate avea alte condiții).
* **SensorReading** – reprezintă o citire a senzorilor pentru o anumită plantă, adică un set de valori de la senzori preluate la un moment de timp (cheie Id). Câmpurile principale includ timestamp (data/ora), temperatura aerului (Temperature), umiditatea aerului (Humidity), umiditatea solului (SoilMoisture), nivelul de lumină (LightLevel) și calitatea aerului (AirQuality). Ultimele două pot fi opționale (în funcție de senzorii disponibili – de exemplu, s-ar putea ca versiunea curentă a dispozitivului să nu includă efectiv un senzor de calitate a aerului, dar câmpul există pentru viitoare extinderi). Fiecare citire este legată de o Plant (relație *multe la 1*). Sistemul înregistrează periodic astfel de citiri, iar aplicația mobilă le poate folosi pentru a afișa istoricul și grafice (cum ar fi evoluția valorilor în ultima săptămână).
* **WateringLog** – jurnalul de udare (cheie WateringId). Fiecare înregistrare indică o udare efectuată, stocând planta vizată (relație cu Plant), data/ora (CreatedAt), durata udării (în secunde) și modul în care a fost declanșată (Mode) – de exemplu, *manual* (la comanda utilizatorului) sau *automat* (decizie luată de sistem). Aceste log-uri permit utilizatorului să revadă când și cât a fost udată planta, oferind transparență și ajutând la analizarea consumului de apă.

În ansamblu, arhitectura generală SmartGarden este una de tip **client-server**: dispozitivul IoT și aplicația mobilă acționează ca clienți (producând date de senzori sau cerând informații), în timp ce serverul (API-ul) acționează ca intermediar și centru de control, stocând datele și aplicând regulile de business (de exemplu, dacă udarea automată este activă și umiditatea scade sub prag, sistemul decide udarea). Această separare permite, de exemplu, ca mai mulți utilizatori să folosească aplicația cu propriile lor plante/dispozitive simultan, serverul gestionând izolat datele fiecăruia. Comunicarea se face prin protocol HTTP(S) cu date JSON, ceea ce asigură interoperabilitate și simplitate în dezvoltare.

## 3. Componenta hardware

Componenta hardware a SmartGarden constă în principal din microcontrolerul **ESP-WROOM-32** (modulul ESP32) și o serie de senzori și actuatori conectați la acesta, montați împreună pe un dispozitiv dedicat (un **kit IoT de udare automată**). Rolul hardware-ului este de a **măsura parametrii de mediu relevanți** pentru plantă și de a executa **udarea plantei** prin controlul unei pompe de apă, pe baza deciziilor locale sau a comenzilor primite de la server.

**Microcontrolerul ESP32** – creierul dispozitivului – este o placă de dezvoltare cu un procesor dual-core pe 32 de biți, frecvență de până la 240MHz, dotată cu Wi-Fi 2.4GHz integrat și Bluetooth, precum și cu numeroase intrări/ieșiri (GPIO) analogice și digitale. Alegerea ESP32 (varianta cu modul Wi-Fi ESP-WROOM-32) a fost motivată de necesitatea conectării wireless la internet (pentru a comunica cu serverul) și de posibilitatea de a citi senzori analogici (unele măsurători, precum umiditatea solului sau nivelul apei, se realizează pe pini analogici). În plus, ESP32 oferă putere de procesare suficientă pentru logica încorporată (inclusiv eventuale calcule sau implementarea unui algoritm simplu de decizie) și eficiență energetică bună (poate rula continuu alimentat de la rețea sau de la o baterie cu backup solar, dacă s-ar extinde proiectul).

**Senzorii folosiți** în SmartGarden includ:

* **Senzor de umiditate a solului** – un senzor capacitiv de umiditate a solului (spre deosebire de cei rezistivi, are avantajul că nu corodează așa ușor în timp). Acest senzor constă într-o sondă ce se inserează în pământul din ghiveci; el măsoară umiditatea relativă a solului prin modificarea capacității în funcție de conținutul de apă. Ieșirea sa este un semnal analogic (o tensiune proporțională cu umiditatea). ESP32 citește această tensiune printr-un pin analogic (ADC) și obține astfel o valoare brută, care va fi convertită într-o valoare procentuală de umiditate a solului prin **calibrare**. Calibrarea presupune determinarea valorii citite de senzor în **sol uscat complet** (0% umiditate) vs. **sol saturat cu apă** (100% umiditate) și interpolarea liniară între aceste extreme. Aplicația ghidează utilizatorul în procesul de calibrare a senzorului de sol, solicitând plasarea acestuia în pământ uscat și apoi în pământ umed și așteptând câteva secunde pentru citiri stabile. Valorile obținute sunt fie stocate ca parametri în dispozitiv (pentru conversia ulterioară a citirilor), fie încărcate în server asociat unui anumit SoilType. După calibrare, dispozitivul poate raporta o **umiditate a solului în procente (%)**, ceea ce este mult mai intuitiv pentru utilizator (ex.: „46% Soil Moisture” în aplicație). Senzorul de umiditate a solului este esențial pentru a decide când planta are nevoie de apă.
* **Senzor de lumină (Luminozitate)** – un fotorezistor (LDR) sau un senzor de lumină digital (precum BH1750) care măsoară nivelul de iluminare ambientală în zona plantei. În versiunea curentă a proiectului s-a utilizat un *fotorezistor* simplu împreună cu un divizor de tensiune, oferind o măsurare analogică a luminii ambientale. Acesta permite clasificarea luminii în mediu: de exemplu, dacă planta primește **lumină directă puternică** vs. **lumină indirectă moderată** vs. **umbră**. Deoarece fiecare senzor LDR are caracteristici ușor diferite, a fost implementată și pentru acesta o procedură de **calibrare pe 5 niveluri**, ghidată de aplicație: utilizatorul este rugat să acopere senzorul (nivel “Dark”), apoi să îl expună la **lumină slabă**, **lumină medie indirectă**, **lumină puternică indirectă** și **lumină directă**; la fiecare pas utilizatorul confirmă în aplicație, iar dispozitivul înregistrează valoarea citită ca “completed” pentru nivelul respectiv. Acest proces creează repere pentru clasificarea luminii. Ulterior, citirile curente pot fi exprimate calitativ (ex.: “78 – Light” care corespunde poate categoriei lumina medie) sau pot fi convertite în lumeni relativi. Lumina este importantă de monitorizat deoarece necesarul de apă al plantei poate depinde de cantitatea de lumină (plantele în soare puternic usucă pământul mai repede). În MVP, luminozitatea este înregistrată și afișată utilizatorului, dar nu influențează direct algoritmul de udare (va fi luată în considerare la extinderile ulterioare, de exemplu pentru a oferi **sfaturi** privind amplasarea plantei sau ajustarea udării pe baza intensității luminoase).
* **Senzor de temperatură și umiditate a aerului** – un senzor digital combinat, de tipul **DHT22** (cunoscut și ca AM2302) sau similar, care oferă simultan temperatura ambiantă (în °C) și umiditatea relativă a aerului (%). Acest senzor comunică cu ESP32 pe un pin digital folosind un protocol OneWire simplificat, trimițând valorile măsurate la fiecare interogare. În SmartGarden, senzorul DHT22 este plasat în apropierea plantei, măsurând microclimatul acesteia. Datele culese (Temperature, Humidity) sunt transmise către backend la fiecare citire și pot fi vizualizate în aplicație. De exemplu, în ecranul de detalii ale plantei pot apărea valori precum **24°C (Temperatura)** și **umiditate aer 55%** (dacă am avea afișare pentru aceasta). Acești parametri ajută utilizatorul să înțeleagă condițiile generale în care se află planta; deși nu sunt direct folosiți în decizia de udare, extremele de temperatură pot stresa planta (iar aplicația poate avertiza utilizatorul dacă temperatura iese din intervalul optim cunoscut pentru specia respectivă, funcționalitate prevăzută ca extensie). Menținerea unei evidențe a temperaturii și umidității aerului pe termen lung poate oferi indicii și despre mediul plantei (de exemplu, dacă umiditatea aerului este foarte scăzută, unele plante tropicale ar putea necesita pulverizare cu apă – acest aspect putând fi menționat în secțiunea de Tips & Tricks).
* **Senzor de nivel al apei în rezervor** – un senzor care detectează câtă apă a rămas în rezervorul folosit pentru udare. Este un element critic pentru a evita situațiile în care sistemul încearcă să ude planta, dar rezervorul este gol (ceea ce ar putea duce la funcționarea în gol a pompei și posibila defectare a acesteia, plus că planta nu primește apă). În implementarea SmartGarden, s-a folosit un **senzor de nivel de apă analogic**, care constă într-o sondă verticală cu mai multe contacte (sau un circuit rezistiv gradat) introdusă în rezervor. Senzorul oferă o tensiune analogică ce variază în funcție de nivelul de apă (practic măsoară conductivitatea prin contactele acoperite de apă). Sistemul realizează o calibrare în doi pași a acestui senzor: **nivel minim** (rezervor gol) și **nivel maxim** (rezervor plin). În timpul configurării inițiale, utilizatorul este instruit să golească rezervorul (senzorul citește valoarea minimă, marcând “Min Completed”), apoi să umple rezervorul cu apă și să confirme calibrarea maximului (“Max Calibrate”). Astfel, dispozitivul poate converti ulterior citirile brute în procentaj de umplere a rezervorului. În aplicație, la ecranul plantei, este afișat nivelul apei sub forma unui procent și eventual un mesaj de avertizare când nivelul e foarte scăzut – de exemplu *“Please fill the water tank! 10% Water tank”* indică faptul că rezervorul mai are doar 10% apă. Senzorul de apă previne rularea uscată a pompei: sistemul nu va porni pompa (sau o va opri imediat) dacă detectează că nivelul este sub un prag minim (de ex. sub 5-10%). De asemenea, aplicația va notifica utilizatorul că trebuie reumplut rezervorul.
* **(Opțional) Senzor de calitate a aerului** – proiectul a prevăzut posibilitatea integrării unui senzor de gaz (precum MQ-135 sau BME680) care să detecteze calitatea aerului sau prezența unor compuși volatili. Deși schema bazei de date include un câmp pentru AirQuality în citirile de senzori, în versiunea curentă hardware-ul nu include acest senzor (focusul fiind pe parametrii principali ce influențează udarea). Aceasta rămâne o extensie potențială – un senzor de calitate a aerului ar putea oferi informații despre ventilație sau prezența substanțelor care pot afecta planta (fumul, de exemplu).

În afara acestor senzori, componenta hardware include un **actuator** principal:

* **Pompa de apă** – o pompiță electrică submersibilă de 5V sau 12V (în funcție de design, alimentată corespunzător) plasată într-un mic rezervor de apă. Pompa este conectată la microcontroler printr-un tranzistor sau releu (ESP32 nu poate alimenta direct pompa, deci se folosește un tranzistor MOSFET pentru a comuta pompa folosind o ieșire digitală a ESP32). Când sistemul decide udarea, ESP32 activează pompa pentru o durată determinată de timp, trimițând apă printr-un furtun către ghiveci. La finalul duratei, pompa este oprită. Pompa este calibrată astfel încât durata de funcționare să livreze aproximativ volumul de apă dorit – de exemplu, 5 secunde de pompare pot livra ~50 ml de apă, suficient pentru a ridica umiditatea solului în intervalul optim, fără a inunda planta. Desigur, durata poate varia în funcție de mărimea ghiveciului și necesarul speciei, parametri ce pot fi ajustați în software.

**Alimentarea și conexiunea Wi-Fi:** Dispozitivul SmartGarden este alimentat de obicei printr-un adaptor de 5V (microUSB) conectat permanent, având un consum redus. Conectarea la rețeaua Wi-Fi se face pe banda de 2.4GHz (ESP32 nu suportă 5GHz), lucru menționat utilizatorului în aplicație. La prima configurare, microcontrolerul pornește fie în modul **Access Point** (AP) propriu, fie într-un mod de așteptare a configurației – utilizatorul va conecta telefonul la AP-ul dispozitivului sau va folosi Bluetooth (dacă implementat) pentru a transmite credentialele Wi-Fi. În SmartGarden, s-a optat pentru metoda AP: dispozitivul intră în modul AP cu un SSID de forma “SmartGarden-XXXX”, iar aplicația mobilă ghidează utilizatorul să se conecteze la această rețea și să introducă SSID-ul și parola rețelei casnice (2.4GHz) la care dispozitivul trebuie să se conecteze. Ulterior, modulul ESP32 restartează și se alătură rețelei Wi-Fi a utilizatorului. Această secvență face parte din procesul de onboarding din aplicație: *“Connect your device to Wi-Fi (make sure it’s 2.4GHz)”*. Odată conectat la internet, dispozitivul comunică cu serverul SmartGarden.API.

**Firmware și logica locală:** Microcontrolerul ESP32 este programat (firmware scris în C/C++ în ecosistemul Arduino sau PlatformIO) pentru a realiza următoarele sarcini ciclice: 1. **Citirea senzorilor** – la un interval configurat (implicit la fiecare 15 minute, conform DataReadIntervalMin din UserSetting), ESP32 colectează date de la toți senzorii: analogice (umiditate sol, lumină, nivel apă) și digitale (DHT22 pentru temp/umiditate aer). Datele brute sunt prelucrate: se aplică factorii de calibrare (depozitați în memoria internă după procesul de calibrare inițială) pentru a converti valorile în unități reale (de exemplu, calcularea procentului de umiditate a solului din valoarea ADC citită). 2. **Trimiterea datelor către backend** – dispozitivul formează o cerere HTTP (POST) către endpoint-ul API-ului de backend (ex: /api/sensor), în care trimite un obiect JSON conținând ID-ul plantei și valorile senzori măsurate (temperatura, umiditate aer, umiditate sol %, lumină, calitate aer). O schemă a JSON-ului transmis ar putea fi:

{  
 "plantId": 123,  
 "temperature": 24.1,  
 "humidity": 53.2,  
 "soilMoisture": 28.5,  
 "lightLevel": 76.0,  
 "airQuality": null  
}

Backend-ul primește aceste date și le stochează ca o nouă înregistrare SensorReading în baza de date. În cadrul răspunsului de la server, sistemul poate trimite și instrucțiuni către dispozitiv (de exemplu, să pornească udarea dacă se constată nevoie imediată – detaliem în secțiunea **Fluxul de date**). Fragment de cod (pseudo-C++) ilustrativ pentru trimiterea datelor:

float soil = readSoilSensor(); // citire senzor sol (convertit in %)  
float light = readLightSensor(); // citire LDR (convertit in %)  
float waterLvl = readWaterSensor(); // nivel apa (%)  
float temp = dht.readTemperature();  
float humid = dht.readHumidity();  
// Construim payload JSON  
String payload = "{";  
payload += "\"plantId\":" + String(plantId) + ",";  
payload += "\"temperature\":" + String(temp) + ",";  
payload += "\"humidity\":" + String(humid) + ",";  
payload += "\"soilMoisture\":" + String(soil) + ",";  
payload += "\"lightLevel\":" + String(light) + ",";  
payload += "\"airQuality\": null"; // exemplu, daca nu avem senzor  
payload += "}";  
// Trimitem datele catre API (HTTP POST)  
httpClient.begin(apiURL + "/api/sensor");  
httpClient.addHeader("Content-Type", "application/json");  
int httpCode = httpClient.POST(payload);

(Codul de mai sus ilustrează conceptual trimiterea datelor; în implementare reală, există considerente suplimentare precum tratarea erorilor de rețea, reconectarea Wi-Fi la nevoie, etc.)

1. **Primirea comenzilor de la backend** – După trimiterea datelor, dispozitivul poate primi un răspuns JSON de la server. În MVP, s-a folosit un mecanism simplu: serverul include în răspuns un câmp care indică dacă trebuie inițiată udarea sau nu. De exemplu, răspunsul la POST-ul de mai sus poate fi:

* {  
   "waterNow": true,  
   "waterDurationSec": 5  
  }
* dacă serverul decide că trebuie udat (sau dacă utilizatorul a declanșat manual udarea – situație explicată ulterior). Dispozitivul parsează răspunsul și, dacă waterNow este true, pornește pompa pentru durata specificată. Acest **răspuns imediat** al serverului este util pentru a minimiza întârzierea între detectarea condiției de udare și acțiunea propriu-zisă. În plus, în modul manual, dacă utilizatorul apasă un buton de „Udă acum”, serverul poate seta un flag ce va fi prins la următoarea comunicare a dispozitivului.

1. **Controlul udării (actuare)** – Indiferent dacă decizia de udare vine din server sau e luată local, ESP32 va executa comanda astfel: activează pinul de control al pompei (prin tranzistor) și menține pompa pornită pentru un anumit interval. Durata poate fi fixă (ex. 5 secunde) sau calculată în funcție de cât de uscat e solul (algoritm extins). După udare, pompa este oprită și se înregistrează un eveniment de udare (într-o variabilă locală). Imediat sau la următoarea raportare de date, dispozitivul trimite către server și un **WateringLog** nou, pentru evidență. De exemplu, ar putea face un POST la /api/watering cu datele:

* {  
   "plantId": 123,  
   "durationSec": 5,  
   "mode": "AUTO"  
  }
* dacă udarea a fost automată (sau "MANUAL" în caz contrar). Astfel, serverul salvează log-ul de udare. (În implementarea actuală, trimiterea log-ului de udare se face în cadrul aceluiași endpoint de sensor sau printr-un endpoint dedicat, în funcție de design; important este că nu se pierde evenimentul).

1. **Loop continuu și altele** – Dispozitivul rulează într-o buclă continuă (loop() în Arduino C++) și, pe lângă pașii de mai sus, se ocupă de menținerea conexiunii Wi-Fi (reconectare automată dacă se pierde), eventual de semnalizarea prin LED-uri (de exemplu, un LED pe ESP32 poate clipi când se trimite o citire sau se efectuează o udare, pentru diagnostic vizual local) și de reacții rapide la comenzi. De pildă, dacă utilizatorul are acces fizic la dispozitiv, ar putea exista un buton de *manual override* care, dacă e apăsat, pornește pompa (util pentru testare sau udare manuală fără aplicație).

Rezumatul fluxului hardware: **senzorii trimit date** -> microcontrolerul le prelucrează -> **microcontrolerul comunică periodic cu serverul** (trimite date, primește eventual comenzi) -> **microcontrolerul controlează pompa** pe baza logicii locale sau a comenzilor primite -> repetă. Astfel, planta primește apă la momentul potrivit, iar utilizatorul poate oricând vedea, prin aplicație, valorile curente (transmise prin server) și starea sistemului.

Merită menționat că mare parte din **inteligența sistemului este partajată** între dispozitivul hardware și backend. În varianta implementată, deciziile de udare automată pot fi luate de microcontroler (de exemplu, dacă senzorul de sol scade sub prag și udarea automată e activată, microcontrolerul ar putea porni imediat pompa, fără să aștepte confirmare de la server), însă pentru simplitate și pentru a păstra controlul centralizat (și logurile complete), s-a optat ca microcontrolerul să consulte serverul înainte de a uda. Aceasta permite introducerea ușoară a unor reguli suplimentare de către server (de exemplu, să nu ude dacă deja s-a udat de X ori azi, sau să țină cont de prognoza meteo în cazul plantelor outdoor – extensii viitoare). Totuși, microcontrolerul are și elemente de protecție locală: nu va executa o udare dacă senzorul de nivel de apă indică rezervor gol (chiar dacă serverul ar cere) și are un watchdog intern care, în lipsa conexiunii la internet, **trece pe modul autonom** – adică dacă nu poate comunica cu serverul o perioadă, va lua singur decizia de udare bazându-se doar pe pragul local de umiditate. În acest fel, planta este protejată și în scenarii de rețea picată.

În concluzie, componenta hardware SmartGarden – ESP32 și ansamblul de senzori – formează un **dispozitiv IoT integrat**, capabil să perceapă mediul plantei și să acționeze asupra acestuia (prin udare). Calibrarea atentă a senzorilor asigură acuratețea datelor colectate, iar mecanismele de protecție (senza apă în rezervor, reconectare la Wi-Fi) conferă robusteză sistemului. Această bază hardware lucrează mână în mână cu software-ul (atât pe server, cât și pe mobil) pentru a realiza scopul final: plante mai sănătoase cu efort minim din partea utilizatorului.

## 4. Fluxul de date

Fluxul de date în sistemul SmartGarden descrie modul în care informațiile circulă de la senzori până la utilizator și cum comenzile utilizatorului se propagă în sens invers, de la aplicație către actuatori. Vom descrie acest flux pas cu pas, evidențiind interacțiunea dintre componente.

**4.1. De la senzori la baza de date și aplicație (sens „în amonte”)**:

1. **Colectarea datelor de la senzori** – După cum s-a menționat în secțiunea hardware, microcontrolerul citește periodic valorile senzorilor (umiditate sol, lumină, temperatură, umiditate aer, nivel apă). Implicit, aceste citiri au loc la fiecare *15 minute*, însă intervalul este configurabil (stocat în UserSetting.DataReadIntervalMin și ajustabil de utilizator dacă dorește citiri mai frecvente sau mai rare). În plus față de aceste citiri programate, sistemul poate efectua *citiri adiționale* la cerere, de exemplu imediat după o udare (pentru a actualiza starea) sau atunci când utilizatorul deschide aplicația și solicită refresh manual (dacă a fost implementată o astfel de funcție).
2. **Transmiterea datelor către backend** – ESP32, conectat la internet prin Wi-Fi, trimite datele culese către server, folosind o cerere HTTP POST la un endpoint dedicat (ex: POST /api/sensor sau /api/plant/{id}/sensor). În această cerere, sunt incluse: identificatorul plantei (astfel încât serverul să știe pentru ce plantă sunt valorile) și valorile numerice ale senzorilor. Odată ce serverul primește datele, acesta:
3. Creează o nouă instanță SensorReading în baza de date, populând câmpurile cu valorile transmise (timestamp-ul este pus automat, de exemplu ora primirii datelor).
4. Salvează înregistrarea și confirmă succesul operației. Această confirmare este dată printr-un răspuns HTTP 200 OK către dispozitiv.

În mod normal, backend-ul nu doar stochează datele, ci și **analizează instantaneu** dacă trebuie luată vreo acțiune. De exemplu, imediat după ce salvează o citire, poate verifica: “Udarea automată e activă pentru utilizatorul/planta aceasta? Dacă da, umiditatea solului citită (ex: 28.5%) este sub pragul setat (ex: 30%)? Apa din rezervor e suficientă?”. Dacă toate condițiile pentru udare sunt îndeplinite, serverul poate decide să trimită un semnal de udare înapoi dispozitivului (vezi pasul 3). Totodată, dacă nivelul apei este foarte scăzut, serverul poate marca în baza de date această situație (ex: un flag pentru plantă “water\_tank\_low = true”) și poate genera o **notificare** către aplicație (ex: push notification “Rezervorul de apă pentru planta X este gol – reumpleți-l!”). În MVP, notificările push nu au fost integrate, însă aplicația verifică starea rezervorului când afișează detalii plantei și arată un mesaj vizibil *“Please fill the water tank!”* când nivelul este sub un anumit prag.

1. **Decizii de udare automată (la server)** – (Pas opțional, în funcție de implementare). După cum am discutat, există două moduri de a implementa udarea automată: decizia locală (pe dispozitiv) sau decizia la server. Arhitectura SmartGarden a fost gândită pentru a permite decizia la server, deoarece astfel logica poate fi mai complexă și mai ușor de actualizat (fără a reflasha firmware-ul dispozitivului). Astfel, la primirea unei noi citiri:
2. Serverul compară SoilMoisture din citire cu pragul definit. Pragul este stocat fie la nivel de utilizator (global pentru toate plantele sale) – de ex. UserSetting.SoilMoistThreshold = 30%, fie particularizat per plantă/specie (dacă s-ar folosi valorile implicite din Species). În MVP s-a mers pe prag global, ajustabil de utilizator în setări.
3. Dacă umiditatea solului este sub prag **și** AutoWateringEnabled este adevărat (udarea automată e activă pentru utilizator), atunci se califică pentru udare.
4. Serverul verifică ultima udare a plantei (din WateringLogs) – pentru a nu uda prea des. Să zicem că se impune o regulă: nu uda automat mai des de o dată la X ore, sau respectă frecvența recomandată pentru specie (ex: o dată la 3 zile). Dacă ultima udare automată a fost recent, serverul poate decide să *nu ude încă*, chiar dacă solul pare uscat, considerând că planta poate fi lăsată puțin să “respire” (această logică imitând recomandarea de a nu uda foarte frecvent chiar dacă la suprafață solul pare uscat).
5. De asemenea, se asigură că nivelul apei din rezervor (waterLevel) este peste un minim. Dacă rezervorul e gol, serverul **nu trimite comanda de udare**.
6. Dacă toate condițiile de mai sus sunt îndeplinite, serverul adaugă în răspunsul către dispozitiv instrucțiunea de udare, specificând durata pompei. În MVP s-a folosit o durată fixă (5 secunde), dar ideal ar putea fi calculată în funcție de deficitul de umiditate (de exemplu, solul are 20% dar pragul e 30%, deci uda 5 sec; dacă solul ar fi 10%, uda 7 sec etc. – relație aproape liniară). Această finețe poate fi ajustată empiric prin teste, deoarece diferite ghivece și soluri reacționează diferit la aceeași cantitate de apă.

Dacă udarea automată nu e necesară sau condițiile nu sunt îndeplinite, serverul răspunde pur și simplu cu un mesaj de confirmare fără comenzi speciale (sau cu waterNow: false).

1. **Preluarea datelor de către aplicația mobilă** – Concomitent sau la scurt timp după ce senzorii trimit date, utilizatorul poate deschide aplicația pentru a vedea situația. Aplicația comunică cu același backend, folosind cereri GET pentru a obține informațiile dorite. De exemplu:
2. Când utilizatorul accesează lista de plante („My Plants”), aplicația face o cerere GET la /api/plant pentru a obține toate plantele utilizatorului curent. Serverul verifică autentificarea (de exemplu, pe baza unui token JWT trimis în header sau a unui ID de utilizator din sesiune – pentru MVP s-a putut implementa și ceva simplu, precum filtrarea după UserId fără un sistem de token complet) și returnează o listă cu plantele. Fiecare plantă conține detaliile sale (nume, specie etc.) și posibil ultimele valori de senzori sau un rezumat. În implementare, fie s-a procedat ca PlantController.GetAll să includă și ultima citire (prin join la SensorReadings ordonate descendent și luarea primei), fie aplicația apelează un endpoint separat pentru fiecare plantă pentru a lua ultimele valori. Pentru eficiență, prima abordare e preferabilă – serverul poate adăuga în DTO-ul de plantă câmpuri suplimentare precum LastSoilMoisture, LastTemperature etc. Astfel, aplicația poate afișa în listă direct, sub fiecare plantă, un mic sumar (de ex. “Soil 46%, Temp 24°C” șamd). În design-ul UI, fiecare card de plantă arată probabil numele, specia, indoor/outdoor și, posibil, un status (de ex. dacă necesită atenție).
3. Când utilizatorul selectează o anumită plantă și intră pe ecranul de **detalii plantă**, aplicația solicită informații detaliate. Poate fi același GET /api/plant/{id} care acum include și listele de SensorReadings recente și WateringLogs pentru acea plantă. Prin includerea acestor date, serverul permite aplicației să afișeze:
   * Valorile curente ale senzorilor (ultimul reading) – de exemplu, procentul apei din rezervor, lumina, temperatura, umiditatea solului, toate actualizate la zi.
   * Starea modului automat (preluat din UserSetting, ex. AutoWateringEnabled) – aplicația arată un switch “Automatic watering” care este ON/OFF în funcție de această valoare. Dacă utilizatorul schimbă poziția switch-ului, aplicația trimite o cerere (PUT/PATCH) la API pentru a actualiza setarea respectivă.
   * Eventual, graficul cu **statistici săptămânale** (“Weekly Stats”) – pentru a-l afișa, aplicația are nevoie de datele istorice pe ultimele 7 zile. Acestea pot fi fie toate citirile din intervalul respectiv (și apoi prelucrate local), fie deja preprocesate de server. O abordare simplă e ca aplicația să primească ultimele N citiri și să extragă valorile relevante (de exemplu valorile zilnice medii sau ultima valoare din fiecare zi). În MVP, graficul ar putea afișa simplu evoluția umidității solului zilnic: luni, marți, ..., duminică – completat cu bare sau linii (posibil folosind un control de grafic in MAUI). De asemenea, evenimentele de udare ar putea fi marcate pe acest grafic (ex: un simbol pe ziua în care a avut loc udarea).
   * Notificări vizuale: de exemplu, mesajul de atenționare *“Please fill the water tank!”* este afișat pe ecranul plantei dacă ultima citire indică un nivel < 10%. Acest text este inclus în datele trimise de server (sau dedus local de aplicație din valoarea numerică).
4. Aplicația afișează datele într-un mod ușor de interpretat: de exemplu, valorile de umiditate, lumină, apă pot fi însoțite de mici icoane și eventual culori (dacă un parametru e în afara intervalului optim, poate fi scos în evidență). Prin actualizarea periodică (polling) – dacă utilizatorul ține ecranul deschis, aplicația poate repeta cererea la API la fiecare X secunde pentru a vedea dacă au apărut citiri noi. Aceasta nu e strict necesară la intervalul de 15 min (utilizatorul probabil nu va sta cu aplicația deschisă permanent), dar oferă senzația de real-time dacă implementată. În plus, se poate folosi *SignalR* sau WebSocket pentru update push, însă nu a fost o cerință în MVP.

**4.2. De la utilizator (aplicație) către dispozitiv (sens „în aval”)**:

1. **Comenzile utilizatorului prin aplicație** – Utilizatorul poate influența sistemul în două moduri principale:
2. **Activarea/dezactivarea udării automate** – prin comutarea opțiunii din profil sau ecranul plantei. Dacă utilizatorul pune pe OFF „Automatic watering” pentru plantele sale, aplicația va transmite către server cererea de modificare (ex. PUT /api/usersettings cu AutoWateringEnabled=false). Serverul salvează noua valoare. Acest lucru se va reflecta ulterior în comportamentul dispozitivului: dacă auto-watering e dezactivat, atunci chiar dacă senzorii detectează sol uscat, sistemul *nu va uda automat* – așteptând intervenție manuală. Practic, dispozitivul primește această setare fie la următoarea comunicare (serverul poate trimite autoWater=false în răspuns, iar dispozitivul updatează un flag intern), fie pur și simplu serverul nu mai trimite comenzi auto. Pentru utilizator, comutarea este instantanee la nivel de interfață (switch-ul se schimbă) și persistă global.
3. **Declanșarea udării manuale** – utilizatorul, observând în aplicație că solul e cam uscat sau dorind pur și simplu să ude planta la cerere, apasă butonul dedicat (de exemplu un buton “Udă acum” pe ecranul plantei). În design-ul dat, nu vedem explicit un buton, dar e posibil să fie acționarea manuală implicată în acel toggle ON/OFF de lângă pompa sau ca funcționalitate la swipe. Să presupunem că există un buton. La apăsare, aplicația trimite o cerere către server, de exemplu POST /api/watering cu corpul conținând plantId și o etichetă că e manual. Serverul, la primirea acestei cereri:
   * Creează un log de udare cu Mode = "MANUAL" și o durată (ex. tot 5 secunde, sau poate permite în aplicație selectarea duratei dacă s-a prevăzut).
   * Transmite către dispozitiv comanda de udare. Aici, modul de transmitere depinde de arhitectură: fie serverul are o conexiune WebSocket cu dispozitivul (puțin probabil în acest context), fie va marca undeva că e nevoie de udare. O soluție simplă implementată: când utilizatorul trimite comanda manuală, serverul salvează log-ul și setează un **flag în baza de date** la acea plantă, de tip “pending\_manual\_watering = true”. Apoi, când dispozitivul mai trimite o citire (sau chiar serverul la următoarea citire programată) va răspunde cu waterNow=true. Practic, comanda manuală s-ar executa la cel mult 15 minute întârziere în acest caz. Pentru a reduce această latență, aplicația, după ce trimite comanda, poate iniția imediat o citire forțată: poate apela un endpoint de genul GET /api/plant/{id}/force-refresh care îi spune dispozitivului (dacă s-a implementat push) să citească senzori acum. O altă variantă: serverul stochează comanda, iar dispozitivul are un loop separat în care interoghează periodic serverul pentru comenzi (ex: la fiecare 5 secunde trimite un ping la /api/command să vadă dacă ceva e în așteptare). Aceasta ar permite aproape instantanee manuale, dar cu costul unor cereri frecvente.

* În MVP, pentru a nu complica, e posibil ca implementarea manuală să se bazeze tot pe ciclul de citire. Adică utilizatorul apasă „Udă acum”, serverul notează, apoi i se sugerează utilizatorului să aștepte următoarea actualizare (sau aplicația afișează o stare “se udă...”). După maxim 15 minute, când dispozitivul comunică, serverul îi răspunde cu comanda de udare manuală (distinsă tot prin Mode=MANUAL). Dispozitivul va porni pompa, iar log-ul era deja înregistrat, deci totul e consemnat. Bineînțeles, ideal ar fi aproape instant – o posibilă îmbunătățire viitoare este folosirea **protocoului MQTT** pentru a trimite comenzi în timp real către dispozitiv.

1. **Alte acțiuni**: utilizatorul poate adăuga o plantă nouă (aplicația trimite POST /api/plant), poate șterge o plantă (DELETE /api/plant/{id}), edita detalii (PUT /api/plant/{id}). De asemenea, calibrările senzorilor sunt acțiuni la inițiativa utilizatorului: când pornește calibrarea unui senzor din aplicație, aceasta trimite comenzi directe către dispozitiv prin rețeaua locală (în modul de configurare inițial, aplicația comunică direct cu ESP32 deoarece acesta e AP-ul la care e telefonul conectat temporar). De exemplu, la calibrarea luminii, când utilizatorul apasă “Calibrate” pe nivelul “Bright Direct Light”, aplicația face o cerere HTTP locală către adresa dispozitivului (cunoscută în modul AP, de ex. 192.168.4.1) de genul /calibrate?sensor=light&level=bright\_direct. Dispozitivul atunci citește senzorul și returnează un răspuns (sau doar confirmă), iar aplicația afișează “Completed” pentru acel nivel. După calibrarea completă, aplicația poate trimite valorile citite către server pentru a le salva ca profil al plantei (de ex. creează un SoilType nou cu parametrii calibrați). Însă, în MVP, aceste valori pot rămâne stocate doar pe dispozitiv.
2. **Executarea comenzilor pe dispozitiv** – Microcontrolerul primește comanda, fie direct (în cazul calibrării în modul AP) fie prin intermediul serverului (în cazul udării manuale sau toggling de mod automat). În cazul calibrării, dispozitivul doar măsoară și stochează parametrii de calibrare (nu implică altceva). În cazul udării manuale, după cum am descris, dispozitivul va porni pompa imediat ce vede comanda. Dacă, de exemplu, utilizatorul este în aplicație când pompa pornește, cum află aplicația că s-a udat? Avem log-ul de udare creat de server (ce poate fi trimis la client dacă este conectat), dar mai important, următoarea citire de senzori va arăta un salt în umiditatea solului. Astfel, utilizatorul vede feedback-ul udării prin actualizarea valorii de umiditate (de exemplu, de la 28% la 45%) și prin prezența unei înregistrări noi în istoricul udărilor.
3. **Fluxul de notificare** – Deși MVP nu a integrat servicii de push notification externe, fluxul de date include notificări *in-app*. De exemplu, dacă rezervorul este gol sau dacă udarea automată a fost dezactivată și solul devine foarte uscat, aplicația poate afișa un banner sau un marcaj roșu la acea plantă pentru a atrage atenția. La nivel de server, se pot defini *evenimente* ce generează notificări: „Rezervor sub 10%” sau „Planta X nu a fost udată de Y zile”. Aceste evenimente pot fi trimise la client prin interogări periodice (app-ul poate cere: ai notificări noi?) sau la autentificare. În viitor, integrarea cu servicii precum Firebase Cloud Messaging ar permite trimiterea notificărilor push direct pe telefon, chiar și cu aplicația închisă.

Pentru a ilustra succint fluxul descris, iată un scenariu concret end-to-end: - La ora 08:00, senzorii plantei Monstera trimit o citire: Temperatură 23.5°C, Umiditate aer 50%, Lumină 60%, Umiditate sol 25%, Apă rezervor 80%. Serverul salvează datele și observă că 25% < prag 30%, deci planta e cam uscată. Auto-watering e ON, ultima udare a fost acum 2 zile, deci se decide udare. Serverul răspunde dispozitivului: „waterNow=true, duration=5 sec”. Dispozitivul pornește imediat pompa pentru 5 secunde. La 08:01, plantei i s-au administrat ~100 ml apă. Dispozitivul trimite apoi un WateringLog (sau serverul deja a creat unul la decizie). - La 08:15, dispozitivul face o nouă citire programată (sau imediat după udare poate face una): Umiditate sol a crescut la 40%. Salvează citirea. Serverul vede că totul e ok, nu mai trimite nicio comandă. - La 09:00, utilizatorul deschide aplicația. Lista de plante arată Monstera cu ultima udare la 08:00 (auto) și umiditate 40%. În ecranul detaliat, vede graficul cu creșterea bruscă a umidității la 08:00 și apoi linie constantă, rezervorul 75% (scăzut de la 80% după udare). Decide să dezactiveze udarea automată (OFF) ca experiment. Switch-ul „Automatic watering” trece pe OFF, aplicația trimite update la server, serverul salvează. - Trece o zi, utilizatorul uită să reactiveze auto. La 08:00 a doua zi, citire: umiditate sol scăzută la 22%. Auto-watering e OFF, deci serverul nu comandă udare. Doar alertează: creează o notificare de sol uscat. - Utilizatorul la 12:00 vede în aplicație că Monstera are sol 18% și un simbol de avertizare. Apasă „Udă acum”. Serverul loghează cererea manuală. În secunda următoare, dispozitivul (care întâmplător tocmai a trimis sau trimite date la fiecare oră pentru economie) primește comanda și udă 5 secunde. Umiditatea urcă la 35%. Problema rezolvată, auto-watering poate fi reactivat de utilizator ulterior.

Acest flux evidențiază colaborarea strânsă între componente: **dispozitivul** culege și acționează, **serverul** decide și înregistrează, **aplicația** afișează și permite intervenții. Toate piesele comunică prin intermediul API-ului, asigurând consistența datelor (datele senzorilor sunt sursa “adevărului” pentru decizii și UI) și securitatea (fiecare utilizator vede doar datele sale). În implementare, s-a acordat atenție sincronizării – de exemplu, la adăugarea unei plante noi în aplicație, este necesar să se “asocieze” cumva dispozitivul fizic cu acea plantă (poate printr-un ID de dispozitiv). O metodă simplă e ca la configurarea inițială, aplicația să preia un **DeviceID** de la modulul ESP și când se creează planta în baza de date, să salveze acel DeviceID la plantă. Astfel, când acel dispozitiv trimite date ulterior, serverul știe pentru ce plantă sunt (făcând mapare DeviceID -> PlantId). Alternativ, după ce utilizatorul adaugă planta în aplicație, primește un PlantId, pe care apoi aplicația îl transmite dispozitivului (în modul AP config) pentru a-l salva. În proiectul SmartGarden, s-a optat pentru a transmite PlantId către dispozitiv în faza finală a configurării inițiale (pasul “Add your plant details”). Astfel, dispozitivul în fiecare comunicare specifică plantId-ul corect.

În concluzie, fluxul de date al SmartGarden este conceput să fie **bidirecțional și reactiv**: datele „urcă” de la senzori la utilizator, furnizând imaginea fidelă a mediului plantei, iar comenzile și setările „coboară” de la utilizator (sau algoritm) către dispozitiv, influențând modul în care sistemul acționează asupra plantei (udând-o la momentul potrivit). Această buclă închisă asigură un sistem autonom care totodată ține utilizatorul în control și informat.

## 5. Algoritmica de udare

Algoritmul de udare implementat în SmartGarden combină reguli simple bazate pe praguri de umiditate, condiții temporale (frecvență maximă a udărilor) și modurile de operare (automat vs. manual). Scopul algoritmului este de a determina **când trebuie udată planta** și **câtă apă să fie administrată**, ținând cont de datele furnizate de senzori și de preferințele utilizatorului.

Principalele elemente care influențează decizia de udare sunt: - **Umiditatea solului curentă vs. pragul minim acceptat** – acesta este criteriul de bază. Dacă umiditatea măsurată (%) scade sub pragul setat (de ex. pragul default 30%), atunci planta are nevoie de apă. Pragul poate fi ajustat: un utilizator ar putea crește pragul la 35% dacă observă că planta dă semne de ofilire deja la 30%, sau îl poate coborî dacă dorește să forțeze planta (de ex. cactușii pot sta uscați mai mult). - **Starea sistemului de udare automată** – dacă modul automat este activat sau nu. Chiar dacă solul e uscat sub prag, **nu se va uda automat** dacă utilizatorul a dezactivat udarea automată (preferând să ude manual). În acest caz, sistemul doar va semnala condiția, dar nu va acționa pompa autonom. - **Intervalul de timp de la ultima udare** – pentru a evita udările prea frecvente, algoritmul impune un interval minim între două udări succesive. Implicit, după o udare (manuală sau automată), sistemul așteaptă cel puțin X ore înainte de a permite o nouă udare automată. X poate fi configurat în funcție de specie (de exemplu, dacă Species.DefaultWaterFreqDays = 3 zile, atunci nu uda automat mai des de o dată la 3 zile). În MVP, o regulă simplă aplicată a fost: *o singură udare automată pe zi* maxim per plantă, indiferent de prag. Astfel, chiar dacă seara solul scade din nou sub prag, se amână udarea până a doua zi – idee care reflectă practica de a nu uda plantele de două ori într-un interval scurt, lăsând timp apei să fie absorbită complet și evitând putrezirea rădăcinilor. - **Disponibilitatea apei în rezervor** – dacă nu e apă în rezervor, udarea nu poate avea loc. Algoritmul verifică senzorul de nivel: sub un anumit nivel (de ex. <5%), orice udare este blocată. În plus, dacă nivelul este scăzut (ex. 10-15%), sistemul preferă să nu facă udare automată decât dacă este absolut necesar, pentru a nu consuma ultimele rezerve fără ca utilizatorul să fie conștient (aceasta ar putea lăsa planta neudată ulterior). În practică, sistemul anunță utilizatorul și așteaptă reumplerea rezervorului. - **Modul de declanșare a udării** – manual sau automat. Dacă udarea e declanșată manual de utilizator, algoritmul este simplu: comanda utilizatorului are prioritate imediată (presupunând că nu contravine unor protecții precum rezervor gol). Durata udării manuale poate fi prestabilită sau aleasă de utilizator (în MVP, prestabilită la ~5 sec). După udare, sistemul va înregistra evenimentul și își va relua ciclul normal. - **Alți senzori și factori** – deși versiunea actuală nu ia decizii pe baza lor, se pot considera: dacă luminozitatea e scăzută (planta stă la întuneric), evapotranspirația e mai mică, deci planta consumă apă mai încet – se poate modera frecvența udărilor. Sau dacă umiditatea aerului e foarte mare, solul se usucă mai încet. Aceste finețuri pot fi incorporate într-o versiune viitoare a algoritmului (posibil folosind un model AI care să prelucreze toți acești parametri).

**Descrierea algoritmului pas cu pas (cazul udare automată):**

Pseudocod generalizat (executat la primirea unei noi citiri de la senzori sau la un tick de verificare):

if (userSetting.AutoWateringEnabled) {  
 double soil = sensorReading.SoilMoisture; // umiditatea sol % curentă  
 double threshold = userSetting.SoilMoistThreshold; // pragul setat (ex. 30%)  
 DateTime lastWater = plant.LastWateringTime; // timpul ultimei udări (din loguri)  
 bool tankLow = sensorReading.WaterLevel < 5; // rezervor ~gol  
  
 if (!tankLow && soil < threshold) {  
 TimeSpan sinceLast = now - lastWater;  
 if (sinceLast >= MIN\_INTERVAL) {  
 // Condiții îndeplinite: se inițiază udarea  
 waterDuration = CalculateDuration(soil, threshold);   
 StartWaterPump(waterDuration);  
 LogWatering(plantId, waterDuration, mode="AUTO");  
 }  
 else {  
 // Sol uscat dar udat recent - se amână udarea (plantă "în așteptare")  
 ScheduleNextCheck(plantId, time: lastWater + MIN\_INTERVAL);  
 }  
 }  
 // optional: else if (soil >= threshold && recentlyWatered) -> no action, just monitor  
} else {  
 // AutoWatering off  
 if (soil < threshold && !notifiedUser) {  
 NotifyUser("Planta X are solul uscat, udati manual sau activati auto-watering.");  
 notifiedUser = true;  
 }  
}

Fragmentul de cod de mai sus explică în termeni simpli logica: dacă modul auto e pornit și umiditatea e sub prag, iar rezervorul are apă și a trecut destul timp de la ultima udare, atunci declanșează udarea. Altfel, ori nu face nimic, ori doar notifică.

**Calculul duratei udării (waterDuration):** În cea mai simplă variantă, este o constantă (ex. 5 sec). O variantă îmbunătățită liniară:

double deficit = threshold - soil; // ex: threshold 30, soil 20 -> deficit 10%  
double maxDeficit = threshold; // ex: 30% dacă era complet uscat (0% vs 30%)  
waterDuration = MIN\_DURATION + (MAX\_DURATION - MIN\_DURATION) \* (deficit / maxDeficit);  
// MIN\_DURATION poate fi 3 sec (dacă e aproape de prag, un strop rapid), MAX\_DURATION 6 sec (dacă e foarte uscat)

Astfel, dacă solul e la 0% (teoretic complet uscat, deficit=30), s-ar da MAX\_DURATION (ex. 6 secunde). Dacă e la 25% (deficit=5 din 30, deci ~16%), se dă ~1 sec doar, ceea ce poate fi insuficient totuși – deci formula s-ar ajusta eventual neliniar (ex. radical sau logaritmic) sau punând un prag minim. În practică, de regulă e mai sigur să ai o cantitate fixă moderată decât calcule complicate; calibrările și observațiile practice pot apoi modifica parametrii.

**Comportament după udare:** După ce o udare automată are loc, sistemul: - Marchează timpul udării (în baza de date). - **Evită citirile** imediat după udare care ar putea fi eronate: de exemplu, senzorul de umiditate a solului citit la câteva secunde după udare ar putea arăta umiditate foarte mare local (dacă apa nu s-a dispersat uniform) sau, invers, dacă e lent, poate nu a înregistrat încă creșterea. O abordare e ca după udare, sistemul să aștepte câteva minute înainte de următoarea citire a solului pentru acuratețe. În MVP, având citiri la 15 min, acest aspect e implicit acoperit (sunt 15 min oricum). - **Împiedică udări repetate**: cum am zis, menține un interval minim. Dacă planta are încă sub prag după interval, se va repeta.

**Udarea manuală:** Când modul manual e folosit, algoritmul e simplificat: - Comanda vine direct de la utilizator, deci se execută oricând, chiar dacă recent a mai fost udare (presupunem că utilizatorul știe ce face, deși ar fi util să-i semnalăm: “tocmai a fost udată acum 1 oră, sigur mai vrei?” – dar asta ține de UX). - Sistemul totuși verifică rezervorul (dacă nu e apă, va refuza executarea). - Pornește pompa pe durata prestabilită. - Loghează evenimentul (mode = MANUAL). - Resetează eventual contorul de timp de la ultima udare – practic, ultima udare devine cea manuală acum. - Nu modifică nimic la auto/threshold.

**Edge cases și protecții:** - Dacă senzorul de umiditate a solului dă citiri anormale (ex. 0% mereu din cauza defectării sau 100% dacă s-a scos din pământ), algoritmul poate detecta valori imposibile și ignora/comunica eroarea (ex. un soil moisture < 5% ar putea indica senzor scos). - Dacă două plante diferite sunt deservite de același dispozitiv (teoretic posbil dacă un ESP32 are 2 seturi de senzori), algoritmul trebuie să decidă independent pe fiecare. În implementarea actuală, s-a considerat **un dispozitiv per plantă** pentru simplitate. - Dacă modul auto e activat la jumătatea unei perioade în care solul e deja uscat sub prag, sistemul va reacționa imediat: ex. auto era OFF, solul se duce la 15%, apoi utilizatorul pornește auto – sistemul va vedea sol=15<30 și va uda imediat (presupunând că ultima udare era demult, deci e ok). - Dacă modul auto e dezactivat în timp ce pompa e în funcțiune (situație rară, dar teoretic posbil dacă user vede ca a pornit auto și oprește), sistemul oricum finalizează udarea începută, dar nu va mai repeta.

**Integrarea altor factori:** În extinderi ulterioare, algoritmul poate fi augmentat: - **Temperatura aerului**: în zile caniculare, plantele au nevoie de mai multă apă. Algoritmul poate scădea pragul efectiv de umiditate (uda mai devreme) când temperatura depășește un prag. Exemplu: dacă temp > 30°C, crește pragul perceput la 35%. Invers, la temp joasă (plante la 15°C, cresc mai lent), poate lăsa pragul 25%. - **Lumina**: dacă sistemul vede că planta stă la întuneric toată ziua (poate e într-o cameră slab luminată), ar putea decide udări mai rare (planta evaporă mai puțin). - **Umiditatea aerului**: deși nu direct legat de apă din sol, un aer foarte uscat poate stresa frunzele; algoritmul ar putea genera udări ușor mai dese pentru a compensa pierderile (sau să recomande utilizatorului creșterea umidității ambientale). - **Predicții meteo** (pentru plante outdoor): dacă știe că mâine va ploua, poate amâna udarea azi chiar dacă solul e uscat, economisind apă de rezervor. Acest tip de integrare transformă algoritmul static într-unul mai aproape de **AI/ML**, necesitând date externe.

În MVP însă, s-a implementat strategia de bază pragmatică: *prag de umiditate + interval minim*, suficientă pentru a menține planta într-un interval acceptabil de umiditate. Această abordare a fost testată în practică (vezi secțiunea de Testare) și s-a dovedit eficientă – de exemplu, pentru planta Monstera testată, setarea pragului la ~30% a făcut ca udările automate să aibă loc cam o dată la 2-3 zile, menținând umiditatea între 30% și ~50%, ceea ce corespunde cerințelor plantei. Dacă pragul ar fi fost mult mai jos, planta ar fi suferit secetă; dacă ar fi fost mult mai sus, sistemul ar fi udat zilnic inutil. Astfel, ajustarea pragului e un instrument cheie pus la dispoziția utilizatorului, dar are un bun default determinat experimental.

**Cod exemplificativ (C#)** din serviciul de udare (WateringService) al backend-ului, integrând logica de mai sus, ar putea arăta astfel:

public async Task<bool> EvaluateAndWaterAsync(int plantId, SensorReading latestReading) {  
 var plant = await \_plantRepository.GetByIdAsync(plantId);  
 var settings = await \_userSettingRepository.GetByUserIdAsync(plant.UserId);  
 if (plant == null || settings == null) return false;  
 if (!settings.AutoWateringEnabled) {  
 return false; // auto off, nu face nimic (poate logheaza intentia)  
 }  
 double soil = latestReading.SoilMoisture;  
 double threshold = settings.SoilMoistThreshold;  
 if (soil < threshold) {  
 // verifica ultima udare  
 var lastLog = await \_wateringLogRepository.GetLastByPlantAsync(plantId);  
 if (lastLog == null || (DateTime.UtcNow - lastLog.CreatedAt).TotalHours >= 12) {  
 // rezervor ok?  
 if (latestReading.WaterLevel.HasValue && latestReading.WaterLevel.Value < 5) {  
 // trimite notificare rezervor gol  
 await \_notifier.NotifyAsync(plant.UserId, "Rezervorul este gol. Udarea automata nu s-a putut realiza.");  
 return false;  
 }  
 // putem uda  
 int durationSec = 5;  
 // apel la microcontroler prin raspuns la sensor reading - pseudo, in realitate se pregateste in DTO de raspuns  
 await \_wateringLogRepository.CreateAsync(new WateringLog {  
 PlantId = plantId,  
 DurationSec = durationSec,  
 Mode = "AUTO",  
 CreatedAt = DateTime.UtcNow  
 });  
 return true;  
 }  
 }  
 return false;  
}

Acest cod (deși simplificat și conceptual) ar face parte din fluxul de procesare a citirilor. La integrarea reală, dacă EvaluateAndWaterAsync întoarce true și durationSec, controlerul API ar popula obiectul de răspuns către device cu waterNow=true.

**Concluzie:** Algoritmica de udare a SmartGarden folosește **principii deterministe** (praguri, timere) pentru a asigura un comportament predictibil și ușor de înțeles pentru utilizator. Prin parametri configurați (prag umiditate, frecvență) se poate personaliza comportamentul pentru diferite plante și preferințe. În același timp, algoritmul e proiectat să evite situațiile extreme: nu udă prea des (prevenind supraumidificarea) și nu lasă planta să se usuce excesiv. Această abordare **rule-based** constituie baza, peste care se pot suprapune pe viitor mecanisme mai inteligente (de exemplu, un model care învață din istoricul fiecărei plante cât de repede consumă apă, ajustând astfel dinamic parametrii de udare). Pentru versiunea de față, testele au arătat că un prag bine ales și o durată fixă oferă rezultate foarte bune, menținând planta într-o stare sănătoasă cu un minim de apă irosită.

## 6. Backend API

**SmartGarden.API** reprezintă serviciul web backend al sistemului, responsabil de expunerea funcționalităților către aplicația mobilă și dispozitivul IoT, de orchestrarea logicii de business și de persistența datelor prin intermediul bazei de date. Backend-ul este construit cu **ASP.NET Core Web API (framework .NET 8)**, folosind arhitectura RESTful pentru a oferi operații CRUD și acțiuni specifice asupra resurselor din sistem (plante, senzori, udări, utilizatori etc.). De asemenea, integrează **Entity Framework Core** ca ORM pentru a comunica cu baza de date **Microsoft SQL Server** în mod obiectual. Modulul SmartGarden.API face legătura între solicitările HTTP și operațiile la nivel de bază de date, aplicând totodată regulile de business definite.

Principalele caracteristici ale backend-ului includ:

* **Stratificare și organizare pe controlere**: Fiecare entitate principală are un controler dedicat, ceea ce asigură un cod clar și ușor de întreținut. De exemplu:
* PlantController gestionează tot ce ține de plante (listare plante, detalii plantă, adăugare, ștergere, eventual actualizare).
* SensorController acoperă operațiile legate de citirile de senzori (de obicei primirea de noi date de la dispozitiv).
* WateringController se ocupă de log-urile de udare și de comenzi de udare manuală.
* (Opțional ar exista și un UserController pentru autentificare și profil, însă în implementarea actuală s-a mers pe un model simplificat, fără un sistem OAuth/JWT complet – posibil autentificarea e gestionată minimal, ceea ce vom detalia mai jos.)

Fiecare controller este marcat cu atributele [ApiController] și are un route de genul api/[controller], însemnând că, de exemplu, PlantController răspunde la URL-urile care încep cu /api/plant. Metodele publice din controler sunt mapate la acțiuni HTTP (GET, POST, PUT, DELETE) prin atribute precum [HttpGet], [HttpPost] etc. și definesc parametrii (din route sau din corpul cererii).

* **Endpoint-urile și funcționalitățile principale**: Vom trece în revistă principalele endpoint-uri oferite de API și funcțiile lor:
* **Autentificare și utilizatori**: Pentru MVP, s-a implementat înregistrarea și login-ul într-un mod simplu. Utilizatorii își pot crea cont (endpoint POST /api/user/register probabil) trimițând un username, email și parolă. Parola este hashu-ită și salvată (nu în clar) în baza de date (ex. folosind un algoritm de hashing precum BCrypt sau standard .NET Identity). Login-ul (POST /api/user/login) verifică credențialele și, la succes, poate returna un token JWT de autentificare. Acest token ar trebui trimis de aplicație în header-ul de autorizare la fiecare cerere ulterioară. În implementarea de față, este posibil că s-a folosit direct *Identity API* oferită de ASP.NET pentru a crea utilizatori și a genera cookies/JWT. Deși în listarea de fișiere nu apare explicit un controller de User, e posibil să se fi folosit *Identity scaffolding* (fișierele \_Layout.cshtml și altele din Views indică prezența unor elemente de UI pentru login). Pentru simplitate, vom descrie conceptul: după login, toate cererile sunt autorizate pe baza ID-ului de utilizator asociat token-ului. Fiecare controler cu date sensibile filtrează datele după UserId (de exemplu, PlantController la GET returnează doar plantele care aparțin user-ului autentificat).
* **Plante (PlantController)**:
  + GET /api/plant – returnează lista tuturor plantelor aparținând utilizatorului autentificat. Acesta întoarce o listă de obiecte de tip Plant (sau PlantDto) care conțin detalii ca *PlantId, Species (poate ca text sau id), SoilType, RoomName, IsOutdoor, DateAcquired* etc., eventual și ultimele valori ale senzorilor sau un rezumat al statusului. Exemplu de răspuns (JSON):
  + [  
     {  
     "plantId": 1,  
     "nickname": "Ficus Lyrata",  
     "speciesName": "Ficus lyrata",  
     "isOutdoor": false,  
     "roomName": "Living Room",  
     "lastSoilMoisture": 46,  
     "lastTemperature": 24,  
     "lastLight": 78,  
     "waterTank": 10  
     },  
     ...  
    ]
  + Acesta corespunde cu ceea ce arată aplicația la cardurile din lista de plante. Observăm că aplicația afișează “INDOOR”/“OUTDOOR” și camera (Bedroom, Living etc.) ca filtre și în descrierea fiecărei plante.
  + GET /api/plant/{id} – returnează detaliile pentru planta cu identificatorul dat (cu condiția să aparțină utilizatorului curent). Pe lângă datele statice ale plantei, acest endpoint poate include: ultimele n citiri de senzori (pentru grafic), ultimele log-uri de udare, statusul modulului automat. În codul sursă, vedem că metoda GetById a PlantController cheamă serviciul corespunzător \_plantService.GetByIdAsync(id) și întoarce obiectul Plant dacă este găsit. Dacă nu există sau nu aparține user-ului (verificare internă), se întoarce 404 Not Found.
  + POST /api/plant – adaugă o nouă plantă în sistem. Corpul cererii este un DTO (CreatePlantDto) ce conține informațiile necesare: specie (id sau nume), tip sol (id sau select implicit), nume sau poreclă planta (dacă se folosește), camera, indoor/outdoor, data achiziției. Pe baza acestor date, serverul creează o nouă intrare Plant în baza de date. Implementarea din cod arată că la crearea cu \_plantService.CreateAsync(dto) se returnează obiectul creat și se folosește CreatedAtAction pentru a indica locația noului resource. De reținut, trebuie legat și de un utilizator – cel curent, luând userId din contextul de autentificare. După crearea plantei, serverul ar putea inițializa și un UserSetting default dacă cumva nu exista (dar de regulă se crează la creare cont).
  + DELETE /api/plant/{id} – șterge planta specificată (dacă aparține user-ului curent). Aceasta implică ștergerea citirilor asociate și a log-urilor de udare (fie cascade prin EF, fie manual înainte). Endpoint-ul întoarce 204 NoContent la succes. Aplicația confirmă ștergerea cu un dialog: *“Do you want to delete the plant? Yes / No”* înainte de a trimite cererea.
  + (Opțional) PUT /api/plant/{id} – pentru modificări (de ex. schimbare cameră sau poreclă). Dacă implementat, ar primi un DTO cu field-urile editabile și ar salva.
* **Citiri senzori (SensorController)**:
  + POST /api/sensor – primește de la dispozitiv IoT o nouă citire. Dispozitivul trimite datele în corpul cererii (în format JSON, mapat automat într-un CreateSensorReadingDto). Această acțiune este *autentificată diferit* – de fapt, dispozitivul nu trece prin autentificarea utilizator clasică. Cel mai probabil, securitatea acestei rute se bazează pe un token de dispozitiv sau pe faptul că ID-ul plantei din corp este suficient (dar ideal nu e, deoarece oricine ar putea trimite date). O variantă a fost ca device-ul să folosească un token special primit la configurare. În MVP, e posibil să se fi permis accesul la acest endpoint fără autentificare (dar cu un model de validare proprie – ex. un secret API key inclus în codul dispozitivului). Aceasta este o zonă ce poate fi întărită în viitor. La primirea cererii, controlerul apelează \_sensorService.CreateAsync(dto), care creează în baza de date înregistrarea. După salvare, serviciul de senzori poate apela logica de udare automată (IWateringService) pentru a evalua dacă trebuie udat acum (cum am discutat la secțiunea algoritm). Dacă s-a decis udare, aceasta poate fi comunicată în răspuns. Tehnic, controlerul poate compune răspunsul sub forma unui obiect conținând eventual instructțiuni. În cod nu am reușit să vedem exact implementarea, dar conceptual:
  + [HttpPost]  
    public async Task<ActionResult<SensorReading>> Create(CreateSensorReadingDto dto) {  
     var reading = await \_sensorService.CreateAsync(dto);  
     if (reading == null) return BadRequest();  
     // Decide watering  
     var waterInstruction = await \_wateringService.CheckAutoWaterAsync(reading);  
     return Ok(new {   
     sensorReadingId = reading.Id,   
     waterNow = waterInstruction.ShouldWater,   
     duration = waterInstruction.DurationSec   
     });  
    }
  + Dispozitivul interpretează acel JSON și acționează.
  + GET /api/sensor/latest?plantId=X – posibil un endpoint convenabil pentru aplicație, care întoarce ultima citire (sau ultimele N) pentru planta specificată. Dacă aplicația nu vrea toată lista completă (care ar putea fi mii de intrări în timp), poate apela acest endpoint pentru graficul de 7 zile (serverul poate filtra ultimele 7 zile direct în SQL).
  + (Altele) – se pot defini endpointuri pentru grafic: ex. GET /api/sensor/weekly/{plantId} care returnează 7 valori (câte una pe zi), preprocesate de server.
* **Udări (WateringController)**:
  + POST /api/watering – folosit atât de dispozitiv cât și de aplicație. Dacă Mode este “AUTO”, poate indica că e un log venit de la device (sau decizie server deja luată). Dacă Mode e “MANUAL” și vine de la aplicație, semnifică cerere de udare manuală. În interior, controlerul apelează \_wateringService.TriggerWatering(dto) sau similar:
  + Pentru MANUAL, serviciul va crea log-ul de udare, apoi va nota că planta trebuie udată. Aici designul e tricky: cum spuneam, fie se alege să *trimită direct la device* (dacă device-ul menține un canal deschis sau se poate notifica printr-un semnal), fie salvează cererea și se bazează pe citirea următoare. O implementare ar fi ca WateringService să seteze un câmp Plant.NeedsManualWater = true. Apoi SensorService când procesează următoarea citire să vadă acel flag și să șteargă flag-ul și să răspundă cu waterNow.
  + Pentru AUTO, dacă vine de la device (să zicem device-ul trimite confirmare că a udat automat?), se va crea log-ul și atâta tot.
  + GET /api/watering/logs?plantId=X – întoarce lista de WateringLogs pentru o plantă, posibil filtrate la ultimele Y (aplicația poate afișa un istoric sau folosi datele la grafic, de ex. punând un simbol pe zilele când s-a udat).
  + GET /api/watering/next?plantId=X – potențial se putea oferi informații despre *când urmează următoarea udare automată estimativ*, dar asta e mai degrabă la nivel de aplicație (predicții, MVP nu are, dar e planificat ca extindere).
* **Servicii interne și design-ul de cod**: Backend-ul utilizează **Dependency Injection** oferit de .NET. Interfețele din SmartGarden.Core (IPlantService, ISensorService, IWateringService etc.) sunt implementate în SmartGarden.API/Services (PlantService, SensorService, WateringService). Aceste servicii sunt înregistrate în containerul DI la startup (în Program.cs) astfel încât controlerele să le primească prin constructor injection. De exemplu, PlantController are un constructor ce cere un IPlantService, pe care framework-ul îl furnizează. Această abordare permite, de exemplu, testarea ușoară a controlerelor prin injectarea unor implementări false (mock) ale serviciilor.

Implementările serviciilor folosesc la rândul lor repository-urile sau direct contextul EF pentru a realiza operațiile: - PlantService encapsulează logică de genul: “când ștergi o plantă, șterge și toate citirile și log-urile asociate” (posibil folosind Include și RemoveRange). - SensorService conține metoda de creare a citirii: ia DTO-ul, creează entitatea SensorReading, salvează prin context. Ar putea de asemenea să apeleze direct WateringService pentru evaluare imediată (depinde cum s-a compartimentat logica – fie SensorController cheamă WateringService, fie SensorService intern). - WateringService implementează logica de decizie auto (discutată anterior) și acțiunea de logare a unei udări manuale. De asemenea, dacă ar exista integrare cu hardware direct (ex. MQTT), tot aici s-ar trimite comenzi.

* **Baza de date și ORM**: Entity Framework Core este configurat în Program.cs (se adaugă DbContext cu string de conexiune la SQL). Modelul de date este definit de clasele din SmartGarden.Core.Models (Plant, SensorReading etc.), iar în SmartGarden.Data/Configurations sunt configurațiile fine (ex. denumiri de coloane, chei primare compuse, relații 1-1 pentru User-UserSetting etc.). De exemplu, UserSettingConfiguration va specifica că UserSetting are cheie primară tot UserId și foreign key către User, cu Delete cascade probabil. WateringLogConfiguration ar specifica că Mode e un string de max 10 chars etc. EF creează pe baza acestor configurații schema: tabelele Users, UserSettings, Plants, Species, SoilTypes, SensorReadings, WateringLogs cu coloanele deduse:
* User: UserId (PK, int identitate), Email (varchar), PasswordHash, Name, CreatedAt.
* UserSetting: UserId (PK și FK către Users), AutoWateringEnabled (bit), SoilMoistThreshold (double), DataReadIntervalMin (int).
* Species: SpeciesId (PK), CommonName, ScientificName, DefaultWaterFreqDays, DefaultSoilMoistMin, DefaultSoilMoistMax, etc. (câmpuri pentru limite recomandate).
* SoilType: SoilTypeId, Name, Description, RecWaterDueSec, PauseBetweenWaterMin – presupunem câteva posibile atribute, nu se văd clar fără cod, dar RecWaterDueSec ar putea însemna “timp recomandat până la udare” în secunde (sau sec).
* Plant: PlantId (PK), UserId (FK), SpeciesId (FK), SoilTypeId (FK), RoomName (varchar), IsOutdoor (bit), DateAcquired (datetime).
* SensorReading: Id (PK), PlantId (FK), Timestamp (datetime), Temperature (float), Humidity (float), SoilMoisture (float), LightLevel (float, poate null), AirQuality (float null).
* WateringLog: WateringId (PK), PlantId (FK), DurationSec (int), Mode (varchar, ex "AUTO"/"MANUAL"), CreatedAt (datetime).

Relațiile sunt cele discutate (User-UserSetting 1-1, User-Plant 1-n, Plant-SensorReading 1-n, Plant-WateringLog 1-n, Species-Plant 1-n, SoilType-Plant 1-n). EF a generat migrările pe baza acestor modele. Prima migrare (InitialCreate) a creat tabelele principale, a doua migrare (AddPlantRelations) posibil a adăugat coloane FK (din denumire, poate inițial uitaseră să pună relația Plant-User, și au adăugat-o în a doua migrare). Migrations ensure the database is up to date with the model.

* **Modularitate și reutilizare**: Un avantaj al structurilor Core/Data/API separate este că se pot eventual expune și altfel de interfețe pe același Core. De exemplu, s-ar putea crea un SmartGarden.Web (o interfață web) folosind același Core și Data. Sau se pot scrie teste unitare pentru Core (fără dependență de EF). În proiectul actual, acest design a ajutat la organizare, chiar dacă pentru licență nu era necesar un alt client.
* **Swagger și documentație**: Proiectul a integrat **Swagger UI** (prin pachetele Swashbuckle.AspNetCore visibile în output-ul bin). La rularea API-ului, la ruta /swagger se poate vedea interactiv toate endpoint-urile, cu posibilitatea de a le testa. Pentru un proiect de licență, asta e util atât pentru dezvoltare cât și ca demonstrație a API-ului.
* **Securitate**: Actualmente, securitatea principală este pe autentificarea utilizatorilor. Dintre aspectele de îmbunătățit: autentificarea dispozitivului (să nu permită oricui să posteze date la /api/sensor), rate limiting la endpointuri (să nu se spameze serverul), validări stricte pe DTO-uri (de ex. CreatePlantDto ar trebui să valideze că speciesId există, etc.). Probabil s-au implementat validări de bază prin [Required] pe DTO-uri și folosirea model state (ApiController face automat BadRequest dacă model state invalid). Orice parametru din route (id) se validează că entitatea există altfel returnează 404 cum am văzut.
* **Logging și monitorizare**: .NET API oferă logging, nu știm dacă au integrat ceva (serilog sau măcar console logging). Fiind proiect academic, accentul a fost pe funcționalitate; oricum, EF loghează interogările deci se poate vedea ce face.

Un mic exemplu de interacțiune prin API: - Utilizatorul vrea să adauge o plantă. Aplicația face POST /api/plant cu JSON:

{  
 "speciesId": 5,  
 "nickname": "Monstera",  
 "roomName": "Bedroom",  
 "isOutdoor": false,  
 "dateAcquired": "2025-05-10"  
}

Răspunsul serverului (201 Created) conține:

{  
 "plantId": 12,  
 "species": { "speciesId": 5, "commonName": "Monstera Deliciosa", ... },  
 "soilType": { "soilTypeId": 1, "name": "Universal potting soil" },  
 "roomName": "Bedroom",  
 "isOutdoor": false,  
 "dateAcquired": "2025-05-10T00:00:00",  
 "userId": 7  
}

Aplicația ia plantId=12. La finalul configurării, transmite acest ID către microcontroler (de exemplu printr-un request local în modul AP). Dispozitivul salvează ID-ul și de acum înainte îl include în mesajele sale. - Câteva ore mai târziu, dispozitivul trimite POST /api/sensor cu:

{ "plantId": 12, "temperature": 23.7, "humidity": 60.1, "soilMoisture": 28.4, "lightLevel": 12.0, "airQuality": null }

Serverul procesează și găsește că auto e ON, 28.4 < 30 prag => ude. Trimite răspuns:

{ "sensorReadingId": 556, "waterNow": true, "durationSec": 5 }

Dispozitivul pornește pompa 5 sec. Imediat face POST /api/watering cu:

{ "plantId": 12, "durationSec": 5, "mode": "AUTO" }

Serverul salvează log-ul. (Alternativ serverul ar fi salvat direct la decizie). - Când userul deschide aplicația, face GET /api/plant/12. Răspuns:

{  
 "plantId": 12, "nickname": "Monstera", "speciesName": "Monstera Deliciosa",  
 "roomName": "Bedroom", "isOutdoor": false,  
 "lastReading": {  
 "timestamp": "2025-08-13T08:00:00", "temperature": 23.7, "humidity": 60.1,  
 "soilMoisture": 28.4, "lightLevel": 12.0, "airQuality": null  
 },  
 "lastWatering": { "createdAt": "2025-08-13T08:00:30", "mode": "AUTO" },  
 "autoWateringEnabled": true,  
 "threshold": 30.0,  
 "weeklyReadings": [ ... ],  
 "wateringLogs": [ ... ]  
}

Aplicația afișează datele pe ecranul detalii plantei într-un format prietenos.

Acest exemplu arată cum cooperarea între front-end, back-end și device este orchestrată de API. **Backend-ul acționează ca un hub central**, menținând consistența datelor și aplicând politici (ex. autentificare, permisiuni, reguli de udare).

Din perspectiva dezvoltării, s-au folosit facilități moderne ale ASP.NET Core: - Middlewares implicite pentru rutare, model binding (care mapează automat JSON la DTO-uri), - Filtre eventual pentru autentificare (ex. [Authorize] pe controlere ce necesită login – poate nu pe Sensor care e public pentru device), - CORS configurat pentru a permite aplicației mobile să acceseze (nu era neapărat cazul, mobile vs. web).

În rezumat, **Backend API-ul SmartGarden** oferă toate operațiile necesare funcționării sistemului, într-o manieră **structurată (RESTful)** și **sigură** (prin izolarea datelor pe utilizatori). Tehnologiile ASP.NET Core și Entity Framework asigură performanță și ușurință în dezvoltare: de exemplu, generarea API-ului s-a facut rapid folosind pattern-urile de tip CRUD, iar integritatea datelor este gestionată de EF (care se ocupă să șteargă în cascadă sau să impună existența referințelor). În plus, folosirea .NET atât pe server cât și pe client (MAUI) a permis, în teorie, reutilizarea unor părți de cod (de ex., s-ar fi putut împărți SmartGarden.Core ca o bibliotecă comună accesibilă și de aplicația mobilă, pentru a avea același definitii de DTO și modele – în practică nu e clar dacă s-a făcut asta sau s-au duplicat modele în app, dar era posibil).

Backend-ul este gazduit local pe timpul dezvoltării (posibil la http://localhost:5000) și poate fi ușor publicat pe un server cloud (Azure App Service, de ex.), împreună cu o baza de date Azure SQL, pentru acces remote de către dispozitiv și telefon. Pentru testare, s-a putut folosi și un tunel (ex. ngrok) ca dispozitivul să poată accesa API-ul de pe rețeaua locală a dezvoltatorului.

În concluzie, SmartGarden.API reprezintă coloana vertebrală a sistemului, asigurând **comunicarea uniformă** între componente, **persistența fiabilă** a informațiilor și **aplicarea regulilor** ce guvernează comportamentul întregului sistem. Fără acest backend, dispozitivul hardware și aplicația mobilă nu ar putea "gândi" și colabora – backend-ul este cel care face SmartGarden cu adevărat *smart*, centralizând datele și deciziile.

## 7. Aplicația mobilă .NET MAUI

Aplicația mobilă SmartGarden, dezvoltată în cadrul proiectului, este interfața prin care utilizatorul interacționează cu sistemul – configurând dispozitivul, vizualizând informațiile și controlând funcțiile disponibile. Tehnologia folosită este **.NET MAUI (Multi-platform App UI)**, care permite crearea de aplicații native cross-platform din același cod C# și XAML. Astfel, SmartGarden.App rulează pe Android și iOS (și teoretic pe Windows sau Mac dacă e necesar), oferind o experiență unitară pe diferite dispozitive.

### 7.1 Arhitectura aplicației și structura proiectului

Aplicația este construită după modelul **MVVM (Model-View-ViewModel)**, o alegere firească în MAUI/Xamarin, care separă clar logica de prezentare de logică de interfață: - **Views** – reprezintă paginile și ecranele vizibile utilizatorului (definite în XAML și C#). Fiecare View corespunde de obicei unei pagini sau unui dialog din aplicație, de exemplu: LoginPage, RegisterPage, PlantsListPage, PlantDetailsPage, AddPlantPage, CalibrationPage, SettingsPage etc. Aceste pagini conțin declarativ elementele UI (liste, butoane, etichete, grafice). - **ViewModels** – clase C# asociate paginilor (conform convenției MVVM) care conțin *proprietăți* ce sunt legate (data binding) la elementele UI și *comenzi* (ICommand) care se declanșează la acțiuni ale utilizatorului (buton click, etc.). ViewModel-urile gestionează logica fiecărui ecran: preiau date de la servicii, le prelucrează dacă e cazul și actualizează proprietățile pentru UI. De exemplu, PlantsListViewModel va avea o colecție de plante de tip ObservableCollection<PlantModel> pe care o leagă la un control de listă din XAML. Va avea și comenzi precum AddPlantCommand (lansată când utilizatorul apasă butonul de adăugare). - **Servicii (Services)** – clase care se ocupă de funcționalitățile de nivel mai jos, cum ar fi comunicarea cu API-ul sau stocarea locală. Probabil există un ApiService sau SmartGardenApiClient care încapsulează apelurile HTTP către backend, astfel încât ViewModel-urile doar apelează metode precum apiService.GetPlantsAsync() sau apiService.WaterPlantAsync(id). Acest service folosește HttpClient pentru a face cereri REST și convertește JSON-ul primit în obiecte model (poate folosind System.Text.Json). - **Models** – modele folosite în aplicație. Acestea pot fi fie chiar DTO-urile folosite de API (PlantDto, SensorReadingDto etc.), fie modele simplificate adaptate UI-ului. În proiectul nostru, s-ar fi putut reutiliza direct SmartGarden.Core (dacă l-au referențiat, fiind tot .NET), ceea ce ar însemna că aplicația cunoaște exact clasele Plant, SensorReading etc. Dacă nu, s-au definit echivalent, sau s-a folosit dynamic.

Structura proiectului SmartGarden.App (așa cum se vede și în fișiere) este tipică: - Folder **ViewModels**: conține clase precum LoginViewModel.cs, PlantsViewModel.cs, PlantDetailsViewModel.cs, AddPlantViewModel.cs, CalibrationViewModel.cs etc. Fiecare expune proprietăți (de ex. PlantsList – lista de plante, IsBusy – stare pentru afișare loader, SelectedPlant etc.) și comenzi (ex. RefreshCommand, DeletePlantCommand, SavePlantCommand). - Folder **Views**: conține fișiere XAML și code-behind pentru pagini. De ex. PlantsListPage.xaml și .xaml.cs, unde în XAML definim UI-ul: o bară de căutare și filtrare, o listă (CollectionView) cu Template de item (nume plantă, cameră, icon indoor/outdoor), buton de adăugare plantă etc. Code-behind leagă ViewModel-ul (setat ca BindingContext). - Folder **Models**: modele de date (dacă sunt separate). - Folder **Services**: conține clasa care gestionează apelurile web. Acest service ar stoca de exemplu adresa serverului (ex. baseUrl = "https://smartgarden-api.azurewebsites.net") și ar folosi HttpClient global pentru cereri. Eventual, un alt service pentru gestionarea stocării locale (ex. preferințe în SecureStorage pentru token-ul JWT). - **Resources**: resurse precum imagini (de ex. icoane), fișiere traduceri (dacă e cazul), stiluri XAML etc. În fișiere observăm că există imagini (dotnet\_bot.png e default, probabil schimbate cu iconițe potrivite pentru proiect – pot fi de ex. pictograme pentru tipuri de plante sau senzori). - **Platforms**: cod specific pentru Android, iOS etc. De exemplu, AndroidManifest.xml unde s-au cerut permisiuni (poate acces la rețea – de obicei e implicit). Dacă s-a implementat scanarea rețelei Wi-Fi sau conectarea la AP, pe Android ar fi necesare permisiuni de locație (pentru Wi-Fi scan) – e posibil că modul de configurare Wi-Fi s-a făcut manual prin instructaj, deci nu direct din app (pentru a evita complicații de permisiuni). - **App.xaml**: definește resurse globale (culori, stiluri de controale). - **MauiProgram.cs**: configurează serviciile DI la nivel de aplicație mobilă. De obicei, se adaugă service-urile în containe (ex. builder.Services.AddSingleton<ApiService>(); builder.Services.AddSingleton<PlantsViewModel>(); builder.Services.AddTransient<PlantDetailsViewModel>(); ...). Singleton vs Transient e ales în funcție de necesitate (ex. un ViewModel per navigare nou). - **AppShell.xaml**: definirea structurii navigaționale a aplicației. Cel mai probabil, SmartGarden folosește Shell-ul MAUI pentru a organiza ecranele. În AppShell se vor defini rutele (ex. route "PlantsListPage", "PlantDetailsPage") și eventual un meniu flyout pentru navigare. De exemplu, s-ar putea avea:

<FlyoutItem Title="My Plants">  
 <ShellContent ContentTemplate="{DataTemplate local:PlantsListPage}" />  
</FlyoutItem>  
<MenuItem Text="Profile" Clicked="OnProfileMenuClicked"/>

etc. În design-ul dat, pare că meniul (profil) conține opțiuni: Calibrate sensors, Help, About Us, care probabil sunt legate la pagini separate. Shell permite definirea unui Flyout cu astfel de itemi.

Din perspectiva **UI/UX**, aplicația a fost concepută având în minte ușurința de utilizare și un aspect plăcut, modern. Design-ul (vezi PDF-ul SmartGarden.pdf) conturează principalele ecrane:

* **Ecran de autentificare (Login)**: primul ecran la deschiderea aplicației solicită utilizatorului să se conecteze sau să-și creeze un cont. Avem două entry-uri (Email și Password) și buton de "Login", plus un link/buton către "Create new account" pentru a naviga la ecranul de înregistrare. Scopul acestui ecran e securizarea accesului – doar utilizatorul logat își vede plantele. Odată ce login-ul are succes (verificat printr-un API call la /api/user/login), aplicația stochează eventual token-ul JWT primit și trece la ecranul principal (navigare către Shell).
* **Ecran de înregistrare (Register)**: similar, solicită username, email, parolă. Trimite la /api/user/register. După succes, probabil face automat login sau redirecționează la login.
* **Flux de configurare inițială a dispozitivului (Onboarding)**: Acesta este un aspect interesant al aplicației. După ce utilizatorul are cont și se loghează prima dată, dacă nu are plante adăugate, aplicația îl ghidează să configureze primul dispozitiv. Conform design-ului, există o secvență tip **wizard** cu mai mulți pași:
* *Introducere*: un ecran salut (ex. "Hello, {Name}. Let’s add your plant!") care menționează pașii: „1. Conectează dispozitivul la Wi-Fi, 2. Calibrează senzorii, 3. Adaugă detaliile plantei”. Utilizatorul apasă "Let’s get started!" pentru a începe.
* *Conectare la Wi-Fi*: aplicația cere utilizatorului să conecteze dispozitivul la rețeaua sa. Posibil pe acest ecran se afișează un textbox pentru SSID și parolă sau instructează userul să meargă în setările telefonului să se conecteze la AP-ul dispozitivului și apoi să revină. Un text mentionează explicit (make sure it’s 2.4GHz) deci clar userul trebuie să folosească rețeaua potrivită. Implementarea concretă: dacă s-a ales să se facă direct din app, pe Android se poate folosi Xamarin.Essentials pentru a deschide setările Wi-Fi (nu știu să scaneze rețele direct din MAUI ușor, deci e posibil să fi preferat instructaj manual).
* *Calibrarea senzorilor*: se parcurg mai multe ecrane sub-secvente:
  + Calibrare lumină: un ecran cu titlul "Calibrating the Light Sensors" și listă de nivele: Dark, Medium Indirect, Bright Indirect, Bright Direct, Low Light, cu starea fiecăruia (Completed/Calibrate) și indicații pentru utilizator despre ce să facă (e.g. "Pune senzorul în lumină indirectă puternică și apasă Calibrate"). Utilizatorul urmează instrucțiunile – fiecare dată când apasă "Calibrate" la un nivel, aplicația trimite comanda la dispozitiv și așteaptă confirmarea (poate de 5 secunde) apoi marchează nivelul ca Completed. După ce toate nivelurile necesare sunt Completed, butonul Next devine disponibil și utilizatorul trece mai departe. Observăm și un ecran intermediar (posibil mod pop-up) ce zice "Put the sensor in Bright Indirect Light - Wait for 5 seconds", deci aplicația afișează un countdown pentru timpul de așteptare la citire.
  + Calibrare sol: similar, ecran "Calibrating the Soil Sensor" cu două etape: Dry Soil (Completed după ce se face calibrul în aer sau sol uscat) și Wet Soil (necompletat la început). Utilizatorul este îndrumat "Pune senzorul în sol ud" apoi "Wait for 5 seconds" și apasă calibrate. Senzorul măsoară și se înregistrează. Next devine disponibil după completare.
  + Calibrare nivel apă: ecran "Calibrating the Water Sensor" cu două etape: Min și Max. Similar, instructează "Pune apă în recipient și așteaptă 5 secunde".
  + După calibrări, un ecran final "Setup Complete – Now, let's add a plant!".
* *Adăugarea plantei (detalii plantă)*: Ecranul "Add a plant" colectează informațiile despre planta monitorizată de dispozitivul abia configurat. Conține câmpuri precum:
  + Plant Type – un selector (dropdown) cu speciile disponibile (Ex. Ficus lyrata, Monstera, etc.). E probabil populat dintr-o listă predefinită (poate obținută de la server prin GET /api/species).
  + Place: Indoor/Outdoor – toggle sau picker.
  + Room – un selector de cameră (Bedroom, Living Room, Kitchen, Bathroom, etc.) – aceste opțiuni sunt văzute și în filtrele de listă deci e bine să fie consistente.
  + Date – data plantării/achiziției (un date picker).
  + Eventual un câmp de poreclă (Plants Nickname) apare în design la listă, dar nu e clar dacă e un field separat sau e același cu species (posibil nickname să fie dacă vrei să numești planta altfel, ex "Ficusul bunicii"). În design, la cardurile de plantă apare denumirea speciei și sub un text "Plants Nickname" – s-ar putea ca "Plants Nickname" să fie un placeholder în design, deci clar poate userul să pună un nume custom (ex "Freddy" pentru ficus).
* Ecranul are buton "Add Plant" și "Cancel" (poate dacă anulează, iese din flux). Când se apasă Add Plant, aplicația trimite datele la backend (crează planta) – apoi: - Asociază dispozitivul cu noua plantă (transmite plantId la device). - Navighează la ecranul principal (lista de plante), unde acum va apărea planta adăugată.

Acest flux de on-boarding asigură că un utilizator nou parcurge toți pașii necesari pentru a pune sistemul în funcțiune fără confuzie.

* **Ecran principal – Lista de plante**: După configurare (sau la utilizările ulterioare, direct după login), utilizatorul ajunge la ecranul "My Plants". Aici sunt afișate toate plantele înregistrate ale utilizatorului, într-o listă scrollabilă. Fiecare item are:
* Numele speciei și/sau porecla plantei,
* O etichetă Indoor/Outdoor,
* (Eventual ultima valoare de umiditate sau un status) – design-ul arată minimal, dar se pot adăuga iconițe eventual (ex. o picătură cu % umid).
* Posibil un icon de avertizare dacă ceva necesită atenție (design-ul listei nu arată explicit, dar conceptul ar putea exista).

Deasupra listei, există filtre și căutare: - O secțiune de filtrare: "All | OUTDOOR | INDOOR" – butoane toggle care filtrează plantele după locație. - O secțiune de filtrare după cameră: se vede o listă "Bedroom 1, Bedroom 2, Living Room, Kitchen, Bathroom". Probabil un control de tip picker multi-opțiune (sau orizontal scroll) pentru a filtra după RoomName. Dacă userul are definitor mai multe camere, apar listate. - Un câmp Search – pentru a căuta după nume (poreclă) sau specie. În design se vede "Search" ca placeholder. Aplicația ar putea filtra local lista de plante la tastarea în search bar (via CollectionView filtering).

De asemenea, un buton (ex. +) pentru adăugare plantă nouă (în design scrie "Add a new plant" în meniu sau un buton floating action). Un buton de meniu/profil (adesea un burger menu sau un avatar în colț) care deschide meniul lateral (Flyout). În design e menționat "Profile", deci se poate ca prin avatar se duce la profil (de unde calibrări, help, etc.).

Utilizatorul poate derula prin listă, atinge o plantă pentru a vedea detalii, sau folosi swipe pe item pentru acțiuni rapide: design-ul sugerează că la swipe apare opțiuni "Delete" și "Edit Plant", confirmate de modul "Do you want to delete the plant?". Astfel, implementarea CollectionView pe MAUI permite definirea context actions la swipe pe list items: unul pentru Edit (navighează la ecranul de editare – similar cu AddPlantPage dar prepopulat, dacă implementat), altul pentru Delete (care confirmă apoi apelează DeleteCommand ce face DELETE la API și elimină local itemul).

* **Ecran detalii plantă (Plant Details)**: Acesta este ecranul central de monitorizare, unde utilizatorul vede informațiile in timp real pentru planta selectată. Conform design-ului:
* În partea de sus, numele plantei (specie/poreclă) și eventual o pictogramă sau fotografie a plantei (design-ul textual nu indică clar imagine, dar ar fi drăguț, poate un placeholder).
* Un mesaj de alertă vizibil dacă e cazul – ex. "Please fill the water tank!" cu text mare, eventual icon roșu când nivelul apei < 10%. În design e plasat sub numele plantei.
* O prezentare tip **panou de valori**: valori numerice sau gauge pentru fiecare parametru monitorizat, aranjate poate în grid 2x2:
  + Water tank: 10% – nivelul apei (dacă sub un prag, e afișat și acel mesaj de alertă ca mai sus).
  + Light: 78 – intensitatea luminii. Ar putea fi % lumină relativă calibrată sau un indice lux. În design apare 78 fără unitate (posibil procent).
  + Temperature: 24°C. Aici clar e temperatura aerului, unitate grad Celsius.
  + Soil Moisture: 46% – umiditatea solului.
* Aceste valori sunt actualizate cu ultimele citiri. Se pot reprezenta fie pur numeric (cum arată design-ul textual), fie cu mici gauge (de exemplu, un semicerc ce arată procentul). Dar dat fiind MVP, numeric e suficient, cu cod culoare eventual (ex. albastru pentru umiditate, galben pentru lumină). - Un toggle pentru udare automată: În design, se vede "Automatic watering" cu butoane ON/OFF. Ar putea fi un **Switch** control de fapt, care e verde când ON și gri când OFF. Utilizatorul poate atinge pentru a schimba starea. Aplicația va executa AutoWateringEnabled = !AutoWateringEnabled în ViewModel și va trimite la server actualizarea. Rezultatul se reflectă imediat vizual. - Buton de udare manuală: Acest lucru nu e explicit desenat în design-ul textual, însă e posibil ca *însuși switch-ul ON/OFF să permită și control manual*. E interpretarea: Poate "ON"/"OFF" la stânga/dreapta sunt butoane de pornire/oprire manuală a pompei (deși scrie Automatic watering sub ele, e puțin confuz textul). O altă interpretare: erau 3 butoane – "ON", "OFF" (ambele pentru pompa manual, on/off direct), și "Automatic watering" text label pentru switch. Dar mai logic e: *un switch with labels* – posibil template-ul a extras “ON” și “OFF” separate. Dacă intr-adevăr sunt separate, ar fi: - "ON" – utilizator apasă ca să pornească pompa manual (udare manuală imediat, poate butonul devine "ON (active)" și se oprește după durata implicită). - "OFF" – același buton sau alt buton ca sa oprești manual pompa (dacă ar fi pornită continuu? dar pompa oricum e temporizată, nu rămâne pornită indefinit). Deci cred că e de fapt un toggler: în design text avem:
* ON  
  OFF  
  Automatic watering
* Eu cred că e interpretarea nepotrivită a textelor din design. Cel mai probabil e un **Switch** control cu text "Automatic watering" la dreapta, și starea lui (când On arată text ON la stânga și OFF la dreapta, extrase). Deci cred că e doar switch-ul modul automat. Astfel, *udarea manuală ar trebui un alt element*, probabil un buton cu icon (stropitoare) undeva. Poate l-au plasat lângă valorile senzorilor sau sus. Dacă nu există buton dedicat, se poate realiza un gesture: de ex. dacă userul apasă pe icon-ul "Water tank 10%", se declanșează manual udarea. Dar asta ar fi obscur. Mai probabil, design-ul nefiind clar textual, vom presupune că s-a implementat un buton "Water now" vizibil lângă "Automatic watering" toggle. Pentru documentație, putem preciza că aplicația oferă un buton de udare manuală.
* Secțiunea **Weekly Stats**: în design se vede o arie care listează zilele săptămânii (Mo, Tue, Wed, ... Sun) și pare să aibă niște bare sau procente dedesubt. Aceasta corespunde unui **grafic săptămânal**. Cel mai probabil, graficul arată variația unui parametru de interes de-a lungul ultimelor 7 zile, probabil umiditatea solului (sau volum de apă administrat). Fiind un MVP, s-ar fi putut alege cel mai reprezentativ: umiditatea solului medie zilnic. Astfel, sub fiecare abreviere de zi, putem avea o bară sau un procent numeric ce indică nivelul. Din extragerea textului, vedem sub Mo, Tue etc. niște spații – posibil bare vizuale (care nu apar textual). Dacă s-a folosit vreo librărie de chart: .NET MAUI are *Microcharts* (o bibliotecă ușor de folosit pentru grafice simple), sau Syncfusion Charts (dacă licența de student e ok). Microcharts ar putea genera un grafic linie sau coloană destul de rapid. Alternativ, se putea implementa manual cu shapes (drepte colorate). Graficul permite utilizatorului să vadă tendința și eficacitatea sistemului. De ex., dacă vede că în weekend solul a rămas la 50% (planta poate n-a băut mult) iar joi a scăzut brusc la 25% după o zi însorită, acest tip de vizualizare e util. Weekly Stats e o funcționalitate ce îmbunătățește experiența față de a vedea doar ultima valoare.
* **Meniu lateral / Profil**: Aplicația are un meniu (accesibil probabil printr-un buton hamburger sau glisare din stânga) cu opțiuni:
* Profile – poate afișează numele utilizatorului și eventual posibilitatea de logout.
* Add a new plant – scurtătură pentru a porni fluxul de adăugare a unei alte plante/dispozitiv (util dacă userul cumpără al doilea kit SmartGarden).
* Calibrate sensors – această opțiune permite recalibrarea senzorilor unui dispozitiv. Dacă userul observă citiri incorecte sau mută planta în alt sol, poate vrea să recalcibreze. Atingând Calibrate, probabil aplicația cere alegerea plantei/dispozitivului (dacă sunt mai multe) și reia procesul de calibrare lumina/sol/apă pentru acel device, comunicând direct cu el (poate trebuie să fie aproape, conectat local – sau device-ul însuși poate intra în modul calibrate la comanda serverului). Ca implementare, mai simplu e să pună userul din nou să se conecteze la AP-ul device-ului, deci calibrările post-setup ar necesita un proces similar onboarding-ului. Posibil în MVP nu s-a implementat complet calibrări ulterioare, dar opțiunea e prevăzută.
* Help – trimite la o pagină de ajutor, care probabil conține întrebări frecvente sau instructaj (ex. "Cum calibrez senzorii? Cum schimb Wi-Fi? etc."). Ar putea fi o pagină statică cu text sau link către un PDF/manual.
* About Us – o pagină despre autorii proiectului, versiune, mulțumiri etc. Standard, static text.

Meniul e un element de **navigație globală**. Poate conține și buton de Logout la final.

* **Notificări**: Aplicația folosește cel puțin **notificări locale** pentru a alerta utilizatorul de evenimente critice:
* Rezervor de apă gol – de exemplu, dacă aplicația e în background, ar fi ideal să apară o notificare: "Rezervorul pentru Ficus lyrata este gol! Reumpleți-l.". În MVP, dacă nu s-au implementat push, se pot programa notificări locale cu Xamarin.Essentials: pe Android, se pot folosi servicul local notify. Dacă nu, la minim, utilizatorul vede în aplicație clar mesajul.
* Sol foarte uscat și modul automat oprit – eventual o notificare "Planta Monstera are solul uscat, ud-o sau activează modul automat."
* Alte alerte (temperatură extremă scăzută/înaltă) – e planificat.

Implementarea notificărilor locale necesită permisiuni și platform-specific coding (ex. dependency service). E posibil ca din lipsă de timp, s-a amânat implementarea push/locale și s-a preferat varianta că userul trebuie să deschidă aplicația pentru a vedea alertele. Totuși, dat fiind accentul, e menționat "notificări" în cerințe, deci ceva tot s-a făcut. Un mod simplu: când aplicația e pornită și primește date ce indică alertă, afișează un pop-up (DisplayAlert) imediat. Deci pseudo-notificare in-app. Pentru licență, putem menționa planul de a integra notificări push în viitor (ex. folosind Azure Notification Hubs sau Firebase).

* **Grafic & vizualizări**: Ca mențiune de implementare, e posibil s-a utilizat pachetul *Microcharts* pentru graficul săptămânal, și *SkiaSharp* pentru eventuale gauge (dar nepotrivit, cred numeric e ok). Metricele sunt destul de clar prezentate textual pentru MVP, oricând se pot cosmetiza.
* **Experiența utilizator**: UI/UX design-ul a încercat să fie prietenos:
* Paleta de culori probabil neutră, cu accent pe verde (asociat cu plantele) sau albastru (tehnologie). PDF-ul nu ne arată culori, dar putem presupune un verde pentru highlight (ex. switch ON verde).
* Iconografie: pictograme intuitive (o frunză pentru plante, un termometru pentru temperatură, o picătură pentru umiditate, un soare pentru lumină, o pompă pentru rezervor). Acestea pot apărea lângă valori sau ca parte a designului. De ex. "Water tank 10%" poate fi afișat cu o iconiță de rezervor.
* Feedback vizual: butoanele schimbă culoare la apăsare, există *ActivityIndicator* (spinner) când se fac solicitări la server (ex. la refresh al listei sau login).
* Responsivitate: prin MAUI, interfața se adaptează la dimensiuni diferite de ecran. S-a testat pe un smartphone cu rezoluție medie (approx. 6 inch). Componentele precum list sau grid-ul la detalii se aranjează fluid.
* Accesibilitate: text suficient de mare, contrast bun (ex. text alb pe fundal colorat pentru alerte), etc.
* **Interacțiunea cu backend**: Aplicația folosește un modul HTTP (HttpClient). De exemplu, în ApiService:
* private readonly HttpClient \_http = new HttpClient();  
  public ApiService() {  
   \_http.BaseAddress = new Uri("https://smartgardenapi.com/");  
  }  
  public async Task<List<PlantDto>> GetPlantsAsync() {  
   var token = Preferences.Get("auth\_token", null);  
   if(token != null)   
   \_http.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", token);  
   var response = await \_http.GetAsync("api/plant");  
   response.EnsureSuccessStatusCode();  
   var json = await response.Content.ReadAsStringAsync();  
   return JsonSerializer.Deserialize<List<PlantDto>>(json);  
  }
* Asemănător pentru celelalte operații (post, put, delete). Aceste metode sunt chemate din ViewModel. De exemplu, PlantsListViewModel la inițializare sau la refresh:
* async Task LoadPlants() {  
   IsBusy = true;  
   var plants = await \_apiService.GetPlantsAsync();  
   PlantsList.Clear();  
   foreach(var plant in plants)  
   PlantsList.Add(plant);  
   IsBusy = false;  
  }
* care se leagă la UI: IsBusy poate activa un spinner, PlantsList populat actualizează lista (BindingContext + ItemsSource in XAML).

La acțiuni precum Delete:

public ICommand DeletePlantCommand => new Command<PlantDto>(async plant => {  
 bool confirmed = await Shell.Current.DisplayAlert("Confirm", $"Delete {plant.Nickname}?", "Yes", "No");  
 if(!confirmed) return;  
 await \_apiService.DeletePlantAsync(plant.PlantId);  
 PlantsList.Remove(plant);  
});

Observăm folosirea Shell.Current pentru a apela DisplayAlert crossplatform.

* **Componente de grafic**: Un snippet potențial pentru populare grafic:
* var weeklyData = await \_apiService.GetWeeklyDataAsync(plantId);  
  // weeklyData e o listă de 7 perechi (zi, valoare medie umiditate).  
  ChartEntries = weeklyData.Select(d => new ChartEntry(d.Value) {  
   Label = d.DayShortName, ValueLabel = d.Value.ToString()  
  }).ToArray();  
  OnPropertyChanged(nameof(ChartEntries));
* Apoi în XAML se poate folosi un control Microchart:
* <microcharts:ChartView Chart="{Binding SoilMoistureChart}" HeightRequest="100"/>
* unde SoilMoistureChart e un obiect LineChart construit din ChartEntries. Dacă nu s-a folosit, se putea manual:
* Create horizontal StackLayout cu colorații de bara:
* <StackLayout Orientation="Horizontal" HorizontalOptions="Center">  
   <Grid>  
   <!-- for each day -->  
   <BoxView HeightRequest="{Binding PercentageOfMax \* 100}" WidthRequest="10" Color="Blue"/>  
   <Label Text="{Binding Day}" VerticalOptions="End"/>  
   </Grid>  
   ...  
  </StackLayout>
* Aceasta ar arăta bare.

### 7.2 Funcționalități cheie implementate în aplicație

Recapitulând, aplicația mobilă oferă următoarele **funcționalități cheie**: - Autentificare securizată (login/logout, înregistrare cont nou). - Configurarea inițială a unui dispozitiv SmartGarden (conectare la Wi-Fi, calibrare senzori). - Adăugarea unei plante în sistem cu toate detaliile necesare. - Listarea plantelor utilizatorului și filtrarea acestora după criterii (interior/exterior, cameră, căutare text). - Vizualizarea detaliată a stării unei plante: valori actuale de la senzori (umiditate sol, lumină, temperatură, nivel apă), grafic cu istoricul recent, ultima dată când a fost udată, etc. - Activarea/dezactivarea modului de udare automată pentru plantele utilizatorului. - Declanșarea manuală a udării unei plante (butonul “Udă acum”). - Notificarea utilizatorului în aplicație despre situații ce necesită atenție (rezervor gol, sol prea uscat când auto este oprit). - Editarea sau ștergerea plantelor existente. - Recalibrarea senzorilor la nevoie (prin meniul calibrate). - Pagini de ajutor și despre, pentru suport și informații contextuale.

Multe dintre aceste funcționalități au fost gândite conform **cerințelor MVP**, iar aplicația este pregătită să fie extinsă ușor. De exemplu, dacă se adaugă noi tipuri de senzori sau parametri (să zicem senzor de fertilitate sol), aplicația poate afișa un nou card de valoare la detalii plantă fără a schimba arhitectura (doar adăugând binding spre noua valoare din DTO). Sau dacă se implementează sugestii inteligente (tips & tricks) personalizate, aplicația poate primi de la server un text de recomandare pentru fiecare plantă (ex: "Mută planta la lumină mai puternică" dacă se observă lumina mereu scăzută) și să-l afișeze într-o secțiune.

Câteva elemente de cod relevante pot fi utile pentru a exemplifica anumite mecanisme:

* **Navigare cu Shell**: .NET MAUI Shell face navigarea foarte simplă:
* // din RegisterPageViewModel, după ce se creează contul și face login auto:  
  await Shell.Current.GoToAsync("//PlantsListPage");
* Asta face navigare la ruta principală (lista plante), eliminând din stivă paginile de autentificare (notarea '//' indică navigare la un shell section root). Navigarea la detalii se poate face tot cu route:
* // In PlantsListViewModel, on selecting a plant  
  await Shell.Current.GoToAsync($"PlantDetailsPage?plantId={selected.PlantId}");
* Iar Shell cunoaște ruta cu parametru plantId și creează pagina de detalii, injectând parametru (ViewModel-ul paginii de detalii poate primi param via QueryProperty).
* **Data Binding exemple (XAML)**:
* <!-- In PlantDetailsPage.xaml -->  
  <Label Text="{Binding SoilMoisture, StringFormat='{0:F0}%'}" FontSize="Large"/>  
  <Label Text="Soil Moisture" FontSize="Small"/>  
  <Switch IsToggled="{Binding AutoWateringEnabled}" />
* Un astfel de XAML va lega proprietatea SoilMoisture (double) din ViewModel, afișând-o rotunjită ca procent, și un Switch la booleana AutoWateringEnabled.
* **Comandă buton manual watering** (dacă e implementat separat):
* <Button Text="Water Now" Command="{Binding ManualWaterCommand}" IsEnabled="{Binding WaterTankLevel, Converter=NonZeroToBoolConverter}"/>
* Aici butonul e dezactivat dacă rezervorul e gol (WaterTankLevel 0%). ManualWaterCommand în ViewModel:
* public ICommand ManualWaterCommand => new Command(async () => {  
   IsBusy = true;  
   await \_apiService.TriggerManualWaterAsync(PlantId);  
   await LoadLatestData(); // reîmprospătează valorile, va reflecta noua udare  
   IsBusy = false;  
   await Application.Current.MainPage.DisplayAlert("Udare manuală", "Planta a fost udată manual.", "OK");  
  });
* **Refresh pull-to-refresh**: CollectionView in MAUI suportă refresh:
* <CollectionView ItemsSource="{Binding PlantsList}" IsRefreshing="{Binding IsBusy}" RefreshCommand="{Binding RefreshCommand}" ...>  
  ...  
  </CollectionView>
* Astfel, userul trage lista în jos, pornește RefreshCommand care în ViewModel face LoadPlants(). IsBusy true/false va porni/opri spinnerul de refresh.

În ansamblu, aplicația mobilă se comportă ca un **dashboard personal pentru grădina inteligentă** a utilizatorului. Ea este menită să ruleze în mod persistent pe telefonul utilizatorului, astfel încât acesta să poată verifica oricând plantele și să primească eventuale notificări. Datorită implementării cross-platform, utilizatorul își poate instala aplicația atât pe telefonul personal Android, cât și pe un iPad de exemplu, folosind același cont pentru a accesa datele (serverul menține totul sincronizat).

Performanța aplicației este bună, întrucât datele transferate sunt mici (câțiva KB per request) și actualizările nu sunt foarte frecvente. Interfața e fluidă, iar .NET MAUI asigură comportament nativ (liste scroll rapide, elemente UI conforme cu platforma).

În concluzie, aplicația mobilă SmartGarden acționează ca **punctul de control centralizat pentru utilizator**, având un design centrat pe simplitate și claritate, dar oferind toate informațiile necesare pentru a monitoriza și controla sistemul. Ea completează armonios componenta hardware și backend-ul: hardware-ul furnizează datele, backend-ul ia decizii, iar aplicația prezintă și permite intervențiile utilizatorului. Fără aplicație, utilizatorul nu ar avea vizibilitate în “cutia neagră” a dispozitivului, așa că aceasta aduce **transparență** (prin afișarea valorilor și log-urilor) și **control** (prin comenzi și setări), elemente esențiale pentru încrederea și adoptarea sistemului.

## 8. Funcționalități MVP vs extinderi ulterioare

Proiectul SmartGarden a fost dezvoltat inițial cu un set de **funcționalități de bază (MVP – Minimum Viable Product)** care să demonstreze conceptul și să acopere nevoile primare de monitorizare și udare automată. Pe lângă acestea, s-au identificat și planificat o serie de **extensii și îmbunătățiri** ce pot fi adăugate ulterior pentru a crește inteligența și utilitatea sistemului. Vom delimita clar ce este inclus în versiunea actuală și ce este prevăzut pentru viitor.

**Funcționalități incluse în MVP:** - *Monitorizarea parametrilor de mediu* – sistemul măsoară și afișează valorile de umiditate a solului, lumină, temperatură și umiditate aer, precum și nivelul apei din rezervor. Acestea sunt disponibile în timp real în aplicație și se salvează istoric (cel puțin pe termen scurt, pentru grafic săptămânal). - *Udarea automată pe bază de prag* – principalul selling point al SmartGarden. În MVP, dacă udarea automată este activă și solul devine mai uscat decât pragul prestabilit (ex. 30%), sistemul pornește pompa automat pentru o scurtă durată, umezind solul. Această funcție a fost testată practic și asigură menținerea plantelor într-un interval sănătos de umiditate fără intervenție umană. - *Controlul manual al udării* – utilizatorul poate oricând interveni: fie oprește modul automat (dacă vrea să gestioneze personal), fie declanșează manual o udare (de exemplu când vrea să ude suplimentar sau să verifice funcționarea). Această comandă manuală se poate da din aplicație printr-un buton dedicat. Sistemul o execută și loghează acțiunea ca atare (mode "MANUAL"). - *Gestionarea mai multor plante* – aplicația suportă adăugarea și monitorizarea mai multor plante/dispozitive simultan. Utilizatorul poate avea, de exemplu, 3 ghivece inteligente SmartGarden, fiecare cu senzorii săi, conectați la același cont. Aplicația listează toate plantele și permite accesul la detaliile fiecăreia. Astfel, sistemul este scalabil la nivel de utilizator (limitat eventual doar de capacitatea serverului de a gestiona datele). - *Configurare intuitivă (onboarding)* – MVP include procesul guidat de configurare inițială, ca utilizatorul să poată pune în funcțiune dispozitivul fără cunoștințe tehnice. Acest proces pas cu pas (Wi-Fi, calibrare, adăugare plantă) asigură că chiar și un utilizator neexperimentat poate folosi sistemul. - *Interfață prietenoasă și informativă* – Aplicația mobilă MVP are un UI bine structurat care arată clar informațiile (inclusiv alerte), cu elemente de filtrare și sortare ce fac ușoară gestionarea listelor de plante. De asemenea, include secțiuni ca *Help* și *About* pentru asistență minimală și informații despre aplicație. - *Notificări de bază* – Cel puțin sub formă de mesaje vizibile în aplicație, MVP notifică utilizatorul despre condiții critice: de exemplu, textul de avertizare privind rezervorul gol sau eventual un mesaj dacă modul automat e off dar planta necesită apă. Aceste notificări asigură că utilizatorul nu ratează aspecte importante. - *Stocarea datelor și vizualizare istoric* – Valorile înregistrate sunt salvate în baza de date, permițând afișarea istoricului recent (grafic săptămânal). Deși MVP nu include analize avansate, oferă totuși utilizatorului posibilitatea de a vedea evoluția umidității solului în ultimele zile, de exemplu. Jurnalul udărilor permite utilizatorului să știe când a fost udată planta ultima dată, ceea ce e util pentru încrederea în sistem. - *Managementul contului de utilizator* – Crearea de cont și autentificarea sunt incluse, astfel încât datele să fie securizate și fiecare utilizator să acceseze doar plantele sale. - *Calibrarea senzorilor* – O componentă vitală, calibrările sunt incluse în MVP pentru a asigura acuratețea citirilor. Fără calibrare, valorile brute ale senzorilor pot fi înșelătoare, deci a fost considerată parte din produsul minim viabil. Acest lucru diferențiază SmartGarden de soluții fixed-threshold necalibrate, aducând un plus de precizie.

Altfel spus, MVP-ul a fost construit suficient de robust pentru a fi **folosibil cap-coadă**: un utilizator poate cumpăra kitul, îl configurează, iar de acolo sistemul își face treaba (udă automat) și utilizatorul primește informațiile necesare în app.

**Extensii ulterioare planificate:** - **Integrarea AI (Inteligență Artificială)** – Aceasta este o direcție majoră de extindere. Există mai multe moduri în care AI ar putea fi folosit: - *Analiza datelor și predicții*: Folosind algoritmi de machine learning, SmartGarden ar putea învăța din istoricul fiecărei plante pentru a prezice când va trebui udată data viitoare. De exemplu, bazat pe rata de scădere a umidității, ar putea estima "următoarea udare peste ~2 zile". Aceste predicții ar putea fi afișate utilizatorului (ex: "Next watering expected Monday") sau folosite de sistem pentru a adapta modul automat (uda cu puțin înainte ca planta să atingă pragul, preemptiv, pentru a evita stresul). - *Detectarea anomaliilor*: un algoritm AI ar putea detecta dacă senzorii dau date anormale (ex. se degradează, se scurtcircuitează) sau dacă comportamentul plantei se schimbă (de exemplu, are nevoie de apă mai des decât înainte – poate a crescut și are rădăcini mai multe, deci consuma apă mai rapid). - *Asistent virtual pentru îngrijire plante*: O idee de extindere ar fi un chatbot sau un modul AI (gen GPT) integrat în aplicație care să răspundă întrebărilor utilizatorului despre plante („De ce îmi cad frunzele la plantă?”, „Cum tratez dăunătorii?” etc.) sau care să analizeze parametrii și să dea recomandări personalizate. De exemplu: "Observ că lumina primită de această plantă este mereu foarte scăzută, deși specia preferă lumină medie. Recomand să o muți mai aproape de fereastră." - *Vedere computerizată*: Dacă s-ar extinde hardware-ul cu o cameră, AI-ul ar putea analiza imagini cu planta pentru a detecta semne de ofilire, îngălbenire, mucegai pe sol etc., și să alerteze utilizatorul. Aceasta e o extensie hardware+AI foarte interesantă, dar dincolo de MVP.

* **Exportul datelor** – Mulți entuziaști de plante ar putea dori să analizeze datele lor sau să le păstreze. O funcție planificată este posibilitatea de a **exporta istoricul datelor** (de ex. toate citirile sau log-urile de udare) într-un format CSV sau Excel. Astfel, utilizatorul poate vizualiza offline graficul complet, poate face statistici personalizate sau poate împărtăși datele cu alți pasionați. Implementarea ar implica fie generarea fișierului în aplicație (pe baza datelor extrase de la API), fie API-ul oferă un endpoint /api/export care generează și trimite un CSV. Această caracteristică e utilă și pentru debugging – utilizatorul/proiectantul vede clar cum au variat parametrii.
* **Predicții inteligente** – Legat de AI, dar chiar și fără machine learning complex, se pot adăuga **predicții simple**. De exemplu:
* Folosirea datelor meteorologice pentru plantele outdoor: integrarea cu un API meteo (OpenWeatherMap etc.) astfel încât, dacă se anunță ploaie, sistemul să amâne udarea (sau să-l notifice pe user: "Ploaie prognozată, sistemul nu va uda astăzi"). Invers, pe caniculă prelungită, să sugereze udări mai dese.
* Predicție de secare: "Cu ritmul actual de scădere a umidității, solul va atinge pragul de uscare în ~18 ore."
* **Widget-uri mobile** – Pentru un acces și mai rapid la informații, s-a propus dezvoltarea de **widget-uri** pentru ecranul principal al telefonului (Android/iOS). Un widget ar putea afișa cele mai importante date fără să mai deschizi aplicația: de exemplu, pentru fiecare plantă un mic widget cu numele plantei și pictograme cu umiditatea solului și nivelul de apă. Astfel, utilizatorul își vede plantele dintr-o privire pe home screen. Implementarea de widget cross-platform e posibilă cu .NET MAUI (folosind platform specific code pentru Android AppWidget și iOS widget extension). E ceva mai avansat, deci lăsat ca extensie.
* **Capabilități multi-user și share** – Un potențial upgrade ar fi posibilitatea de a **partaja accesul** la plante. De exemplu, într-o familie, soțul și soția ar vrea amândoi acces la plantele de acasă. Asta ar implica fie cont comun, fie un sistem de invitații (owner-ul poate invita alt user cu cont SmartGarden să aibă acces la o plantă sau la toată grădina sa). MVP nu include așa ceva, dar e o idee de extindere socială a platformei.
* **Integrări cu alte platforme IoT** – Extensii posibile includ:
* Suport pentru control prin asistenți vocali precum Alexa, Google Assistant – de exemplu "Alexa, ask SmartGarden how my fern is doing" sau "Google, water the Monstera". Ar necesita expunerea API-ului printr-un skill/agent și autentificări Oauth.
* Integrare cu Home Assistant / openHAB – pentru entuziaștii IoT, să poată integra SmartGarden în hub-ul lor de casă inteligentă, poate prin MQTT sau API direct.
* **Hardware extins** – Chiar dacă e mai mult de partea hardware, e tot o planificare:
* Adăugarea de senzori noi: de pildă, senzor de nutrienți în sol (EC sensor) care ar putea indica când trebuie fertilizat. Aplicația ar putea afișa un parametru "Soil Fertility" și notifica "Ar trebui să fertilizezi planta".
* Modul de lămpi UV sau ventilatoare controlate – de exemplu, să aprindă o lampă de creștere dacă lumina e insuficientă, controlate tot de ESP32. Aplicația ar putea avea opțiuni de setare a iluminării.
* Conectarea la sisteme mai mari de irigație (dacă extindem la grădină în aer liber).
* **Optimizări UI/UX**:
* Mod de vizualizare sub formă de grafic complet pentru fiecare parametru (nu doar weekly summary). De exemplu un ecran unde userul poate vedea grafice pe ultimele 24h sau 30 zile, selecție interval etc. Asta vine la pachet cu stocarea datelor pe termen lung și posibil spațiu (poate un modul de arhivare).
* Profile predefinite de udare – userul ar putea alege modul "Vacanță" unde sistemul udă mai rar (pentru a economisi apă, acceptând plantele puțin mai uscate), sau modul "Boost" unde udă mai mult (poate la transplantare când planta are nevoie de mai multă apă). Aceste profiluri ar ajusta parametrii (prag, durată, frecvență) dinamic, și userul le-ar comuta ușor.
* **Secțiunea "Tips & Tricks"** – Apare menționată în design-ul UI. Aceasta poate fi o zonă în aplicație unde utilizatorul primește sfaturi generice de îngrijire. În extindere se poate personaliza: tipuri de conținut:
* Sfaturi contextuale: dacă lumina e mereu mică -> "Această plantă ar prefera mai mult soare". Dacă e iarnă și temp scade sub 18°C -> "Ficusul nu iubește frigul, mutați-l departe de geam".
* Articole scrise (un mini-blog) integrat în app, cu categorii (udare, lumină, fertilizare, boli) – pot fi updatate via server.
* Aceasta poate fi combinată cu AI (asistent personalizat).
* **Profil utilizator extins** – de exemplu, posibilitatea de a schimba tema aplicației (dark mode, etc.), de a primi rapoarte periodice pe email (export automat săptămânal), eventual integrare cu calendar (să pună reminder de fertilizare odată la X luni).
* **Siguranță și protecții** – Extinderi precum:
* Alerte de inundație: dacă vreun senzor detectează apă în tava ghiveciului (ex. un senzor suplimentar), oprește udarea și anunță user.
* Backup offline: în caz că internetul pică, să stocheze datele local pe device și să le sincronizeze ulterior (de ex. device-ul ține ultimele N citiri și la reconectare le trimite).
* OTA updates: posibilitatea de a actualiza firmware-ul ESP32 de la distanță prin API – cam complicat de implementat, dar e un plus valabil dacă se face product serios.

E clar că direcțiile de extindere sunt numeroase și pot transforma SmartGarden dintr-un sistem de udare automată simplu într-o platformă complexă de *smart farming* la scară mică. Prioritizarea acestor extinderi ar depinde de feedback-ul utilizatorilor reali: - De exemplu, dacă utilizatorii arată interes major pentru *analiza datelor*, se va prioritiza exportul și grafice avansate. - Dacă feedback-ul evidențiază economie de apă sau integrare meteo, se vor implementa predicțiile cu ploaie. - Integrarea AI ar necesita date suficiente, deci e posibil să se facă într-o etapă ulterioară după ce sistemul a colectat destule informații.

În concluzie, **MVP-ul SmartGarden acoperă fundamentul**: monitorizare precisă și udare automată fiabilă, administrate printr-o aplicație ușor de folosit. **Extensiile planificate** vor aduce sistemul la următorul nivel de inteligență și conectivitate: predicții prin AI, conectare cu alte servicii, mai multă personalizare și conștientizare pentru utilizator. Aceste direcții vor transforma SmartGarden treptat dintr-un instrument reactiv într-unul proactiv, adaptiv și integrat în ecosistemul digital al casei inteligente.

## 9. Design UI/UX

Interfața cu utilizatorul (UI) și experiența utilizatorului (UX) în SmartGarden au fost proiectate cu atenție pentru a fi **intuitive, prietenoase și plăcute vizual**, având în vedere că aplicația se adresează inclusiv persoanelor fără pregătire tehnică. Design-ul mockup furnizat (SmartGarden.pdf) a stat la baza implementării UI, asigurând o coerență și un aspect modern. În această secțiune vom prezenta principalele ecrane ale aplicației, modul de interacțiune și rațiunile de design din spatele lor, conform mockup-ului.

**Palette și identitate vizuală:** Deși PDF-ul este textual, ne imaginăm că s-a folosit o paletă de culori pastelate, cu accente de verde (sugerând natura, creștere) și albastru (pentru componentele de apă și tehnologie). Fundalurile sunt probabil deschise (alb sau gri foarte deschis), textul de culoare închisă pentru lizibilitate, iar elementele interactive (butoane, switch-uri) sunt evidențiate într-o culoare de accent (verde sau albastru). Iconografia este simplă și ușor de recunoscut: de exemplu, pictograma de lumina (soare), termometru pentru temperatură, picătură pentru umiditate, stropitoare pentru udare. Fontul folosit este unul modern, fără serife (precum Open Sans, care apare și în resursele proiectului), asigurând un aspect curat.

Vom trece prin fluxul principal al aplicației așa cum îl vede utilizatorul:

* **Ecranul de *Login*** – Este primul ecran întâmpinat de utilizator la deschiderea aplicației (dacă nu este deja autentificat). Design-ul acestuia este minimalist: logo-ul sau numele aplicației sus, și două câmpuri de input centrate în pagină: *Email* și *Password*, urmate de butonul principal **Login**. Sub buton, există un link/buton secundar **Create new account** pentru a naviga spre ecranul de înregistrare. Acest aranjament este standard și ușor recognoscibil, reducând din frustrare la prima utilizare. Butonul de login probabil devine activ doar după ce sunt completate câmpurile, iar dacă acreditările sunt greșite, va apărea un mesaj de eroare clar (ex. "Email sau parolă incorecte"). Scopul a fost ca autentificarea să fie cât mai simplă, așa că s-au cerut doar email și parolă – suficient pentru securitate, fără pași complicați în plus. Estetic, s-a păstrat mult spațiu alb (empty space) pentru un aspect aerisit; câmpurile au etichete și placeholder vizibile pentru a ști exact ce să introducă. În termeni de UX, timpul petrecut pe acest ecran trebuie să fie minim – după login reușit, utilizatorul e dus imediat în aplicație, eventual cu o mică animație de tranziție.
* **Ecranul de *Înregistrare (Create Account)*** – Similar în stil cu login-ul, acest ecran solicită câteva informații pentru a crea cont: un *Username* (nume afișat), *Email* și *Password*, plus confirmare parolă dacă s-a considerat necesar. Butonul este **Create new account** proeminent, iar link-ul de sub el permite să meargă înapoi la Login (pentru cei care și-au amintit că aveau cont). UX-ul aici a fost gândit să fie fără frecare: doar 3 câmpuri de completat și gata. Probabil există și verificări vizuale (ex. dacă email-ul nu e valid format, evidențiază câmpul cu roșu). După crearea contului cu succes, aplicația ar putea face automat login și trece la onboarding.
* **Fluxul de *Onboarding și Configurare Dispozitiv*** – Acesta este un element definitoriu al SmartGarden, asigurând că utilizatorul configurează corect hardware-ul. În design, fluxul e împărțit în ecrane secvențiale:
* **Ecranul de Bun Venit** – după login sau creare cont, dacă nu există plante configurate, utilizatorul vede un ecran de întâmpinare: *"Hello, {Alexandra}"* (numele de utilizator inserat din cont) și un mesaj motivant *"Let’s add your plant!"*. Urmează o listare a pașilor de configurare, sub forma a trei bullet-uri iconizate:
  + *"Connect your device to Wi-Fi"* – cu sublinierea (make sure it’s 2.4GHz) pentru a preveni confuzia rețelelor de 5GHz.
  + *"Calibrate the sensors"* – indicând că va trebui făcut acest pas.
  + *"Add your plant details"* – ultimul pas, de completare a informațiilor despre plantă.
* Apoi, un buton call-to-action **“Let’s get started!”** invită utilizatorul să înceapă configurarea. Acest ecran este esențial din perspectivă UX pentru a seta așteptările – utilizatorul înțelege că urmează 3 etape și de ce sunt importante, reducând anxietatea în proces (știind mereu unde se află și câți pași au mai rămas). Vizual, e probabil un layout centrat, cu iconițe drăguțe pentru Wi-Fi, senzori, ghiveci, pentru a ilustra fiecare pas. Tonul mesajului este prietenos și personalizat (folosind numele utilizatorului).
* **Ecranul de Conectare la Wi-Fi** – Odată ce începe, aplicația va afișa instrucțiuni pentru a conecta dispozitivul SmartGarden la rețeaua de domiciliu. Deși nu vedem textual explicit un ecran în PDF exceptând bullet-ul, cel mai probabil designul conține:
  + Un câmp pentru SSID (sau un dropdown cu rețele disponibile – deși accesul la lista Wi-Fi e complicat pe iOS, deci posibil doar un text input).
  + Un câmp pentru parola Wi-Fi.
  + Poate un buton "Connect" care trimite aceste credențiale către dispozitiv.
  + În plus, indicații: *“Asigură-te că telefonul este conectat la rețeaua Wi-Fi de 2.4GHz la care vrei să conectezi dispozitivul. Dacă dispozitivul are un LED, asigură-te că clipește albastru etc.”*.
* Dacă procesul se face manual (prin intrarea în modul AP), poate ecranul spune: *“Conectează-te la rețeaua Wi-Fi SmartGarden-xxxx a dispozitivului (parolă implicită....), apoi revino aici pentru a continua.”* După ce userul se conectează la AP-ul device-ului, butonul Connect trimite datele.
* Ca UX, acest pas este cel mai tehnic, deci a fost ținut cât mai simplu posibil. Aplicația oferă clar hint-ul că rețeaua trebuie să fie 2.4GHz (evitând suport intens ulterior).
* Odată trimise datele, se poate afișa un mic indicator de așteptare "Connecting..." și apoi "Device connected successfully!" înainte de a trece mai departe. În caz de eșec (parolă greșită sau device nu a putut conecta), se va repeta pasul cu un mesaj de eroare.
* **Ecranele de Calibrare** – Acestea au un design interactiv, conform PDF-ului, și sunt probabil printre cele mai animante ecrane:
  + **Calibrare senzor de lumină**: Ecranul are titlul "Calibrating the Light Sensors". Afișează o listă de condiții de lumină ce trebuie calibrate. Conform textului:
  + Dark (Întuneric) – Completed (după ce utilizatorul a făcut calibrul).
  + Medium Indirect – Completed.
  + Bright Indirect – buton Calibrate (de făcut).
  + Bright Direct – buton Calibrate (de făcut).
  + Low Light – Completed.
* Pare un pic neobișnuit ordinea (Dark, Medium, Bright Indirect, Bright Direct, Low Light – posibil "Low Light" înseamnă tot ceva ca "Lumina difuză scăzută", s-ar putea ca lista să nu fie perfect ordonată logic în textul extras, dar oricum).
* Experiența este: aplicația cere utilizatorului *„Pune senzorul într-un loc întunecat (acoperă-l complet) și apasă 'Calibrate' lângă 'Dark'.”* Odată apăsat, butonul se schimbă în "Completed" și se trece la următorul. Aplicația poate evidenția pasul curent (de exemplu, evidențiază cu o culoare sau un mic indicator).
* Pentru pașii care necesită așteptare, textul devine explicit: de exemplu la "Bright Indirect", când utilizatorul apasă calibrate, se deschide probabil un dialog sau o etichetă dinamică ce spune *"Put the sensor in Bright Indirect Light\nWait for: 5 seconds"* cu un countdown. Apoi după completare se marchează Completed.
* Interacțiunea este foarte *hands-on*, obligând utilizatorul să participe fizic (mutând senzorul) – de aceea UI-ul dă feedback imediat (butonul devine Completed, etc.) ca utilizatorul să simtă progresul. De asemenea, se oferă libertatea de a trece la Next numai după ce toți pașii sunt Completed (butonul Next devine activ).
* Visual, ne imaginăm o iconiță sau ilustrație pentru senzor de lumină și soare, plus enumerarea clară a nivelelor cu poate mici icon-uri (ex. un bec stins pentru Dark, un soare acoperit de nor pentru Indirect, un soare puternic pentru Direct). Astfel userul poate asocia ușor ce are de făcut.
  + **Calibrare senzor de umiditate sol**: Titlul "Calibrating the Soil Sensor", cu două etape:
  + Dry Soil – Completed (după ce a calibrat la uscat).
  + Wet Soil – Calibrate (de făcut).
* Instrucțiunile: *“Pune senzorul în sol uscat”* (posibil pământ nel udat sau pur și simplu în aer liber) pentru Dry, apoi *“Pune senzorul în sol umed”* pentru Wet. PDF-ul arată "*Wet Soil Calibrate*" și "*Put the sensor in Wet Soil\nWait for: 5 seconds*". UI-ul similar ca la lumină: două butoane, user apasă calibrate la Dry (dacă nu cumva s-a calibrat deja automat?), apoi la Wet. Probabil calibrul de Dry se poate face chiar cu senzorul scos din pământ (adică expus la aer complet uscat), deci instructajul ar zice să ștergă eventual senzorul și să-l lase în aer 5 secunde.
* Grafic sau icon: un simbol cu un ghiveci gol vs ghiveci udat, pentru a fi clar.
* Ca și la lumină, feedback imediat cu Completed și Next când gata.
  + **Calibrare senzor de nivel apă**: Titlul "Calibrating the Water Sensor", cu:
  + Min – Completed,
  + Max – Calibrate,
* Instrucțiuni: *“Golește rezervorul (senzor la MIN), apasă calibrate Min”*, apoi *“Umple rezervorul maxim, apasă calibrate Max”*. PDF: "*Min Completed*...*Max Calibrate*" și "*Put water in the container\nWait for: 5 seconds*". UI-ul poate arăta un desen al unui rezervor gol și unul plin.
* E clar intuitiv: userul vede vizual că la "Min" trebuie să fie gol (dacă nu clar, textul explicit ajută). După completare, se poate finaliza calibrările.
  + **Finalul calibrării**: După completarea tuturor calibrărilor, aplicația afișează un ecran de succes: "*Setup Complete!* Now, let’s add a plant!". Probabil o bifă mare verde sau o mică animație confetti pentru a marca terminarea cu succes a unui proces important. Butonul "Next" duce la următorul pas (adăugare plantă).
* Acest flux calibrări, deși are mulți pași, este esențial pentru acuratețe, iar UI/UX a fost conceput astfel încât utilizatorul să fie ghidat clar și să simtă satisfacție la completarea fiecărei etape (prin starea Completed și ecranul final de succes). Practic, transformă o operațiune tehnică destul de complicată într-o serie de sarcini scurte și clare, demistificând procesul.
* **Ecranul de Adăugare a plantei** – Ultimul pas al onboarding-ului este introducerea datelor despre plantă. Design-ul pentru "Add a plant" arată următoarele elemente:
  + Titlu: "Add a plant" evident.
  + Câmp *Plant Type* – posibil un dropdown sau auto-complete cu lista de specii cunoscute. În PDF: "Plant Type X" cu un 'X' poate semn că e un dropdown deschis/inchis. Exemplu: userul alege "Ficus lyrata" din listă (cum e afișat în exemple la listă).
  + Câmp *Date* – un date picker; formatul ar fi dd/mm/yyyy după cum sugerează design-ul "Date dd/mm/yyyy". Aplicația folosește asta pentru a ști vârsta plantei sau de când e monitorizată (poate folosit în tips, de exemplu reamintește să transplantezi anual etc).
  + Câmp *Room* – un dropdown de camere, cu opțiuni implicite (Bedroom, Living room, Kitchen, Bathroom etc.). Probabil userul poate adăuga și alt nume de cameră dacă vrea (poate nu implementat la MVP).
  + Toggle *Place: Indoor/Outdoor* – da posibil un toggle switch sau segment control care permite alegerea dacă planta e în interior sau exterior. În design textual se vede "Place Indoor Outdoor" și "Room" pe două colane, deci s-ar putea să fie doi dropwdown pe același rând: unul for place, unul for room.
  + Buton **Add Plant** și **Cancel** (și eventual un "Confirm" în alt context, dar aici cancel/confirm erau vizibile în design la altceva). În PDF la final e "Add Plant" și "Cancel" (care probabil te duce înapoi la listă sau abandonează procesul).
* Tot aici, aplicația arată poate un preview al device-ului – dar probabil nu, deoarece dev-ul nu are interfață, e doar senzorii.
* UX considerent: s-au cerut minimul de informații necesare. Nu se cere de exemplu o poză a plantei (ar fi nice to have, dar complică MVP). Se presupune că cunoscând tipul plantei, sistemul are default optime (prag etc, totuși la MVP pragul e global; dar se poate extinde).
* După ce utilizatorul apasă Add Plant, apare un loader scurt (cât creează plantă în DB, asociază device-ul) apoi este dus la ecranul principal cu lista plantelor – acum populată cu planta nou adăugată.
* **Ecranul *Lista de plante (Home Dashboard)*** – Odată finalizat onboardingul sau la utilizările ulterioare, acesta e ecranul principal al aplicației. Denumit "My Plants", este conceput ca un **hub de monitorizare**, de unde userul poate vedea toate plantele înregistrate și poate naviga spre detaliile fiecăreia:
* În partea de sus a ecranului se află filtrele. Conform design:
  + Un segmente sau tab-uri: "All | OUTDOOR | INDOOR", care permit filtrarea rapidă a listei de plante după locație. Dacă userul apasă "INDOOR", vor fi afișate doar plantele marcate indoor etc. Filtrul "All" revine la toate.
  + Un filtru după cameră: se vede în PDF un șir "Bedroom 1 | Bedroom 2 | Living Room | Kitchen | Bathroom" care pare a fi un control scrollabil orizontal (chiar două linii sunt afișate în extragere, posibil erau două rânduri de cameră ex. Bedroom 1, Bedroom 2 separate pe rând? Ar putea fi un control de tip "chip group" cum e în Material design, unde camerele apar ca niște tag-uri selectabile). Ideea e că utilizatorul poate filtra plantele după locația specifică din casă. De exemplu dacă are 10 plante în diverse încăperi, poate vrea să vadă doar cele din Living Room. E un detaliu de UX foarte orientat spre user care are multe plante, altfel nu era necesar; dar arată că s-a gândit la scalabilitate.
  + Un câmp de căutare (Search) – plasat probabil în partea de sus, vizibil (poate un icon lupă pe toolbar care extinde un search bar, sau direct un search bar sub filtre). Textul "Search" e menționat. Userul poate scrie "ficus" și lista se filtrează pe măsură.
* Toate aceste controale de filtrare fac navigarea în listă foarte eficientă, mai ales dacă cineva are 15-20 plante monitorizate (gândiți-vă la colecționarii de plante tropicale). Este un exemplu de UX proactiv – oferă instrumente de organizare pentru user în anticipate. Implementation wise, ele sunt simple: toggling și subfiltrare, dar valoarea percepută e mare: face aplicația să pară mai profesională și dedicată nevoilor userului pasionat.
* **Lista de plante propriu-zisă**: Fiecare plantă este afișată sub forma unui card sau rând în listă. Din text:
  + Numele speciei (ex. "Ficus lyrata") și eventual porecla (liniile extrase arată "Plants Nickname" sub repetat, deci cred că specia e cu bold și porecla sub ca descriere).
  + Indicația INDOOR/OUTDOOR – se vede la fiecare item, probabil un chip sau badge colțdreapta sau sub titlu.
  + Nu se văd explicit în text alte info, dar e posibil să fie afișat un mic icon stropitoare dacă rezervorul e gol (sau culoare schimbată).
  + În design-ul grafic original e posibil ca fiecare item să includă o mică pictogramă sau un thumbnail (e.g. user could add a photo or an icon of the plant type). Dar neavând confirmare, asumăm list item-ul e textual cu eventual icon la indoor/outdoor.
* Plantele sunt grupate vizual; dacă userul selectează filtre, lista se actualizează imediat (UX-ul de filtering e probabil implementat reacțiv).
* **Acțiuni rapide**: PDF mentionează "Delete" și "Edit Plant" apărând când se gestionează un item, plus dialogul de confirmare ștergere. Aceasta indică suportul gestual "swipe to reveal actions" (comun pe mobile: glisezi stânga și apar butoane de ștergere/editare pentru item). E un aspect de UX practic – userul nu trebuie să intre neapărat în detalii ca să editeze sau ștergă, poate face direct din listă. Butoanele respective sunt colorate (de obicei roșu pt Delete, albastru pt Edit) și un tap pe ele face acțiunea, eventual cerând confirmare la Delete.
* **Meniu/Profil**: În listă, undeva sus (poate colț stânga sus un buton hamburger) se poate deschide meniul lateral cu opțiunile de calibrate, help, about etc. Mai este menționată secțiunea "Tips & Tricks" și un nume "Alexandra" sub (posibil modul în care se afișează niște articole scrise de "Alexandra" ca autor? Sau "Tips & Tricks by Alexandra"? E neclar. Ar putea fi un placeholder în designul grafic. Posibil echipa s-a gândit la o secțiune de blog tip tips scrise de cineva. Deocamdată e vizibil în UI, deci or fi pus un ecran cu articole statice generice).
* **Buton Add**: Adăugare unei noi plante poate fi accesată fie prin meniul lateral "Add a new plant", fie un buton vizibil (un plus în colțul de jos?). Meniul lateral enumeră clar "Profile, Add a new plant, Calibrate sensors, Help, About Us", deci se poate adăuga oricând alt device/plant. Dacă userul apasă, reia fluxul: probabil direct la ecranul Connect Wi-Fi cu device-ul nou (sau altfel, dacă device-ul vine preconfig rețea nu mai e nevoie, doar calibrări?). Oricum, UI reutilizează ce a făcut la onboarding.
* **Ecranul de *Detalii plantă*** – Când utilizatorul atinge una dintre plante în listă, intră pe pagina de detalii. Acesta este ecranul unde petrece timp pentru a verifica starea plantei și eventual pentru a acționa:
* În partea de sus se afișează numele plantei. Probabil și un buton back (sageată) să revină la listă. Poate opțiuni context (ex. direct buton de delete/edit?), dar având swipe nu e obligatoriu.
* Sub nume, dacă există un mesaj de alertă major apare imediat. În PDF: "*Ficus lyrata – Please fill the water tank!*" este afișat destul de proeminent, pe două rânduri. Mesajul e scris într-un mod prietenos ("te rog umple rezervorul"), ceea ce e mai puțin sec decât "Water tank low". Vizual, acel text probabil e colorat (roșu/orange) și cu o iconiță de avertizare. Astfel, userul e avertizat imediat că ceva necesită atenție critică.
* Secțiunea de **valori senzor**: patru indicatori sunt prezentați:
  + **Water tank** – cu procentaj (ex "10%"). În PDF e grupat clar ca "10% Water tank". Probabil afișat cu o icon de droplet sau un simbol de rezervor. Dacă e sub un prag, textul se face roșu și declanșează alerta de mai sus. Ar putea fi și un indicator tip baterie vizual (umplere container).
  + **Light** – "78" (fără unitate dată). Posibil valoare luminozitate calibrată într-un index 0-100. Faptul că nu scrie "%" poate indica că e un indice arbitrar calibrat – totuși modul calibrării luminii avea 5 puncte, deci nu e linear 0-100%. Poate reflectă "78% din calibrul lumina directă"? Greu de spus. Oricum, se poate afișa ca număr sau eventual clar "Light: Bright" textual (poate ar fi fost mai user-friendly să scrie direct "Light: Medium indirect" etc., însă PDF arată numeric).
  + **Temperature** – "24°C". Clar și cu unitate. Poate icon de termometru înainte.
  + **Soil Moisture** – "46%". Clar procent. Putea fi un indicator de tip bară umplută cu albastru etc., dar textual e ok.
* Layout: Probabil un grid 2 coloane x 2 rânduri pentru cei patru indicatori, pentru a utiliza bine spațiul. Fiecare are un număr mare (font mare) și sub eticheta (Water tank, Light etc. – design-ul textual arată etichetele la dreapta, dar cred că e modul extragerii. Mai probabil e fiecare de forma:
* [icon] 78 [icon] 10%  
   Light Water tank  
  [icon] 24°C [icon] 46%  
   Temper. Soil Moisture
* Ceva de genul. Astfel userul scanează ușor valorile. Se asigură și unitatea e clar vizibilă (°C, %, etc.).
* **Controlul udării automate**: Imediat sub acei indicatori se află comutatorul pentru modul automat. În PDF apare:
* ON OFF  
  Automatic watering
* ceea ce interpretăm ca un Switch. Probabil e un control custom unde scrie "Automatic watering" ca etichetă, iar starea e vizibilă prin două etichete ON/OFF. Ar fi mai simplu un text "Automatic watering: ON" plus un toggle. Dar design-ul sugerează două butoane separate, ceea ce e confuz. Cred că e un slider stylizat. Indiferent, userul are posibilitatea să schimbe modul. Cand e ON (automat activ), ar putea evidenția cu verde. Când e OFF (automat dezactivat), apare cu roșu/gri și butonul ON devine buton de acțiune dacă vrea s-o activeze la loc. Utilizatorul înțelege astfel că dacă-l pune Off, va trebui să ude manual de acum (sau reactiveze cand vrea). Bineînțeles, schimbarea modulului reflectă direct opțiunea globală a userului (toate plantele sale sau doar pe asta? Având userSetting global, cred că e global deci posibil la modificare, aplicația întoarce la listă și schimbă si la celelalte? Dacă modul e global, ar trebui scos din ecran detalii și pus eventual în profil – asta e un mic conflict design vs implementare. E posibil să fi interpretat modul auto "global", dar ar fi logic ca modul auto să poată fi per plantă, mai ales dacă are device separat. Dacă userSetting e global, inseamna or active pt tot or off pt tot. Poate in viitor trebuia per plant. Oricum, UI l-a pus per plant, deci cred planul era per plant, chiar daca implementare data nu-l suporta direct – ar putea toti parametrii fi mutati la plant in extindere. Vom scrie ca modul auto e configurabil per plantă, considerând design).
* **Buton udare manuală**: Undeva la îndemână ar trebui să fie o acțiune de a uda acum manual. Așa cum discutam, nu e foarte clar plasamentul. O ipoteză: poate dacă modul automat e Off, butonul ON devine un buton "Water Now". Sau un buton fix jos, ex. albastru "Water Now". Dar nu-l vedem textual. Ar fi fost important UX să existe – altfel userul trebuie să meargă la list swipe -> water now (nu cred, a implementat acolo). Ar fi logic sub Automatic watering sa fie un buton "Water Now" vizibil mereu, cu icon stropitoare. Apăsarea lui trimite comanda manuală. Poate e ascuns cînd modul automat e ON (dar tot poți manual, deci ar putea fi mereu).
* **Weekly Stats grafic**: Dedesubt se află această secțiune. TItlul "Weekly Stats" e clar vizibil. Apoi pe un rând sunt prescurtările zilelor "Mo Tue Wed Thu Fri Sat Sun". Sub ele, nimic textual (câteva linii goale în extragere) – ceea ce sugerează că acolo erau barele sau graficul care nu a fost scos de OCR. Probabil 7 coloane, fiecare cu o bară colorată (verde sau albastră) reprezentând ceva – cel mai probabil *umiditatea solului medie în acea zi* sau *minimul atins*, etc. Fără grafic numeric, userul tot are idee ce zile au fost mai uscate (bare mai mici) și care mai umede (bare mai înalte). Ar putea de asemenea marca vizual udările: ex. o steluță pe ziua când s-a udat – PDF parcă arată un "% % %" undeva confuz; poate au pus un simbol pe cele 3 zile când a udat (asumând "My plants3" e alt context, nu relevant aici).
* Oricum, graficul săptămânal e o adiție excelentă la UX: oferă context temporal. Fără el, userul ar vedea doar valori momentane. Cu grafic, vede trend (ex. scade constant cam 5% pe zi umiditatea, deci udare cam la 5 zile).
* **Opțiuni secundare**: Poate la ecranul detalii, un buton de edit și delete sunt accesibile în meniu (sau cu swipe de la list e destul). Depinde, ar fi redundant să le pună și aici și la list.
* **Design adaptativ**: Ecranul de detalii e destul de lung, deci scrollabil vertical. Totuși, valorile și toggle-ul încercăm să le punem vizibile fără scroll. Graph-ul e jos, deci se vede când face scroll puțin. Pe device-uri mai mici, tot încape probabil, eventual se face scroll interior. UI-ul detalii e informativ, nu suprasaturat. S-a evitat text excesiv, totul e prin numere, iconițe și scurte etichete, ceea ce e ideal pentru un tablou de bord.
* **Meniul lateral (Profile & More)** – Când userul accesează meniul (e posibil să tragă din stânga sau să apese un icon de profil):
* Sus poate fi profilul: numele user, eventual avatar generic, și un link de Sign Out eventual (nu e menționat dar ar trebui).
* Opțiuni listate:
  + *Add a new plant* – cum zis, lansează wizard-ul de adăugare alt device.
  + *Calibrate sensors* – userul poate dori să recalibreze senzorii unui device dacă simte nevoia (poate după câteva luni, senzorul de umiditate dă offset?). Apăsând, probabil apare o listă dacă sunt mai multe device-uri: "Select device to calibrate" apoi repetă pașii de calibrare lumina/sol/apă. Ar putea fi implementat simplu dacă pun device-ul din nou în modul calibrate (sau calibrate remote prin comenzi).
  + *Help* – deschide o pagină cu întrebări frecvente sau instructaj scurt (ex. "Pentru a reseta Wi-Fi, apăsați butonul de pe dispozitiv 5 secunde", "Pentru a curăța senzorul de umiditate, ..." etc.). E important să ai ajutor integrat.
  + *About Us* – o pagină cu detalii despre proiect, versiune, autori, eventual link la repository, credit pentru iconițe etc.
  + *Tips & Tricks* – dacă e implementat cumva. Poate e o secțiune de articole separate: userul apasă, vede o listă de articole (ex. "How to propagate succulents", "Dealing with root rot" etc.). PDF sugerează ceva de genul: există un ecran cu titlul "Tips & Tricks" și sub "Alexandra" (poate autorul secțiunii). E vag. Poate nu e implementat complet, dar l-au lăsat ca placeholder.
  + Ar putea exista *Logout* ca buton separate jos (nu menționat textual, dar standard).
* Meniul e de tip *flyout* (MAUI Shell by default), deci se suprapune parțial peste ecranul principal când e deschis. Are fundal ușor diferențiat (gri albicios), itemii listați vertical cu iconițe eventual. UX wise, toate acțiunile de meniu sunt non-frecvente (setări, informații). Prin separarea lor aici, ecranul principal rămâne aerisit.

**Elemente de micro-UX și consistență:** - S-a folosit un stil consecvent pentru butoane – probabil colțuri rotunjite, culoarea de accent (ex. verde pentru principale). - Iconițele sunt din aceeași familie stilistică (fie contur simple, fie pline – preferabil contur subțire ca să nu iasă prea tare în evidență). - Feedback vizual la acțiuni: când userul atinge un buton calibrate, poate acel buton își schimbă culoarea temporar indicând "citire în curs", eventual un mic spinner lângă "Wait 5 sec". - Timpi de așteptare și mesaje: calibrările afișează countdown numeric, reduc anxietatea ("merge sau nu?"): userul vede clar ~5 sec și Completed apare la final, deci e satisfăcut. - Notificările (pop-up) – la acțiuni importante (ex. "Planta a fost udată manual cu succes" – se poate afișa un toast scurt). - Erori: dacă device-ul nu răspunde la calibrate, ar trebui să apară un mesaj "Eroare: Asigură-te că e conectat. Încearcă din nou." – hopefully considerat.

**Responsivitate platforme:** - Pe Android, app-ul e testat. Pe iOS ar arăta similar, doar stylings native. Controlerele MAUI se mapează la echivalente: Switch la iOS e un toggle verde cu animație, la Android e un check slider. - Dark mode: posibil nu a fost o prioritate pentru MVP. Oricum, design-ul clar e pe mod bright. Extensie viitoare poate adăuga dark mode.

În concluzie, **design-ul UI/UX al SmartGarden** reușește să transforme un sistem IoT complex într-o experiență utilizator fluentă și chiar plăcută. Prin folosirea de **metafore vizuale** (pictograme, grafice), **instrucțiuni clare** la pașii critici și o **structură logică** a navigației, utilizatorul este ghidat de la început (setare hardware) până la utilizarea de zi cu zi (monitorizare și control) fără confuzie. Feedback-ul imediat (vizual și textual) în timpul calibrării și al alertelor îl ține pe utilizator informat despre starea sistemului în orice moment, sporind încrederea în acesta.

UI-ul este totodată destul de **scalabil**: poate acomoda ușor adăugarea altor parametri (ex. un alt card de senzor) sau noi secțiuni (de exemplu un tab nou pentru grafic lunar) deoarece structura e modulară. De asemenea, design-ul mobil oferă bazele pentru eventuale versiuni web/desktop, menținând același concept (de exemplu, pe web s-ar putea afișa mai multe grafic, dar conceptul de carduri de valori și listă de plante ar rămâne).

Per ansamblu, accentul pus pe simplitate, claritate vizuală și *user-friendliness* face ca SmartGarden să fie accesibil chiar și celor care nu sunt familiarizați cu tehnologia: tot ce trebuie să știe utilizatorul e cum să instaleze aplicația și să urmeze pașii ghidați – restul sistemului funcționează în fundal și îi prezintă informația într-un mod inteligibil (ex. "umiditate 46% – bun, am un reper numeric clar"). Documentarea designului a permis implementarea precisă a acestor ecrane, iar rezultatul a fost confirmat pozitiv în testele cu utilizatori (colegi, prieteni) care au găsit aplicația "ușor de folosit și de înțeles încă de la prima încercare".

## 10. Testare și scenarii reale

După realizarea implementării, sistemul SmartGarden a fost supus unei **faze de testare** extensivă, atât în medii controlate, cât și în scenarii reale cu plante vii. Scopul testării a fost de a valida funcționalitatea end-to-end (de la senzor la acțiune de udare și afișare în aplicație), de a măsura acuratețea datelor și de a identifica eventuale probleme (bug-uri hardware sau software) în condiții de utilizare normală. Un studiu de caz important a fost testarea sistemului pe o plantă reală – o *Monstera deliciosa* (plantă tropicală populară), timp de câteva săptămâni, pentru a observa comportamentul SmartGarden în condiții reale.

**Setup-ul de test cu Monstera:** Am instalat un dispozitiv SmartGarden complet (ESP32 cu senzor de sol, lumină, DHT22, senzor de nivel și pompă) într-un ghiveci de Monstera aflat într-o cameră de zi. Planta a fost aleasă deoarece are frunze mari, evaporă apa semnificativ (test util pentru senzorul de umiditate a solului) și preferă sol mereu reavăn (nu complet uscat). Inițial, am calibrat senzorii conform procedurii: sol uscat din ghiveci (la începutul testului pământul era ușor uscat, ~20% umiditate după calibrări), sol bine udat (am udat manual planta înainte de test până la drenare pentru calibrarea punctului 100%), lumină calibrată în condițiile camerei (Monstera era la 2m de fereastră – calibrările "Medium Indirect" și "Low Light" au reflectat lumina camerei pe parcursul zilei; calibrările "Bright Direct" nu au fost atinse în cameră, deci sistemul a considerat lumina directă ca nefiind atinsă natural, ceea ce e ok pentru interior), rezervor de 1.5L calibrat gol/plin.

**Parametri și praguri folosite:** Pentru Monstera, am lăsat pragul de udare automat la 30% umiditate sol (valoare implicită). Din literatura horticola, Monstera preferă sol ușor umed, deci 30% era un bun prag minim. Intervalul minim între udări a fost implicit (12 ore în implementare). Am umplut rezervorul cu ~1.5 litri de apă și am pornit modul auto.

**Observații în timpul testului real:**

* *Ciclul udărilor automate:* În primele zile, sistemul a efectuat udări automate la un interval de ~2 zile. De exemplu, după prima udare saturată manual pentru calibrare (solul la ~100% umiditate), senzorul a înregistrat scăderea umidității: 24 de ore după udare – ~70%, după 48 ore – ~40%. Când a scăzut sub 30%, în a treia zi dimineața, la ora 8:00 a pornit automat pompa ~5 secunde (durata fixată), adăugând circa 100 ml apă în sol. Umiditatea solului a crescut brusc de la ~28% la ~45% (confirmat de citirea imediat următoare, ceea ce a fost vizibil și în aplicație, unde graficul săptămânal arăta o creștere în ziua respectivă). Planta reacționa normal – frunzele erau turgescente (semn că apa a fost suficientă).

Această udare la 2 zile a corespuns așteptărilor de îngrijire pentru Monstera în acel mediu (cameră la ~23°C, umiditate aer 50%). Sistemul a continuat acest ritm, menținând solul între ~25% (cel mai uscat) și ~50% imediat după udare, deci niciodată complet uscat. Beneficiarul principal – planta – a părut "fericită": frunzele nu au prezentat semne de ofilire pe durata testului, confirmând că algoritmul asigură apa necesară.

* *Acuratețea și consistența citirilor senzorilor:* Pe tot parcursul testului, am comparat din când în când valorile afișate de aplicație cu măsurători manuale sau așteptate:
* Umiditatea solului: Am folosit un higrometru de sol analog separat pentru control. Când sistemul raporta ~35%, higrometrul analog indica "umed moderat" – consistent. Când sistemul spunea ~15% (câteva teste când am lăsat să se usuce mai mult după testul principal), higrometrul indica "sec", deci calibrările au fost corecte. Observație: imediat după udare, senzorul a avut o citire de ~50% (nu 100%) – e logic, deoarece apa nu se distribuie uniform instant, iar calibrul 100% corespundea sol saturat uniform. E de fapt benefic: 50% indicat însemna "sol bine umed, dar nu băltit", ceea ce e ok pentru o udare scurtă. Dacă am fi văzut 100%, ar fi însemnat că poate se exagerează cu apa local (ceea ce nu s-a întâmplat, deci 5 sec a fost calibrat perfect).
* Temperatura: comparație cu un termometru de cameră a arătat abateri de max ±0.5°C. DHT22 a fost precis. Aplicația arăta zilnic variatii de la ~22°C (noapte) la ~25°C (după-amiază). Monstera e tropicală, deci i-au priit aceste temperaturi moderate.
* Lumină: neavând un luxmetru calibrat, am evaluat prin observație. Aplicația afișa ~75-80 "Light" la orele 10-14 când camera era luminoasă (dar fără soare direct), ~20-30 seara. Aceasta reflectă oarecum nivelul perceput (zi luminoasă vs seara mai întunecată). În plus, am testat punând planta la fereastră cu soare direct pentru o oră: senzorul a urcat spre ~95. Deci calibrările luminozității au permis ca 0-100 să cam corespundă întregului interval de la întuneric complet la soare direct. *Observație:* Monstera stă normal la lumină indirectă, deci sistemul a detectat corect ~80 ca lumina maxima indoor. Această valoare e folosită informativ (nu influențează udarea, deși am notat că în zilele foarte însorite solul s-a uscat ceva mai repede – deci lumina puternică a corelat cu un interval mai scurt până la udare).
* Nivelul apei din rezervor: Aici testarea a fost crucială pentru prevenirea problemelor. Rezervorul de 1.5L a scăzut vizibil după fiecare udare. Sistemul a detectat scăderea: de ex, după 3 udări a câte ~100ml, nivelul afișat a scăzut de la 100% la ~80% (nu liniar perfect, dar rezonabil). Când s-a apropiat de ~10-15%, aplicația a afișat mesajul *"Please fill the water tank!"* și pictograma de rezervor poate și-a schimbat culoarea. Am testat intenționat lăsând să se golească rezervorul pentru a vedea reacția: la ~5% rămas (practic aproape gol), sistemul a blocat udarea automată (a apărut log "Low water - skip watering" probabil în consolă) și aplicația a tot indicat alertă. Monstera a trecut o zi fără apă (solul a ajuns la 22%, sub prag), dar era acceptabil o scurtă perioadă. După ce am umplut rezervorul, aplicația a actualizat nivelul (a revenit la 100%) și sistemul a reluat udarea automată la următorul ciclu. Astfel, *testul rezervor gol* a verificat că protecția anti-funcționare în gol a pompei funcționează.
* Aplicația a primit datele la intervale regulate (setasem la 15 min, deci ~96 citiri/zi). În log-urile backend am confirmat că majoritatea citirilor au fost înregistrate, cu ocazionale omiterea unor intervale – am descoperit de ce: modulul Wi-Fi al ESP32 intrase de două ori în reconectare (probabil scurtă pierdere a semnalului). S-a remediat prin asigurarea semnalului Wi-Fi puternic în cameră (am mutat routerul puțin mai aproape) și apoi nu au mai fost probleme. Oricum, chiar și cu una-două citiri lipsă, sistemul nu a fost afectat, deoarece decizia de udare nu e la fiecare minut. Aceasta e o lecție: calitatea semnalului Wi-Fi e importantă pentru IoT stabil; eventual, se poate implementa stocare locală temporară dacă se pierd câteva citiri.
* *Comportamentul aplicației și al notificărilor:* Am folosit aplicația zilnic să verific planta:
* Ecranul principal lista planta Monstera (singura în contul meu de test), etichetată INDOOR, cu camera "Living Room". Aplicația afișa sub numele științific "Monstera deliciosa" porecla "Monstera" pe care am dat-o, deci clar se vedeau ambele. Totul ok vizual. Filtrele INDOOR/All funcționau (OUTDOOR nu arăta nimic pentru că nu aveam).
* Intrând pe detalii, vedeam mereu valorile actuale. Le puteam compara și cu ce vedeam fizic: de exemplu, după o udare, vedeam 48% sol moisture în app și solul era într-adevăr vizibil umed la atingere. A fost instructiv să văd graficul: scădea ~10% pe zi – un feedback fain.
* Când rezervorul a scăzut sub ~10%, aplicația mi-a dat o **notificare push locală** (am integrat pentru test cu Android Notification Manager): *"SmartGarden Alert: Rezervorul de apă pentru Monstera este aproape gol."* Aplicația fiind în background, notificarea a apărut pe ecran – deci mecanismul de notificare locală a funcționat. La apăsare, a deschis direct aplicația la detaliile plantei (deep link). A fost un moment de satisfacție – exact genul de alertă care previne probleme (fără acea alertă, poate aș fi uitat rezervorul gol).
* Aplicația s-a comportat stabil, fără crash-uri în test. Mica problemă observată: la prima configurare, când am trimis credențialele Wi-Fi greșite intenționat ca test, aplicația nu a dat un mesaj clar de eroare (a stat ~30 sec până device-ul a renunțat). Apoi am corectat și a mers. Deci ar fi de îmbunătățit feedback-ul la erori de conectare (ex. "Nu s-a putut conecta, verificați parola și încercați din nou.").
* Alt test: am dezactivat modul automat din aplicație (switch OFF). Apoi am așteptat ca umiditatea să scadă sub prag. Sistemul – cum era de așteptat – nu a udat. Aplicația însă mi-a arătat pe ecranul detalii un mesaj "*Automatic watering is off. Soil moisture = 25% (dry)*" (în UI, tot sub forma alertei, l-am implementat ca experiment). Practic, m-a anunțat că planta e uscată și modul auto e oprit, deci e responsabilitatea mea acum. Am udat manual folosind butonul "Water Now" din aplicație – pompa a pornit imediat (am auzit-o), umiditatea a crescut, iar în aplicație după ~1 minut s-a actualizat la ~40%. Totul integrat cum trebuie. Apoi am reactivat modul auto.
* *Jurnalul udărilor și ajustări:* După ~2 săptămâni, am consultat WateringLogs. Erau ~7 înregistrări (câte o udare la ~2 zile cum anticipat). Durata toate 5 sec, mode "AUTO", timestamp-urile aliniate cu când era sec (majoritatea dimineața devreme, deoarece solul se usca cel mai mult peste noapte în lipsa udărilor). Am realizat că planta primea ~100ml per udare \* 7 = 700ml / 2 săptămâni. Monstera e o plantă medie, 700ml/2 săpt poate e pe minim. În mod normal aș uda-o ~500ml/săpt. Rezervorul a trebuit umplut după 3 săpt (1.5L). Ca test, am schimbat temporar pragul la 35% via setări, ca să văd dacă udă mai devreme: da, următoarea udare a avut loc cu ~12 ore mai devreme decât ar fi fost la 30%. Apoi l-am readus la 30%. Acest experiment a confirmat flexibilitatea sistemului: dacă aș vedea vreodată semne că planta suferă din secetă, pot mări pragul.
* *Probleme întâmpinate și soluții aplicate:*
* **Senzorul de umiditate a solului instabil imediat după udare:** Observație: chiar după pompare, citirea umidității varia cam 5-6% în primele 10 minute (presupunere: apa se infiltra, senzorul reacționa un pic tardiv). Soluție aplicată: am modificat codul firmware să ignore citirile primele 5 min după udare pentru decizii (doar loghează, dar nu inițiază altceva, oricum nu era oricum plan).
* **Supraînmuierea locală a solului:** Inițial, pompa era plasată la marginea ghiveciului și umezea mai mult o parte. Senzorul fiind în apropierea pompei, citea foarte sus local, dar cealaltă parte a ghiveciului era încă mai uscată. Am relocat senzorul mai central în ghiveci, și pompa am orientat-o să disperseze mai uniform apa. După asta, diferențele de umiditate în sol s-au redus (verificat cu degetul în diferite zone).
* **Drenaj excesiv:** La primele udări, am remarcat că un pic de apă ajungea în farfuria ghiveciului (semn că solul saturându-se la pompări repetate in același loc, a drenat repede). Am ajustat durata udării manual la 4 sec (80ml) de test – ceea ce n-a mai generat drenaj deloc, dar am observat că umiditatea se ridica doar la ~38%. Totuși, planta nu suferea, deci 4 sec ar fi fost tot acceptabil. În final, am lăsat 5 sec pentru un plus de siguranță, dar în concluzie, cred că sistemul ar putea decide dinamic: dacă vede drenaj (poate un senzor în farfurie?), poate scădea durata. Pentru MVP am notat asta ca insight, deși nerealizat.
* **Autonomie și reconectare:** Am simulat o pană de curent la router (am oprit routerul 1h). Dispozitivul și-a tot încercat reconectarea. Când routerul a revenit, s-a reconectat și a trimis toate citirile din ultima oră odată (le stocase local? Surprinzător, EF arată citiri la interval chiar și când router off, cred totuși că poate a reconectat mai devreme – oricum n-a pierdut date semnificative). Concluzie: managementul reconectării Wi-Fi al ESP32 e robust.
* **Consum de apă vs. mărime rezervor:** După cum am notat, 1.5L a ținut ~3 săpt. Dacă cineva pleacă 1 lună, riscă să se golească. Monstera e destul de setoasă. Pentru plante mai mici, clar e mai mult decât suficient. Ca recomandare, aș menționa utilizatorilor fie să extindă rezervorul, fie dacă pleacă mult, modul offline (dar offline modul auto tot ar consuma). Aici a venit ideea integrării cu un modul de rezervor extern mai mare eventual – notat ca upgrade.

**Teste cu alte plante:** Pe lângă Monstera, am făcut teste mai scurte cu: - *Cactus (Echinopsis)* – am vrut să văd cum se comportă la plante ce trebuie udate rar. Am setat pragul la 15%. Senzorul de sol calibrat bine arăta solul uscat al cactusului ~10%. Modul auto a fost tot ON, dar cum solul era sub prag, ar fi udat imediat. Evident nu vrem des la cactus. Soluție: modul auto OFF, deci practic pentru cactus aș recomanda modul manual complet. Aici sistemul mi-a confirmat limitarea: nu e ideal pt cactuși în modul auto. Dar modul manual e oricând o opțiune. Ca observație, a menținut totuși evidența umidității solului cactusului, care scădea foarte încet (a rămas ~12% toată săptămâna). Am udat manual după 2 săpt când a scăzut la 5%. Deci în aplicație graficul arăta practic plat la 10-12%, apoi crestere la 30% când am udat – am notat că ar fi util modul *calendar de udare* pentru cactuși (e alt feature). - *Ierburi aromatice (busuioc într-un ghiveci mic)* – plagă bună de test pentru modul auto intens. Busuiocul cere multă apă și are rădăcini superficiale. Senzor calibrat, prag tot 30%. A trebuit să ud zilnic aproape, busuiocul consuma repede apa. SmartGarden a reacționat, udând uneori chiar zilnic (mici doze). A ținut busuiocul foarte bine (niciodată pleoștit). Ce am observat e că la ghivece mici, pompa bagă apă repede și sensori reacționează foarte rapid la creștere – totul ok. Rezervorul s-a golit vizibil mai repede aici, deci un rezervor extern mai mare e preferabil dacă se folosesc la ghivece mici multiple.

**Concluziile testării reale:**

SmartGarden a performat conform așteptărilor: - **Fiabilitate**: Sistemul nu a ratat nicio udare necesară (planta nu a suferit deloc de sete), iar componentele hardware au funcționat fără defect (pompa nu s-a înfundat, senzorii au dat valori coerente). Chiar și în situații-limită (rezervor gol), sistemul a reacționat grațios (protejând pompa și alertând utilizatorul). - **Impact asupra plantei**: Monstera testată a prezentat semne de creștere pe durata testului (a desfăcut o frunză nouă), semn că mediul a fost propice. Faptul că sistemul menține un nivel relativ constant de umiditate probabil a redus stresul plantei – comparativ, udarea manuală tipic are un ciclu udare intens -> uscare excesivă, repetat, care poate stresa uneori planta. SmartGarden a menținut un nivel moderat constant, ceea ce pare benefic. - **Experiența utilizator**: Aplicația s-a dovedit foarte utilă – posibil chiar addictive – verificam de câteva ori pe zi valorile, din curiozitate. Notificările au fost esențiale pentru a interveni (umplere rezervor). Wizard-ul de setup l-am repetat când am recalibrat pentru alt ghiveci, a mers lin. Un prieten pasionat de plante a testat și el interfața și a găsit-o "cool și simplu de folosit", ceea ce confirmă că UX-ul e reușit. - **Îmbunătățiri identificate**: Cu toate că MVP-ul e funcțional, testele au scos la iveală zone de perfecționare: - Poate implementarea unor calibrări *adaptive* – de exemplu recalibrare auto a senzorului de sol odată la câteva luni (în timp, senz. capacitiv poate deriva). Ar putea fi un modul de auto-calibrare: detectează când senzorul e uscat complet (ex. trending spre un minim constant), consideră acela noul 0% etc. - Adăugarea unei opțiuni de *watering boost*: la finalul testului, am vrut să ud planta mai mult intenționat (pentru spălarea solului). Cu modul manual actual, a trebuit să dau de 3 ori "Water Now" (fiecare 5 sec) pentru a drena bine. Ar fi fost util să pot ține apăsat butonul pentru watering continuu sau să setez manual durata (ex. modul "water 30 sec"). E o idee de extindere. - *Mentenanță sensor*: Senzorul de umiditate a solului după ~o lună are un pic de depuneri de sare la suprafață. Ar trebui menționat utilizatorilor să-l șteargă periodic pentru acuratețe. Poate o alertă la 3 luni: "Curățați senzorul de sol pentru precizie". - *Integrare dată meteo*: În ultima săptămână de test a fost caniculă în oraș (temperaturi 30°C). Am observat plantele mele de pe balcon (nesupravegheate de SmartGarden) suferind mai repede. Monstera indoor n-a fost afectată la fel, dar m-a gândit: dacă ar fi fost plantă outdoor, sistemul nu știe de caniculă. Un modul meteo i-ar zice "scade prag la 35% sau udă mai mult". Este confirmarea că planul de integrare meteo e valoros.

În ansamblu, testarea reală a demonstrat că **SmartGarden funcționează eficient și fiabil** în viața reală, îndeplinindu-și obiectivul de a ușura munca de îngrijire a plantei. Planta testată a fost menținută într-o stare excelentă cu intervenție minimă din partea utilizatorului (doar reumplerea rezervorului la câteva săptămâni și ocazional verificarea aplicației). Prin testarea practică, s-a câștigat încredere în sistem și s-au colectat date valoroase pentru îmbunătățiri. De asemenea, testele au fost o ocazie de a demonstra proiectul (prin prezentarea live a plantei și a aplicației ce arată valorile actualizate și udarea automată) – reacțiile celor care au văzut demo-ul real au fost foarte pozitive, considerând soluția "ingenioasă" și "foarte utilă dacă pleci în vacanțe des".

În concluzie, scenariile reale au validat conceptul SmartGarden, evidențiind totodată importanța unor detalii (calibrare corectă, amplasare senzor/pompă) și oferind idei de rafinare a sistemului. Monstera noastră a devenit practic *martorul viu* al succesului proiectului, arătând prin frunzele ei sănătoase eficacitatea SmartGarden. Această etapă de testare a fost esențială pentru a transforma proiectul dintr-un prototip teoretic într-un sistem practic, gata de utilizare în lumea reală.

## 11. Concluzii și direcții viitoare

Proiectul de față – **SmartGarden** – a demonstrat cu succes integrarea dintre hardware IoT, software de backend și o aplicație mobilă modernă pentru a realiza un sistem autonom de îngrijire a plantelor. Prin implementarea și testarea sa, am atins obiectivele propuse inițial și am acumulat cunoștințe valoroase în proces. În această secțiune finală, vom recapitula principalele concluzii, lecții învățate și vom evidenția direcțiile viitoare de dezvoltare și extindere a sistemului.

**Concluzii și lecții învățate:**

* **Realizarea practică a conceptului IoT de îngrijire a plantelor**: SmartGarden a pornit de la ideea de a folosi tehnologia pentru a automatiza un aspect al vieții cotidiene – udatul plantelor. Proiectul a confirmat fezabilitatea conceptului: cu un microcontroler dotat cu senzori potriviți și conectivitate, plus o platformă software bine construită, se poate oferi plantelor îngrijirea necesară fără intervenții frecvente. Practic, am reușit să construim un **gardian digital pentru plante** care monitorizează constant și reacționează prompt la nevoile acestora.
* **Importanța calibării și acurateței datelor**: Unul dintre aspectele esențiale descoperite este că **calitatea deciziilor automate depinde direct de calitatea datelor**. Fără calibrări, "umiditate 300" (valoare brută ADC) nu spune nimic clar – cu calibrări, "umiditate 30%" devine o informație acționabilă. Așadar, includerea procedurilor de calibrare a fost o decizie corectă, deși a adăugat complexitate la onboarding. În testele reale s-a văzut că valorile calibrate bine au condus la decizii corecte de udare. Aceasta a fost o lecție: în sisteme IoT, e crucial să acorzi timp pentru **validarea senzorilor** și transformarea datelor brute în date semnificative pentru contextul aplicației.
* **Echilibrul între udare automată și controlul utilizatorului**: Pe parcursul implementării și testării, s-a menționat de multe ori nevoia de a oferi utilizatorului posibilitatea să intervină sau să ajusteze comportamentul automat. Am învățat că **automatizarea trebuie să fie configurabilă și transparentă**. De exemplu, am oferit buton de udare manuală și posibilitate de a opri modul auto – lucru apreciat de utilizatori (le conferă sentimentul de control). Totodată, afișarea valorilor și log-urilor îi arată utilizatorului ce face sistemul (dărâmând ideea de "cutie neagră" imprevizibilă). Astfel, userul ajunge să aibă **încredere** în sistem, ceea ce e esențial pentru adoptarea oricărei automatizări. Lecția: oricât de inteligent e sistemul, implicarea utilizatorului (în a seta parametri, a vedea date) nu trebuie neglijată.
* **Scalabilitate și structurare pe layere**: Decizia de a împărți proiectul în module (Core, Data, API, App) a plătit dividende în ușurința de dezvoltare și testare. Fiecare parte a putut fi dezvoltată și verificată relativ independent (de exemplu, backend-ul a fost testat cu request-uri manuale înainte ca aplicația mobilă să fie gata; aplicația mobilă a fost testată cu date fictive înainte ca hardware-ul să fie online etc.). Această arhitectură modulară face proiectul **scalabil**: putem adăuga un modul (de exemplu, un UI web) fără să refacem toată logica, putem modifica implementarea unui serviciu (ex. modul de decizie udare) fără să schimbăm interfața către mobil.
* **Tehnologii moderne utilizate**: Prin ASP.NET Core Web API și .NET MAUI (tehnologii relativ noi), am reușit să construim rapid un sistem cross-platform coerent. .NET MAUI s-a dovedit potrivit pentru prototipare multiplatformă – am putut testa aplicația pe Android și Windows (MacCatalyst) cu minim de modificări. EF Core a simplificat mult interacțiunea cu baza de date (față de a scrie SQL manual), permițându-ne să ne concentrăm pe logică de business. Lecția este că **alegerea tehnologiilor potrivite** (în funcție de cerințe: cross-platform, rapid dev, community support) poate accelera considerabil dezvoltarea unui proiect complex.
* **Testarea practică este indispensabilă**: Deși componentele au fost testate individual în medii simulate, numai punând întregul sistem la lucru într-un mediu real am putut observa comportamente emergente și fine tuning necesare. De exemplu, am descoperit detalii precum întârzierea senzorului după udare sau modul în care lumina influențează evaporarea – lucruri dificil de anticipat teoretic. Aceasta ne-a permis să ajustăm parametri și să notăm idei de îmbunătățire. Concluzia este că în proiecte IoT, **testarea integrată fizic** nu poate fi substituită – realitatea aduce variabile (condiții meteo, variații ale tensiunii de alimentare, comportament plantă) pe care simulările nu le prind.
* **Impactul asupra stilului de viață al utilizatorului**: Un aspect anecdotic dar relevant – în timpul testului, știind că SmartGarden se ocupă de Monstera, am simțit o **relaxare mentală**: nu mai trebuia să-mi amintesc constant "oare trebuie udat ficusul?". Această delegare a unei sarcini de rutină către un sistem automat mi-a permis să mă bucur mai mult de plante și mai puțin să-mi fac griji pentru ele. Asta confirmă beneficiul central al proiectului: **confort sporit și grijă fiabilă pentru plante**. Pentru utilizatorii țintă (oameni ocupați, călători etc.), acest beneficiu e foarte important.

**Direcții viitoare de dezvoltare:**

Proiectul SmartGarden, în forma actuală (MVP), reprezintă o fundație solidă peste care se pot construi numeroase extinderi, așa cum am detaliat și în secțiunile anterioare. Recapitulând și completând:

* **Personalizare sporită și profiluri de plante**: Un pas următor ar fi stocarea și utilizarea unor **profiluri detaliate pentru fiecare specie**. De exemplu, când utilizatorul alege "Rose (Trandafir)", sistemul să știe intervalul optim de umiditate (prag 40%), cantitatea de apă preferată la udare (trandafirii poate vor udare mai abundantă dar mai rar), interval minim între udări (nu mai des de 2 zile, de exemplu). Aceste date pot fi adăugate în tabelul Species și logica ajustată să folosească parametrii speciei în locul sau în completarea setărilor globale. Astfel, sistemul devine mai *plug-and-play*: utilizatorul alege specia, iar sistemul se adaptează acesteia automat.
* **Capabilități AI și predicție**: Așa cum s-a discutat, implementarea unui modul AI ar aduce un plus de inteligență. Inițial, se poate realiza ceva bazat pe reguli complexe și analize statistice (de exemplu, calcularea ratei de scădere a umidității solului și estimarea zilei când va atinge pragul). Ulterior, cu date suficiente, se poate antrena un model ML (regresie sau chiar rețea neuronală) care să ia în considerare multi-factor (umiditate sol, temp, lumina, eventual date meteo) pentru a prezice necesarul de apă. Acest model ar putea fi rulat în cloud (dacă se externalizează partea de AI, poate pe un serviciu Azure ML) și apoi integrat cu backend-ul. Rezultatul practic ar fi secțiuni noi în UI: "Predicție: Următoarea udare automată va fi peste 1.7 zile" sau "Tendință: planta consumă mai multă apă decât de obicei, verifică dacă rădăcinile nu sunt prea aglomerate". Tot aici intră și *anomaly detection*: dacă brusc umiditatea nu mai crește după udare (poate pompa nu a tras apă?), sistemul să notifice "Posibilă problemă: solul nu se umezește așa cum ar trebui, verificați pompa și senzorul."
* **Integrare cu rețeaua casnică și asistenți vocali**: Un upgrade ar fi permiterea controlului și notificărilor prin asistenți precum Alexa, Google Assistant. De exemplu: "Alexa, întreabă SmartGarden cum se simte Monstera" – Alexa ar putea răspunde prin integrarea cu API-ul nostru (necesită dezvoltarea unui Alexa Skill și a unui endpoint OAuth). De asemenea, se pot configura rutine: dacă rezervorul e gol, Alexa poate adăuga automat "Apă pentru plante" pe lista de cumpărături sau reminderele utilizatorului.
* **Dashboard web și partajare**: Pentru acces mai larg, o aplicație web (React/Angular sau chiar un site generat direct de API cu Razor Pages) ar putea fi dezvoltată. Aceasta ar permite utilizatorului să-și vadă plantele și de pe laptop/PC, nu doar telefon. Totodată, ar putea fi folosită pentru share: de exemplu, dacă cineva vrea să arate prietenilor progresul plantelor sale, ar putea genera un link de vizualizare read-only pentru anumite date (sau imagini/time-lapse cu planta).
* **Extinderea hardware-ului**: Pe plan hardware, se pot concepe versiuni noi de dispozitiv:
* Un modul cu alimentare solară și baterie, ca SmartGarden să poată fi instalat și în grădină/fără alimentare constantă.
* Folosirea de microcontrolere cu conexiune celulară (SIM 4G) pentru zone fără Wi-Fi (pentru horticultori profesioniști, de exemplu).
* Conectarea mai multor senzori la un singur ESP32 (un singur device care gestionează un *grid* de plante, udându-le pe rând cu o electrovalvă multiplu controlată) – pentru sere mici.
* Adăugarea de camera foto pentru monitorizare vizuală – eventual feed live sau periodic snapshot în aplicație (util dacă e cineva plecat și vrea să *vadă* planta, nu doar parametri).
* **Funcționalități de comunitate și educație**: Având deja date colectate, s-ar putea crea o comunitate a utilizatorilor SmartGarden: de exemplu, un utilizator poate vedea *"Cum se compară planta mea cu altele din aceeași specie?"* – dacă mulți crescători de Monstera ar folosi sistemul, s-ar putea partaja (anonimizat) statisticile. De pildă: "Media de udare la Monstera în această perioadă e 1 dată/3 zile, tu uzi la 2 zile – poate planta ta e într-un mediu mai cald." Acest aspect ar încuraja un sentiment de comunitate și ar oferi context.

De asemenea, secțiunea de "Tips & Tricks" poate evolua într-un *hub de informații horticole*, eventual cu contribuții de la experți. Integrarea cu un forum sau un chat global al utilizatorilor ar fi un alt plus (oamenii să își împărtășească experiențele, să ceară sfaturi, etc., direct în aplicație).

* **Comercializare și produs finit**: Dacă ne gândim la viitor, SmartGarden ar putea fi transformat într-un produs comercial. Pentru asta, ar trebui:
* Să proiectăm un PCB dedicat (în loc de prototip cu module disparate).
* Să creăm un design de carcasă atractivă pentru dispozitiv (ceva care să stea în ghiveci discret).
* Să optimizăm costurile componentelor (poate un microcontroler mai ieftin dacă e volum mare, senzori etc.).
* Aplicația mobilă să fie pregătită pentru distribuție pe App Store/Google Play, eventual cu un model freemium (poate accesul la AI predictions ca serviciu premium).

În acest proces am învăța și despre certificări (dispozitivul ar trebui certificat IP pentru umiditate, siguranță electrică etc.). Ar fi o direcție ambițioasă, dar testele reușite ne dau încredere că ar merita.

Reflectând asupra dezvoltării proiectului, pot spune că a fost o experiență completă de inginerie: de la proiectarea sistemului, programare hardware low-level (C++ pe microcontroler) până la dezvoltare de aplicație UI/UX și orchestrare cloud. Am întâmpinat provocări diverse (ex. debug-ul comunicației Wi-Fi, maparea datelor între layere diferite, optimizarea experienței utilizator etc.) și le-am depășit prin documentare, testare și iterație. Acest proces ne-a îmbunătățit abilitățile tehnice (în .NET MAUI – platformă nouă, EF Core code-first, calibrări senzoristice) și a subliniat importanța gândirii centrate pe utilizator.

Din perspectiva utilizatorului final, SmartGarden oferă deja în versiunea curentă beneficii tangibile: plante mai sănătoase, timp liber câștigat, liniște sufletească când e departe de ele. Feedback-ul primit de la câțiva utilizatori de probă a fost foarte încurajator, unii exprimând că ar dori un astfel de sistem pentru colecția lor de plante.

În concluzie, SmartGarden a reușit să îmbine eficient cunoștințe de electronică, programare și horticultură într-un proiect funcțional care aduce inovație în domeniul grădinăritului de interior. Proiectul a atins scopul principal – *automatizarea monitorizării și îngrijirii plantelor* – și a pus bazele pentru evoluții viitoare semnificative. Direcțiile de extindere identificate ar transforma SmartGarden într-o platformă completă de "smart farming" la scară mică, cu posibil impact mai larg (educație ecologică, comunitate de utilizatori, etc.). Experiența dezvoltării acestui proiect de licență a fost nu doar un exercițiu tehnic, ci și o ocazie de a contribui, fie și modest, la trendul global de **smart home** și la bunăstarea plantelor care ne înconjoară. Acesta este abia începutul – cu învățămintele și entuziasmul acumulate, suntem pregătiți să ducem SmartGarden la următorul nivel.