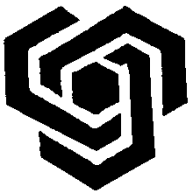


ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ФАКУЛТЕТ „КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ“

ДИПЛОМНА РАБОТА

НА ТЕМА

ПРОЕКТИРАНЕ НА ВГРАДЕНА СИСТЕМА ЗА МОНИТОРИНГ И ПОЛИВАНЕ НА СТАЙНИ РАСТЕНИЯ С УЕБ ИНТЕРФЕЙС

Изготвил: Веселин Венциславов Господинов Фак. №: 121221126

Специалност: Компютърно и софтуерно инженерство

Образователно-квалификационна степен: Бакалавър

Научен ръководител: гл. ас. д-р инж. Камелия Райнова

София, 2025

ДИПЛОМНО ЗАДАНИЕ

ДЕКЛАРАЦИЯ АВТОРСТВО

СЪДЪРЖАНИЕ

I. УВОД

В съвременното динамично и високотехнологично общество, технологиите играят все по-централна роля в ежедневието, трансформирайки начина, по който взаимодействаме със света около нас. Една от водещите концепции в тази трансформация е Интернет на нещата (IoT), която позволява свързването на физически обекти и устройства към интернет, откривайки нови възможности за автоматизация, мониторинг и управление. Паралелно с технологичния напредък, нарастващото екологично съзнание и урбанизацията засилват желанието на хората да внесат частица природа в своите домове и работни пространства. Отглеждането на стайни растения носи естетическа наслада, подобрява качеството на въздуха и допринася за по-добра психологическа среда. Въпреки това, съвременният забързан начин на живот, липсата на специфични познания или честите отсъствия от дома често правят грижата за растенията предизвикателство. Неправилното поливане, недостатъчната или прекомерна светлина, неподходящата температура и влажност са сред основните причини за неуспех в отглеждането им. В този контекст възниква необходимостта от интелигентни решения, които да улеснят и автоматизират грижата за растенията. Съществуващите на пазара комерсиални продукти често са или твърде скъпи за масовия потребител, или предлагат само частична функционалност (например, само автоматично поливане без мониторинг на други важни параметри като светлина или влажност на въздуха). Следователно, съществува реален проблем с липсата на достъпно, интегрирано и лесно за използване решение, което да предоставя комплексен локален мониторинг на ключови фактори за растежа на растенията и възможност за автоматизиран контрол.

Настоящата разработка обхваща проектирането, реализацията и тестването на функционален прототип на система за мониторинг и автоматизирано поливане, предназначен за стайни растения в домашни условия. Системата е базирана на микроконтролер ESP32 и включва сензори за температура и влажност на въздуха, интензитет на светлината, резистивен сензор за влажност на почвата и цифров сензор за ниво на водата в резервоара. Управлението на поливането се осъществява чрез малка DC водна помпа, управлявана от L298N драйвер. Комуникацията с потребителя се реализира чрез Wi-Fi в режим Access Point (AP), като системата поддържа собствен уеб сървър, достъпен от всяко устройство с уеб браузър в обхвата на мрежата.

II. АНАЛИЗ НА ПРОБЛЕМА И СЪЩЕСТВУВАЩИ РЕШЕНИЯ

Разделът има за цел да положи теоретичните основи, необходими за разбирането на разработената система, и да анализира съществуващото състояние на техниката в областта на интелигентните системи за грижа за растения. Разгледани са ключови концепции като Интернет на нещата (IoT) и вградени системи, представени са основните параметри за мониторинг на растенията и е направен преглед на релевантни комерсиални продукти.

1. Въведение в Интернет на нещата (IoT)

Интернет на нещата (Internet of Things, IoT) е парадигма, която описва мрежа от взаимосвързани физически обекти ("неща"), оборудвани със сензори, изпълнителни механизми, софтуер и комуникационни възможности, позволяващи им да събират данни, да комуникират помежду си и с други системи през интернет, често без пряка човешка намеса.[1] Концепцията, макар и датираща от края на 20-ти век, придобива огромна популярност през последните години благодарение на напредъка в миниатюризацията на електрониката, намаляването на цените на сензорите, широкото разпространение на безжичните комуникации и развитието на облачните изчисления. В основата на концепцията за Интернет на нещата (IoT) стоят няколко ключови елемента, които взаимодействат помежду си, за да осигурят функционалността на системата. Първият и най-очевиден елемент са самите "неща" (Things) – това са физически обекти, които са оборудвани с вградени сензори и/или изпълнителни механизми, като примери за такива могат да бъдат термостати, осветителни тела, сензори за влага и т.н. За да могат тези "неща" да обменят информация, е необходима свързаност, която представлява съвкупността от механизми и протоколи за комуникация както между самите "неща", така и между тях и интернет; технологии като Wi-Fi, Bluetooth, Cellular мрежи и LoRaWAN са често използвани за тази цел. Събраните данни от устройствата се нуждаят от място за съхранение, обработка и анализ, което се осигурява от платформи – това са софтуерни системи, често базирани на облачни технологии, които не само управляват потока от данни, но и предоставят инструменти за управление на самите устройства. Накрая, същинската стойност на IoT системите се реализира чрез анализ и приложения**,** които включват извличането на полезна информация от събраните данни и създаването на специфични приложения, които предоставят конкретна полза на крайния потребител, било то под формата на автоматизация на процеси, по-добър мониторинг или възможности за предиктивна поддръжка.

Типичните IoT архитектури, макар и да варират значително по своята сложност в зависимост от конкретното приложение, често могат да бъдат описани чрез няколко логически слоя, които изпълняват специфични функции. Слоят на възприятие (Perception Layer) е най-близо до физическия свят и включва самите устройства – сензори, които събират данни от околната среда, и актуатори, които могат да въздействат върху нея. Следващият е мрежовият слой (Network Layer / Connectivity Layer), който отговаря за преноса на събраните данни от устройствата към по-горните слоеве на архитектурата, използвайки различни комуникационни протоколи и технологии; в този слой често се включват и мрежови шлюзове (gateways), които агрегират данни от множество устройства. Данните, достигнали до следващото ниво, се обработват в слоя за обработка (Processing Layer / Middleware Layer), където те се съхраняват, анализират и се прилагат различни правила за управление; този слой често включва бази данни и аналитични инструменти, като при по-прости системи той може да бъде частично или изцяло реализиран на самото устройство или в локален сървър. Приложният слой (Application Layer) е този, с който крайният потребител взаимодейства най-пряко, тъй като той предоставя специфичните услуги и интерфейси, като например мобилни приложения или уеб страници. В някои по-комплексни архитектури може да съществува и опционален бизнес слой (Business Layer), който управлява цялостните приложения и услуги, базирани на събраните и анализирани IoT данни, и ги интегрира в по-широки бизнес процеси.

1.1.Комуникационни протоколи в IoT

Изборът на комуникационен протокол е от критично значение за ефективността и приложимостта на всяка IoT система, като решението зависи от комплекс от фактори, включващи изискванията за обхват на комуникацията, необходимата скорост на трансфер на данни, допустимото енергопотребление на устройствата и общата цена на имплементацията. Сред най-често използваните протоколи се открояват няколко основни технологии - Wi-Fi, Bluetooth, MQTT.

Несравнимо най-често присъстващ във вградените системи е Wi-Fi протоколът, базиран на стандарта IEEE 802.11, който предлага висока скорост на трансфер на данни. Неговата популярност го прави подходящ за приложения, където има налична съществуваща Wi-Fi мрежа и достъп до постоянно захранване, тъй като той е сравнително енергоемък. В контекста на настоящия проект, Wi-Fi се използва за осъществяване на локална комуникация между системата за мониторинг и потребителските устройства.

Bluetooth и неговата енергоефективна разновидност Bluetooth Low Energy (BLE) са изключително подходящи за комуникация на къси разстояния, като основното предимство на BLE е неговата много ниска консумация на енергия. Тези технологии често се използват за установяване на директна връзка с мобилни устройства или за създаването на персонални мрежи (Personal Area Networks, PAN), където множество устройства в непосредствена близост обменят информация.

MQTT (Message Queuing Telemetry Transport) представлява лек протокол за обмен на съобщения, базиран на модела "публикуване/абониране" (publish/subscribe). Той е оптимизиран за работа в мрежи с ниска пропускателна способност и висока латентност, което го прави изключително популярен за комуникация между IoT устройства и облачни платформи, където ефективността на преноса и минималното натоварване на мрежата са от съществено значение.

За устройства с ограничени изчислителни ресурси и памет (constrained devices), както и за мрежи с ограничени възможности (constrained networks), е разработен CoAP (Constrained Application Protocol). Този протокол работи върху UDP (User Datagram Protocol) и често се разглежда като ефективна алтернатива на HTTP за специфичните нужди на IoT средата, където олекотените протоколи са предпочитани.

Технологиите от категорията LPWAN (Low-Power Wide-Area Network), като LoRaWAN, Sigfox и NB-IoT, предлагат уникална комбинация от комуникация на много големи разстояния, достигащи километри, и изключително ниска консумация на енергия. Това се постига за сметка на по-ниска скорост на трансфер на данни. Поради тези си характеристики, LPWAN технологиите са изключително подходящи за приложения като интелигентно земеделие, където сензорите са разпръснати на големи площи и често разчитат на батерийно захранване, или за системи за градски мониторинг, изискващи широк обхват и дълъг живот на устройствата.

2. Вградени системи

Вградената система е компютърна система – комбинация от компютърен процесор, компютърна памет и входно/изходни периферни устройства – която има специфична, предварително дефинирана функция в рамките на по-голяма механична или електрическа система. За разлика от компютрите с общо предназначение (като персонални компютри),

вградените системи са проектирани да изпълняват конкретна задача или малък набор от задачи.

2.1. Характеристики и приложения

Основна характеристика на вградените системи е, че имат специфична функция - проектирани са за конкретна цел, много вградени системи трябва да реагират на събития в рамките на строго определени времеви интервали(работа в реално време), разполагат с ограничена изчислителна мощ, памет и енергия, очаква се да работят надеждно за дълги периоди без нужда от рестартиране или поддръжка, оптимизирани са за цена, размер, тегло и консумация на енергия.[2]

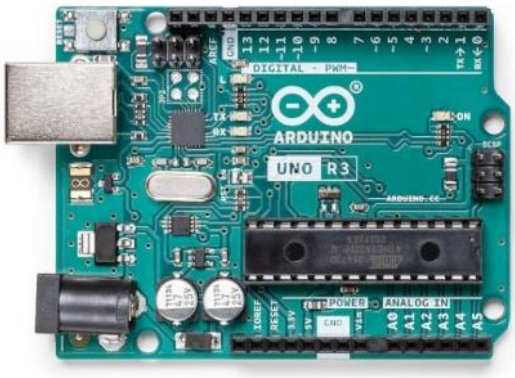
Приложенията на вградени системи са изключително разнообразни: потребителска електроника (телевизори, перални, микровълнови печки), автомобилна индустрия (управление на двигател, ABS), медицинска апаратура, индустриална автоматизация, авионика, телекомуникации и разбира се, IoT устройства.

2.2. Микроконтролери (MCU) – видове и архитектури

Сърцето на повечето съвременни вградени системи е микроконтролерът (Microcontroller Unit, MCU). Представлява компактен интегрален чип, който съдържа процесорно ядро (CPU), памет (Flash за код, RAM за данни, понякога EEPROM) и входно/изходни периферии (GPIO, таймери, серийни интерфейси като UART, SPI, I²C, ADC, DAC) на един чип.

Съществуват различни класове MCU: 8-битови (напр. AVR фамилията на Atmel/Microchip, PIC), 16-битови, 32-битови (доминиращи днес, напр. ARM Cortex-M, ESP32, RISC-V) и 64-битови (по-рядко срещани в класически MCU). В тях се ипълняват две класически архитектури Харвардска (отделни шини за код и данни) или Фон Нойманова (обща шина). Притежават CISC (Complex Instruction Set Computer) или RISC (Reduced Instruction Set Computer) набор от инструкции.[3]

Arduino Uno (Фиг. 1.): 8-битови, лесни за използване с огромна общност от разработчици, но с ограничени ресурси и без вградена безжична свързаност. [5]



Фиг. 1. Arduino Uno

ESP8266/ESP32 (Фиг. 2.): 32-битови MCU с интегрирани Wi-Fi и Bluetooth модули. Предлагат отлична комбинация от производителност, ниска цена и свързаност, което ги прави изключително популярни за IoT проекти. ESP32 е избран за настоящата дипломна работа.

Фиг. 2. ESP32

Raspberry Pi (Фиг. 3.): двуядрен ARM Cortex-M0+ микроконтролер, разработен от Raspberry Pi Foundation, предлагащ добра производителност на ниска цена, но без вградена безжична свързаност в базовия модел.[6]

Фиг. 3. Raspberry Pi

Сравнителната таблица (Таблица 1.) илюстрира разликите между ESP32, Arduino Uno и Raspberry Pi .

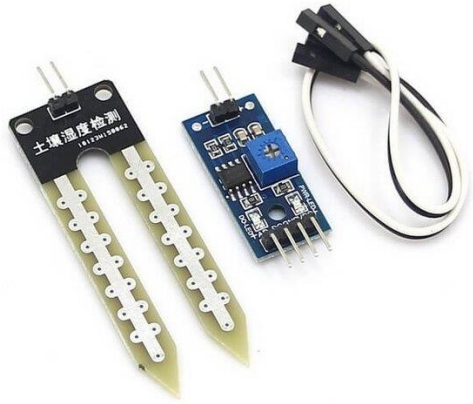
Характеристика	ESP32	Arduino Uno	Raspberry Pi
Процесор	Двуядрен, 240MHz	8-битов, 16 MHz	Четириядрен, 1.2 GHz
RAM	520 KB	2 KB	1 gB
Флаш памет	До 16 MB	32 KB	MicroSD карта
Комуникация	Wi-Fi, Bluetooth, UART, I^2C, SPI	UART, I^2C, SPI	Wi-Fi, Bluetooth, Ethernet, GPIO
GPIO пинове(брой)	36	14	40
Аналогови входове	18 канала, 12-битова резолюция	6 канала, 10-битова резолюция	Липсва
Операционна система	FreeRTOS	Няма	Linux (Raspbian)
Цена	5-10	20-30	35-70

Таблица 1. Сравнение между различни микроконтролери

3. Системи за мониторинг и автоматизация на грижата за растения

Автоматизираните системи за грижа за растения целят да улеснят процеса на отглеждане чрез наблюдение на ключови параметри и предприемане на автоматични действия, най-често поливане. За оптимален растеж растенията се нуждаят от балансирана среда. Основните параметри, които влияят на тяхното развитие и често се наблюдават от интелигентните системи, са: влажност на почвата, може би най-критичният параметър,

както недостигът (засушаване), така и излишъкът на вода са вредни. Влажността се измерва като процент от обемното водно съдържание (%) или чрез косвени методи. Сензорите измерващи влагата в почвата са два вида: резистивен и капацитивен. Резистивният сензор (Фиг. 4.) измерва съпротивлението на почвата между два електрода. Евтин и лесен за използване, но е податлив на корозия, което води до неточности и кратък живот, и се влиае силно от солеността на почвата.



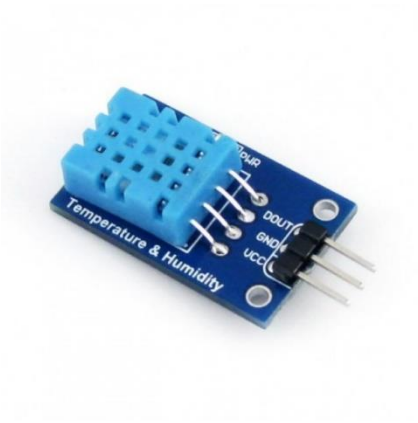
Фиг. 4. Резистивен сензор за влага на почвата

Капацитивният сензор (Фиг. 5.) измерва диелектричната проницаемост на почвата, която е силно зависима от водното съдържание. Не е податлив на корозия, тъй като електродите са изолирани, и са по-малко чувствителни към солеността, което ги прави по-точни и надеждни в дългосрочен план.[7]

Фиг. 5. Капацитивен сензор за влага на почвата

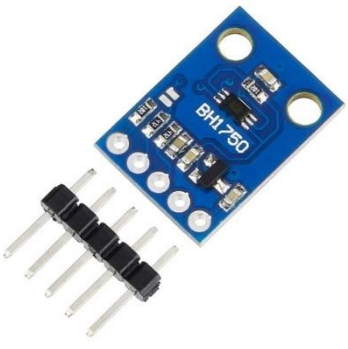
Температурата на въздуха е ключова характеристика, която пряко влияе на скоростта на метаболитните процеси като фотосинтеза и дишане. Повечето стайни растения

предпочитат температури в диапазона 18-24°C, като екстремните стойности могат да забавят растежа или да причинят увреждания. Влажност на въздуха, пък, влияе на скоростта на транспирация (изпарение на вода от листата). Прекалено ниската влажност може да причини изсъхване на краищата на листата, докато твърде високата (над 80-85%) може да създаде условия за развитие на гъбични заболявания. Избран е DHT11 сензор (Фиг. 6.).



Фиг. 6. DHT11 сензор за влага и температура на въздуха

Интензитет на светлината, това е източникът на енергия за фотосинтезата. Важни са както интензитетът, така и продължителността на осветяване и спектралният състав. Различните растения имат различни нужди – от пълно слънце до сянка. Измерването обикновено се прави в луксове (lx), което е мярка за осветеност, видима за човешкото око, или в PPFD (Photosynthetic Photon Flux Density, μmol/m²/s), което измерва количеството фотосинтетично активно лъчение.[10] Сензорът, избран в дипломната работа, BH1750 (Фиг. 7.) измерва в луксове.



Фиг. 7. Сензор за интензитет на светлината BH1750

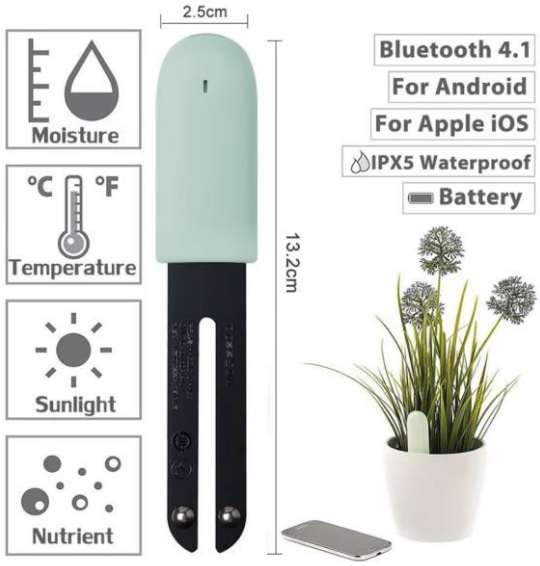
Сензор за ниво на водата в резервоара (Фиг. 8.), макар и да не влияе пряко на растението, този параметър е важен за автоматизираните поливни системи, за да се

гарантира, че има наличен ресурс за поливане и да се предотвратят повреди на помпата при работа на сухо.

Фиг. 8. Цифров сензор за ниво на водата

3.1. Анализ на съществуващи комерсиални продукти

На пазара съществуват редица продукти, насочени към автоматизация на грижата за растения. Типичен пример за това е Xiaomi Mi Flora (Фиг. 9.), състои се от сензор за влага на почвата (резистивен), температура, светлина (lux). Използва за комуникация: Bluetooth Low Energy (BLE) и изисква мобилно устройство в обхват за синхронизация. Поддържа само мониторинг, няма автоматизация. Сравнително ниска цена. Важни недостатъци са: резистивният сензор (корозия, неточност), само мониторинг, изисква BLE връзка, батерийно захранване.



Фиг. 9. Xiaomi Mi Flora

Друг вариант е Parrot Pot (Фиг. 10.), който представлява интегрирана система за автоматично поливане ("самополиваща се саксия"), съдържа сензори за влага на почвата, температура, светлина, ниво на водата в резервоара на саксията. Поддържа Bluetooth Low Energy (BLE). Неговите недостатъци са - високата цена и, че вече е спрян от производство.

Фиг. 10. Parrot Pot

Gardena Smart System (Фиг. 11.) – цялостна модулна система отделен Smart Sensor (влага на почва, температура, светлина), Smart Water Control (управление на клапан), Smart Power Adapter и др. Поддържа собствен безжичен протокол (868 MHz) към Smart Gateway, свързан към интернет (Wi-Fi/Ethernet). Изисква Gateway и облачна връзка. Автоматизира поливането чрез вече създадени графици и данни от сензори, управлявано през мобилно приложение и облак. Главен недостатък е високата цена.

Фиг. 11. Gardena Smart System

Netro Sprite (Фиг. 12.) е интелигентен контролер за поливане, който използва метеорологични данни и данни за почвата (въведени от потребителя или от опционалния сензор Whisperer - влага на почва, температура, светлина). Комуникация: Sprite (Wi-Fi към облак), Whisperer (безжично към Sprite/облак). Предоставя интелигентно планиране на поливането, базирано на прогнози, правила и данни от сензори. Управлява външни клапани. Недостатъци - основно за външни поливни системи (управлява клапани), зависим от облак и е сравнително скъп.

Фиг. 12. Netro Sprite

Анализът на съществуващите решения показва, че комерсиалните продукти често предлагат удобен потребителски интерфейс (мобилни приложения) и интеграция с облачни услуги, но са или скъпи (Gardena, Netro), или с ограничена функционалност/точност (Mi Flora), или вече не се произвеждат (Parrot Pot). Често липсва възможност за лесна модификация или локален контрол без зависимост от интернет/облак. Настоящата дипломна работа има за цел да се създаде достъпна, интегрирана система, която преодолява някои от недостатъците на съществуващите решения.

III. ПРОЕКТИРАНЕ НА ХАРДУЕРНА ПЛАТФОРМА

Този раздел описва процеса на проектиране на IoT базираната система за мониторинг и автоматизирано поливане на растения. Дефинирани са функционалните и нефункционалните изисквания, представена е общата архитектура на системата и са описани детайлно проектирането на хардуера и софтуера, включително обосновка на избора на компоненти и алгоритми

1. Спецификация на изискванията

Спецификацията на изискванията е фундаментална стъпка в процеса на проектиране, тъй като тя дефинира точно какво трябва да прави системата и какви са критериите за оценка на нейното качество и работа. Тези изисквания се разделят на две основни категории: функционални, които описват конкретните действия и възможности на системата, и нефункционални, които задават характеристики като производителност, надеждност и лесната на използване.

По отношение на функционалните изисквания, системата трябва преди всичко да осъществява мониторинг и да предоставя данни в реално време за редица ключови параметри, които са от съществено значение за растежа на растенията. Тези параметри включват температурата на въздуха, измервана в градуси Целзий, влажността на въздуха, изразена в проценти, интензитета на околната светлина, представян в луксове, влажността на почвата, която може да бъде представена като аналогова стойност или чрез относителна скала, и нивото на водата в резервоара, индикирано в процентна скала.

Освен събирането на данни, системата трябва да осигури достъпен през Wi-Fi уеб интерфейс. Този интерфейс трябва да визуализира текущите стойности на всички измервани параметри и да функционира в режим Wi-Fi Access Point (AP), като по този начин създава собствена безжична мрежа за директна връзка с потребителски устройства.

Друга важна функционалност е автоматизираното поливане. Системата трябва автоматично да активира водната помпа за предварително определен период, когато измерената влажност на почвата падне под зададен в програмния код праг и същевременно има налично достатъчно ниво на вода в резервоара. С цел защита на помпата, системата трябва да предотвратява нейното включване или да спира незабавно работата ѝ, ако сензорът за ниво отчете липса на вода в резервоара.

Системата трябва също така да осигурява визуална и звукова сигнализация. Визуалната сигнализация, реализирана чрез светодиоди (LEDs), трябва да индикира определени условия, като например ниска или висока температура, както и статуса на работа на уеб сървъра. Звуковата сигнализация, осъществявана чрез зумер, трябва да предупреждава потребителя при ниско ниво на водата в резервоара. Накрая, системата трябва да предлага базова конфигурируемост, позволяваща лесна промяна в програмния код на основни параметри като името на Wi-Fi мрежата (SSID) и паролата за достъп в AP режим.

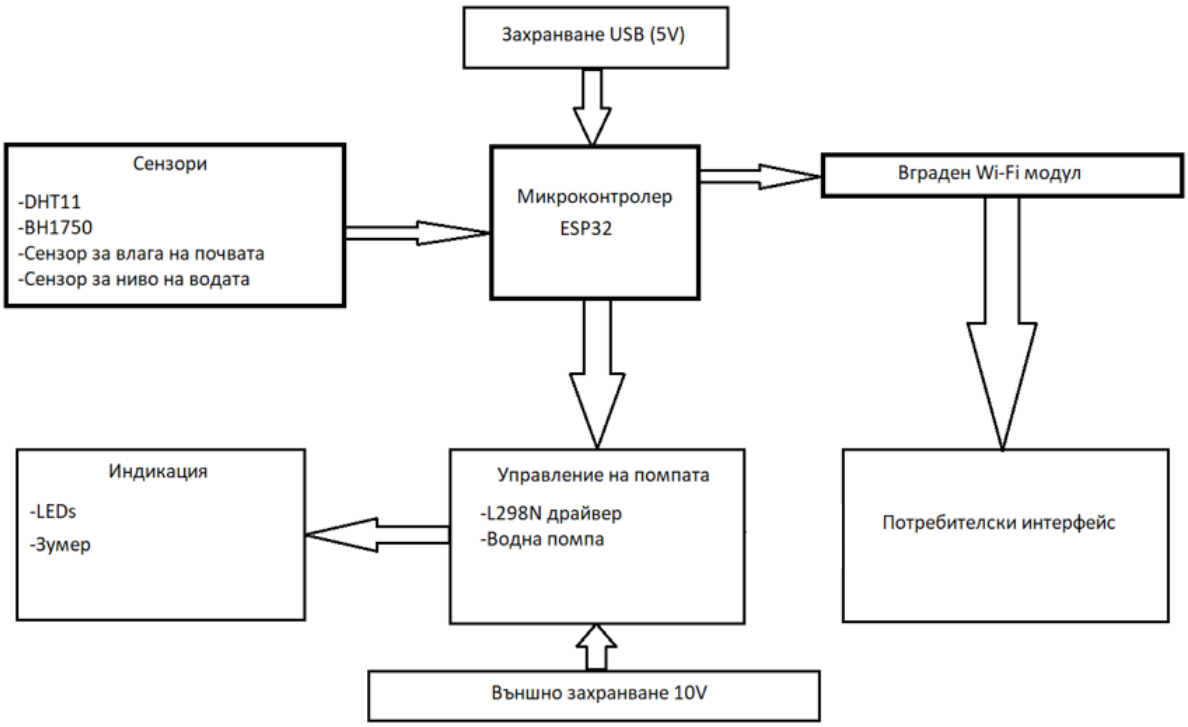
По отношение на нефункционалните изисквания, от ключово значение е системата да бъде реализирана с широкодостъпни и евтини компоненти, като общата цена на хардуера, без да се включват захранването и корпусът, остане в приемливи граници. Уеб интерфейсът трябва да бъде интуитивен и лесен за разбиране дори от потребители без задълбочени технически познания, а процесът на свързване към системата трябва да става лесно чрез стандартните Wi-Fi настройки на потребителското устройство.

Надеждността на системата е друго важно нефункционално изискване - тя трябва да работи стабилно за продължителни периоди и да разполага с базови механизми за справяне с грешки, които могат да възникнат при четене на данни от сензорите. Производителността на уеб интерфейса също е от значение - той трябва да се зарежда в рамките на около пет секунди след установяване на връзка, а обновяването на данните от сензорите на интерфейса следва да се извършва в реално време.

Захранването на системата, с изключение на водната помпа, трябва да може да се осъществява стандартно през USB порт, подаващ 5V. Захранването на самата помпа изисква отделен източник, съобразен с нейните специфични нужди и характеристиките на използвания L298N драйвер. Не на последно място, системата трябва да бъде напълно функционална без необходимост от връзка към интернет или зависимост от външни облачни услуги, което гарантира нейната автономност и поверителност на данните.

1.1. Обща блокова схема

Системата е базирана на централен микроконтролер (ESP32), който координира работата на всички останали компоненти (Фиг. 13.).



Фиг. 13. Обща блокова схема на системата

1.2. Поток на данните

ESP32 периодично прочита данните от свързаните сензори: DHT11 (температура и влажност), BH1750 (светлина) през I²C интерфейс, резистивен сензор за влажност на почвата през аналогов вход (ADC), цифров сензор за ниво на водата през цифров изход. Микроконтролерът обработва суровите данни (напр. преобразува аналоговата стойност от почвения сензор). Въз основа на обработените данни (основно влажност на почвата и ниво на водата) и дефинираната логика, ESP32 решава дали да активира помпата. ESP32 изпраща управляващи сигнали към: L298N драйвер за включване/изключване на помпата, LEDs за визуална индикация на състоянието, зумер за звукова сигнализация. Отделно микроконтролерът работи като Wi-Fi Access Point и слуша за HTTP заявки на порт 80. При заявка от клиент (уеб браузър), традиционните уеб сървъри, често използвани с Arduino и ESP32, като стандартния WiFiServer, работят по синхронен, блокиращ начин. Това означава, че когато сървърът обработва заявка от клиент, останалата част от програмата (например четене на сензори или друга логика в loop()) трябва да изчака. При множество едновременни клиенти или по-дълго обработвани заявки, това може да доведе до значителни забавяния и усещане за неотзивчивост на системата.

Асинхронният уеб сървър, реализиран чрез библиотеката ESPAsyncWebServer, решава този проблем, като работи по неблокиращ начин. Вместо да спира изпълнението на основния код, докато чака данни от клиент или изпраща отговор, асинхронният сървър използва система от обратни повиквания (callbacks). Когато настъпи събитие, като например получаване на нова заявка или завършване на изпращането на данни, сървърът извиква съответната предварително дефинирана функция (callback). Това позволява на ESP32 да продължи да изпълнява други задачи, докато се осъществява комуникацията с клиентите.

Предимствата на използването на асинхронен уеб сървър са значителни, особено за устройства с ограничени ресурси като ESP32. Така може да обслужва много повече едновременни клиентски връзки, тъй като не се блокира от всяка една. Отзивчивостта на цялостната система се подобрява, тъй като критични задачи като четене на сензори или управление на актуатори не се забавят от работата на уеб сървъра. Също така, асинхронните сървъри често предлагат по-удобни начини за обслужване на статични файлове (HTML, CSS, JavaScript), включително и от файлова система като SPIFFS. Това прави разработката на по-сложни и интерактивни уеб интерфейси по-лесна и ефективна.

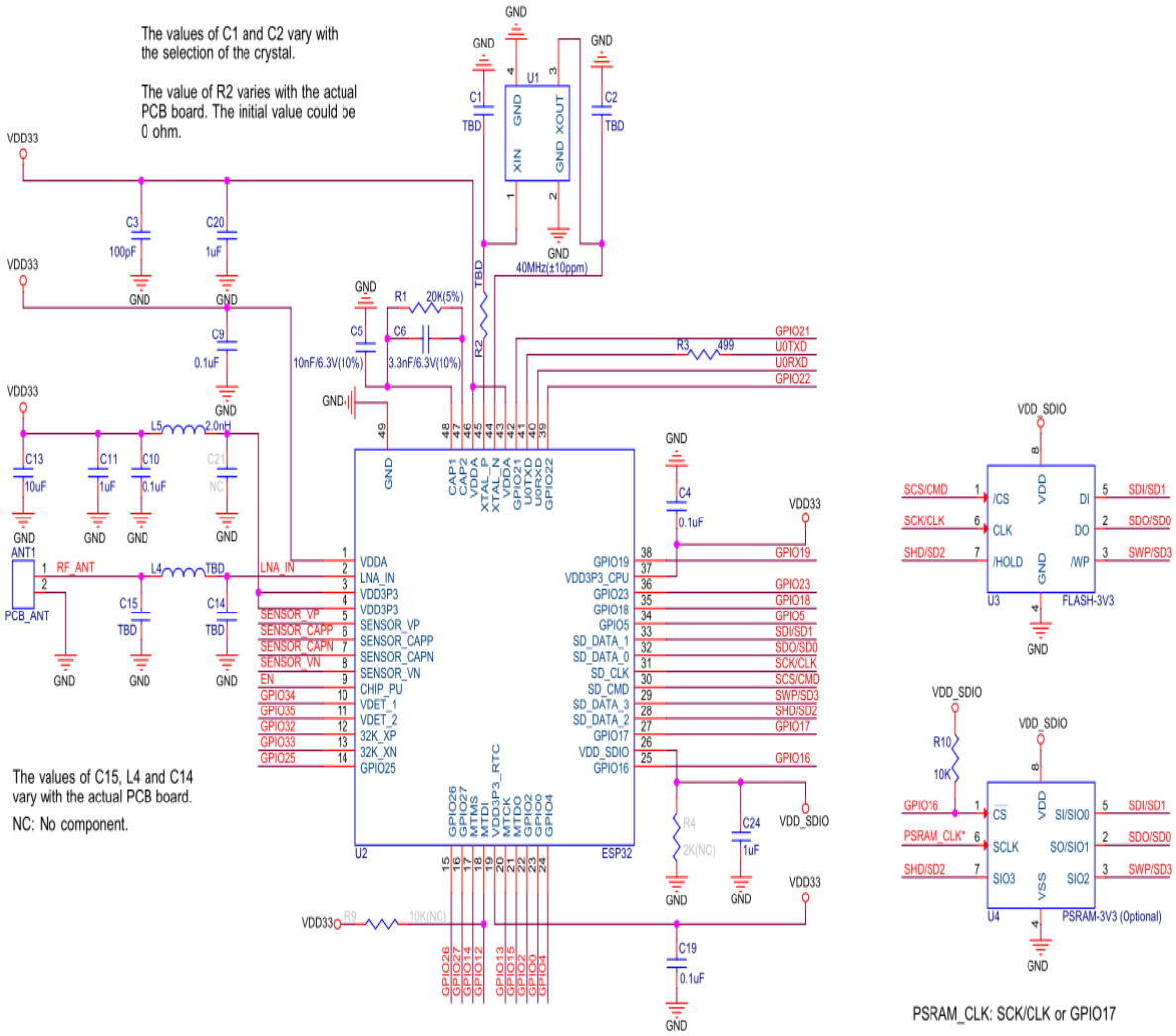
SPIFFS е лека файлова система, специално проектирана за микроконтролери с флаш памет, свързана чрез SPI (Serial Peripheral Interface), каквато е вградената флаш памет на ESP32. Тя позволява на ESP32 да съхранява и управлява файлове по начин, подобен на този при по-големите компютърни системи, макар и с някои специфични ограничения.

Основното предназначение на SPIFFS е да осигури място за съхранение на конфигурационни файлове, уеб страници (HTML, CSS, JavaScript, изображения), малки бази данни, логове или други данни, които трябва да останат запазени дори след рестартиране на устройството. Това е изключително полезно за уеб сървъри, тъй като позволява HTML и другите уеб активи да бъдат съхранени като отделни файлове, вместо да се вграждат директно в програмния код като големи низове. Това прави кода по-чист, по-лесен за поддръжка и позволява по-лесно актуализиране на уеб интерфейса, без да се налага прекомпилиране на целия фърмуер. Размерът на файловата система SPIFFS се конфигурира при компилиране на фърмуера и заема част от общата флаш памет на ESP32. Работата с файлове в SPIFFS (четене, запис, изтриване, изброяване) се осъществява чрез специфичен API, предоставен от съответните библиотеки за Arduino IDE. За да се качат файлове в SPIFFS, обикновено се използва плъгин за Arduino IDE (като "ESP32 Sketch Data Upload")

или други инструменти, които позволяват файлове от компютъра да бъдат записани директно във файловата система на ESP3

2. Проектиране на хардуера

Тази част от дипломната работа описва избора на хардуерните компоненти и начина им на свързване за изграждане на системата, като всеки избор е аргументиран спрямо дефинираните изисквания.



Фиг. 14. Електронна схема на микроконтролер ESP32

Представената електронна схема (Фиг. 14.) илюстрира типичната референтна конфигурация за имплементация на микроконтролер ESP32, обозначен като U2, който е централният чип на системата. На схемата ясно се виждат множеството изводи (пинове) на ESP32, всеки със специфично наименование и функция. Те включват захранващи пинове

като VDDA, VDD3P3 и GND, пинове за свързване на антената като LNA_IN, както и пинове за кварцовия осцилатор XTAL_P и XTAL_N. От съществено значение са и GPIO пиновете (General Purpose Input/Output), които се използват за осъществяване на комуникация с други компоненти, сензори и актуатори. Схемата показва също така пинове, предназначени за комуникационни интерфейси като SPI (SD_CLK, SD_CMD, SD_DATA_0-3) и UART (U0TXD, U0RXD). Специализирани пинове като SENSOR_VP, SENSOR_VN, SENSOR_CAPP и SENSOR_CAPN са свързани със сензорната част на чипа, докато пинове като EN (Enable), CHIP_PU (Chip Power Up) и групата MTDI, MTCK, MTMS, MTDO (JTAG интерфейс) служат за управление, конфигурация и дебъг.[4]

Непосредствено до ESP32 е разположен U1, който представлява кварцов осцилатор, обикновено с честота 40MHz. Този кварцов кристал осигурява прецизния тактов сигнал, необходим за синхронната работа на микроконтролера. Кондензаторите C1 и C2 са свързани паралелно на кристала и техните стойности могат да варират в зависимост от спецификациите на конкретния кристал, а R2 е резистор, също част от осцилаторната верига.

Радиочестотната част на системата е представена от ANT1, която показва свързването на антената, често реализирана като PCB антена (PCB_ANT), към RF входа на ESP32. Компонентите L4, L5, C13, C10, C11, C14, C15 и C21, представляващи индуктивности и кондензатори, формират съгласуваща верига и филтри за RF сигнала. Тяхната роля е да осигурят оптимална работа и производителност на вградените Wi-Fi и Bluetooth модули.

За съхранение на програмния код (firmware) ESP32 обикновено разчита на външна SPI Flash памет, обозначена като U3 (FLASH-3V3). Схемата детайлно показва как тази памет е свързана към съответните SPI пинове на ESP32, използвайки сигнали като CS (Chip Select), CLK (Clock), DI (Data In), DO (Data Out), WP (Write Protect) и HOLD. Някои версии на ESP32 поддържат и външна псевдостатична RAM (PSRAM) за разширяване на наличната оперативна памет, като свързването на такъв опционален чип, U4 (PSRAM-3V3), също е показано на схемата, използвайки SPI интерфейс и pull-up резистор R10.

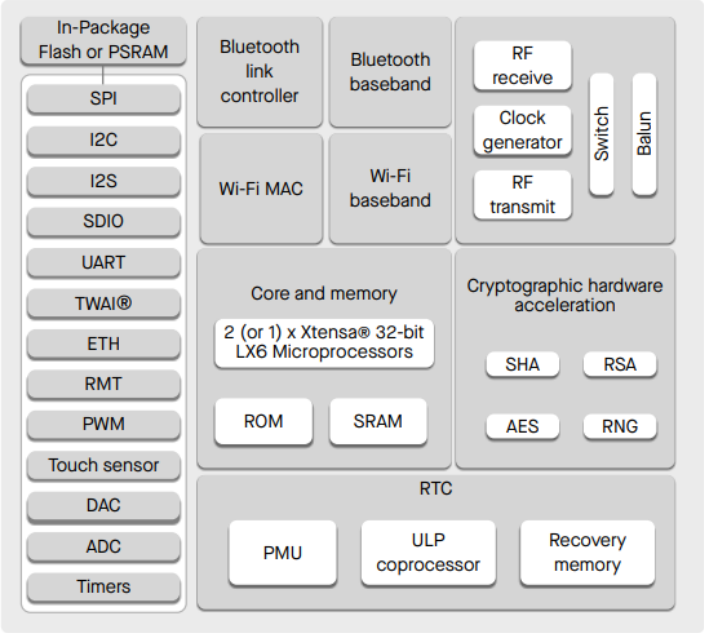
Ключова част от дизайна са захранващите и филтриращи компоненти. Основното захранващо напрежение за ESP32, VDD33, обикновено е 3.3V. Множество кондензатори, като C3, C20, C9, C5, C6, C4, C24 и C19, с различни стойности (например 100pF, 1uF, 0.1uF,

10nF, 3.3nF), са стратегически разположени в близост до захранващите пинове на ESP32 и другите интегрални схеми. Тяхната функция е да филтрират захранващото напрежение, да елиминират шумове и да осигурят стабилно и чисто захранване за всички компоненти. Наред с кондензаторите, в схемата са включени и различни резистори – R1, R3, R4, R9 – които изпълняват разнообразни функции като pull-up (R4, R9), pull-down, токоограничаване (например R3 с номинал 499 Ohm за GPIO21, U0TXD, U0RXD, GPIO22) или служат като част от делители на напрежение.[4]

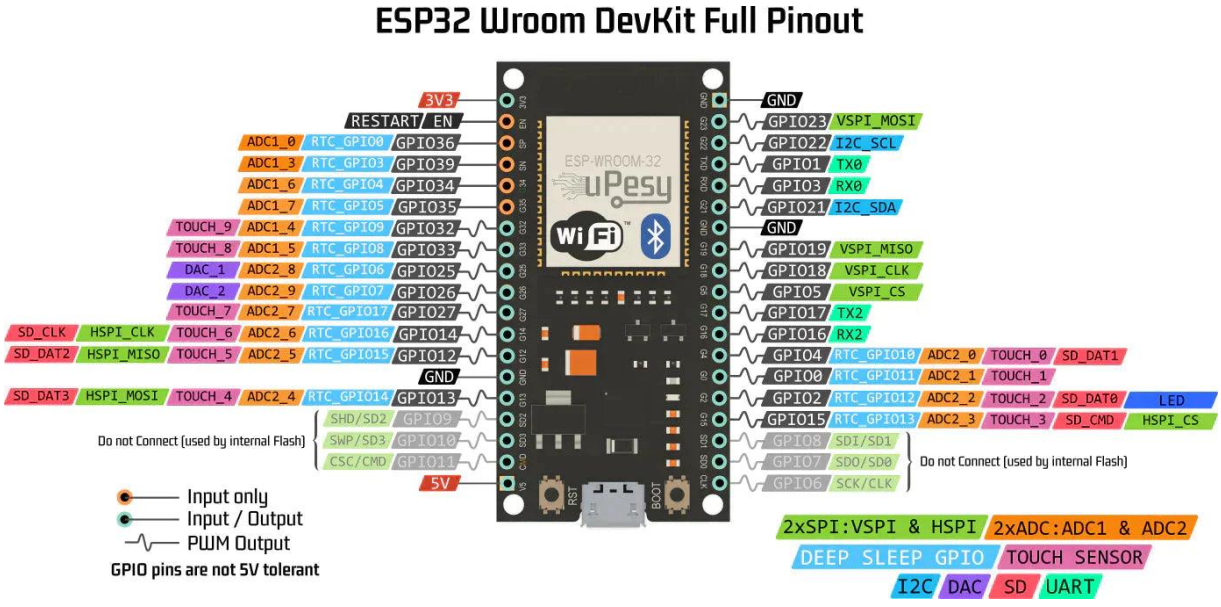
Схемата показва и изведени комуникационни изводи като GPIO21, U0TXD, U0RXD и GPIO22. Тези пинове са предназначени за осъществяване на серийна комуникация (UART), която е важна за програмиране и дебъг на системата, както и за общо предназначение, позволявайки свързване на допълнителни периферни устройства. Серийният резистор R3, свързан към тези линии, често се използва за защита на пиновете или за съгласуване на импедансите.

2.1. Избор на микроконтролер

За централен управляващ модул е избран ESP32**,** защото разполага с вградени Wi-Fi (802.11 b/g/n) и Bluetooth (Classic + BLE) модули (Фиг. 15.), което е ключово за изискването за безжична комуникация без нужда от допълнителни модули, опростявайки дизайна и намалявайки цената. Двуядреният Tensilica LX6 процесор (до 240 MHz) и достатъчното количество RAM (520KB SRAM) осигуряват необходимата изчислителна мощ за едновременно четене от множество сензори, изпълнение на логика за управление, работа на уеб сървър и Wi-Fi комуникация. Разполага с богат набор от периферии, включително множество GPIO пинове (Фиг. 16.), няколко ADC канала (за аналоговия почвен сензор), I²C интерфейс (за BH1750), PWM канали (за управление на зумер, LED, скорост на помпа) и UART (за програмиране и дебъг). ESP32 модулите и развойните платки са широкодостъпни и на много евтини. Има огромна общност от разработчици и богата документация, както и отлична поддръжка в Arduino IDE, което улеснява разработката. Другите микроконтролери като Arduino Uno, който няма вградена Wi-Fi свързаност и е със значително по-малко ресурси и Raspberry Pi , който е много по-мощен, но и по-скъп, по-енергоемък и по-сложен за задачи, изискващи основно управление на входове/изходи в реално време, не са оптимални за целта на дипломната работа.



Фиг. 15. [Функционална блокова диаграма на ESP32]



Фиг. 16. [Пинове на ESP32]

2.2. Избор на сензори

Температура и влажност на въздуха: DHT11 - избран е поради изключително ниската си цена и широка достъпност. Използва прост едножичен цифров протокол, който се поддържа лесно от ESP32 с готови библиотеки. Въпреки че точността му (±2°C за

температура, ±5% за влажност) е по-ниска от тази на DHT22 или BME280, тя се счита за достатъчна за нуждите на мониторинг на стайни растения в домашни условия, където се следят по-скоро тенденциите, отколкото абсолютните стойности.

Интензитет на светлината: BH1750 - избран е пред по-простия фоторезистор (LDR), тъй като предоставя директно измерване в луксове (lx) с добра точност и широк динамичен обхват (до ~65535 lx), което е по-подходящо за оценка на условията за растеж. Използва стандартен I²C интерфейс, което улеснява свързването и намалява броя на необходимите аналогови входове.

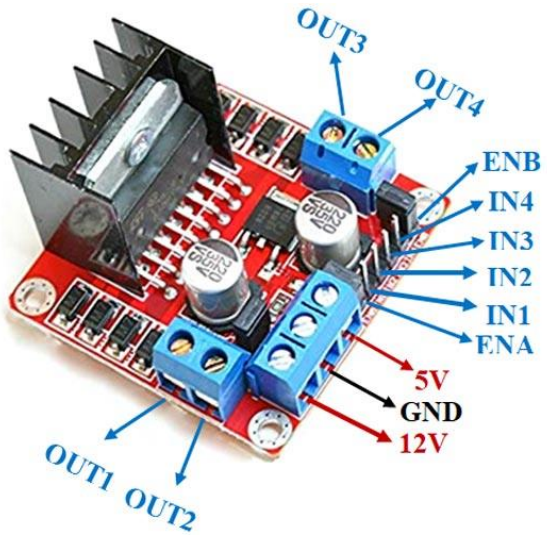
Влажност на почвата: избран е по-евтиния резистивен тип. Капацитивният метод е и по-малко чувствителен към солеността на почвата. Предоставя аналогов изходен сигнал, пропорционален на влажността, който лесно се чете от аналогово-цифровия преобразувател на ESP32.

Ниво на водата в резервоара: цифров сензор – избран поради своята простота, ниска цена и надеждност за детекция на състояния (ниво над/под сензора).

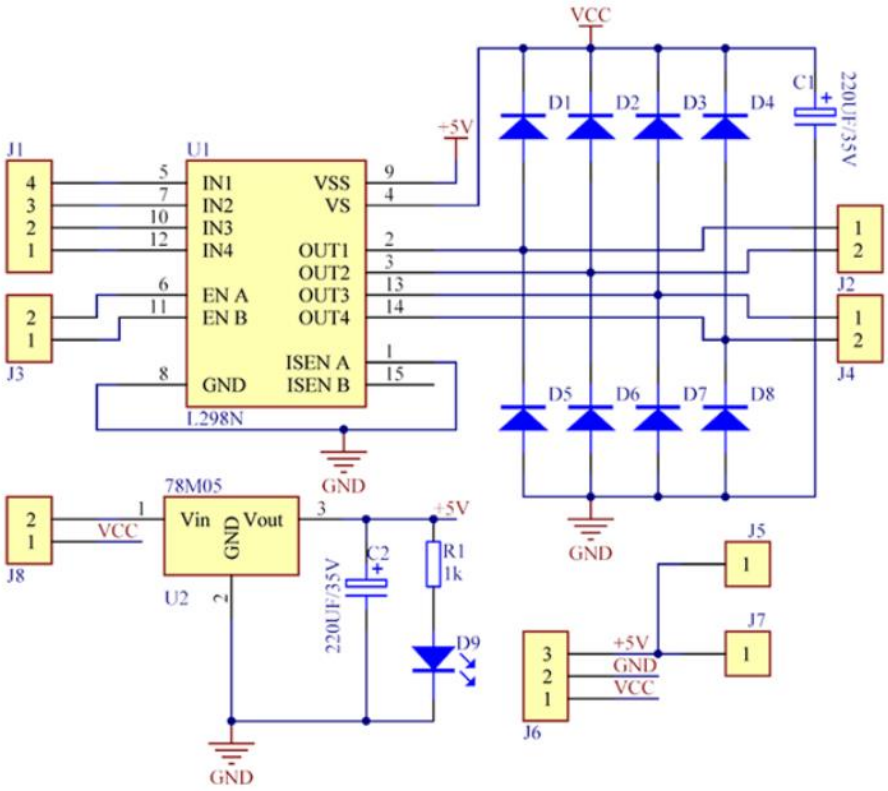
2.3. Избор на други компоненти

Водна помпа: малка потопяема DC помпа (3-5V, ~120 L/h) - тези помпи са евтини, компактни и работят на ниско напрежение, което е съвместимо с логическите нива на микроконтролера (чрез драйвер). Дебитът е достатъчен за поливане на стайни растения. Тъй като е потопяема, трябва да се постави директно в резервоара с вода.

Драйвер за помпа: L298N модул (Фиг. 17.) – популярен, евтин и лесен за управление H-мостов драйвер (Фиг. 18.). Той може да управлява два DC мотора или четири помпи , като позволява контрол на посоката (макар тук да е нужна само една посока) и скоростта (чрез PWM на Enable пиновете, въпреки че тук се ползва само за вкл./изкл.). Може да работи с напрежения за мотора, различни от логическото напрежение, и да осигури необходимия ток за помпата, който ESP32 не може да достави директно[9].



Фиг. 17. Драйвер L298N



Фиг. 18. Електрическа схема на L298N

Звукова сигнализация: пасивен зумер (Фиг. 19.)- той е изключително евтин и изисква само подаване на сигнал с променлива честота (PWM/tone) от ESP32, за да генерира звук, което позволява гъвкавост при избора на тоналност за различни сигнали.

Фиг. 19. Пасивен зумер

Визуална сигнализация: стандартни светодиоди (LEDs) - те са най-простият и евтин начин за визуална индикация. Използват се няколко светодиода с различни цветове за индикиране на различни състояния на системата, като се свързват към цифрови изходи на ESP32 през токоограничаващи резистори.

2.4. Захранване

Системата е проектирана да бъде максимално лесна за захранване - директно през своя Micro-USB порт с 5V от стандартно USB зарядно или компютърен порт. Вграденият регулатор на платката осигурява необходимото 3.3V за самия чип и може да захрани и сензорите, които работят на 3.3V (BH1750). Платката осигурява и 5V изход (Vin), който може да се използва за други компоненти, ако общата консумация не надвишава възможностите на USB източника.

DHT11 и резистивният сензор за почва могат да работят както на 3.3V, така и на 5V. BH1750 обикновено работи на 3.3V. За простота, всички сензори, които го позволяват, могат да се захранят от 3.3V изхода на ESP32 платката.

L298N драйвер: логическата част на L298N (VCC) се захранва с 5V, което може да се вземе от 5V (Vin) изхода на ESP32 платката. Захранването за мотора (помпата) (VMS) се подава отделно. Тъй като помпата е нисковолтова (3-5V), VMS може също да се свърже към 5V (Vin) изхода на ESP32 платката, ако USB източникът може да осигури достатъчно ток (~1A пиков при старт на помпата). За по-голяма стабилност, VMS на L298N да се захрани от отделен 5V източник), като земята (GND) на този източник се свърже към общата земя (GND) на ESP32.Водната помпа се свързва се към изходите (OUT1/OUT2) на L298N.

Зумер и LEDs: Захранват се директно от цифровите изходи на ESP32 (през резистори за LEDs), консумацията им е пренебрежимо малка.

IV. РАЗРАБОТВАНЕ НА СОФТУЕР

Този раздел описва дизайна на програмното осигуряване (firmware), което управлява ESP32 микроконтролера и реализира функционалните изисквания на системата. Описани са изборът на среда и език, общата софтуерна архитектура, основните алгоритми, комуникационният модел и дизайнът на потребителския интерфейс.

1. Избор на развойна среда и език

Развойната среда Arduino IDE (Фиг. 20.) е избрана поради своята простота, леснота на използване и отлична поддръжка за ESP32 чрез допълнителен Board Manager. Предоставя голям брой готови библиотеки за работа със сензори, Wi-Fi и други периферии, което значително ускорява процеса на разработка. Макар да не предлага толкова разширени функции за дебъгване като PlatformIO или ESP-IDF (Espressif IoT Development Framework), за целите на този прототип и предвид изискването за достъпност, Arduino IDE е напълно достатъчна. Arduino използва C++ като основен език, но с опростен синтаксис и набор от функции, предоставени от Arduino Framework. Това позволява както структурирано програмиране, така и използване на обектно-ориентирани подходи, ако е необходимо. Езикът C++ осигурява добър контрол върху хардуера и ефективност, което е важно за вградени системи.

Фиг. 20. Arduino IDE

1.1. Софтуерна архитектура

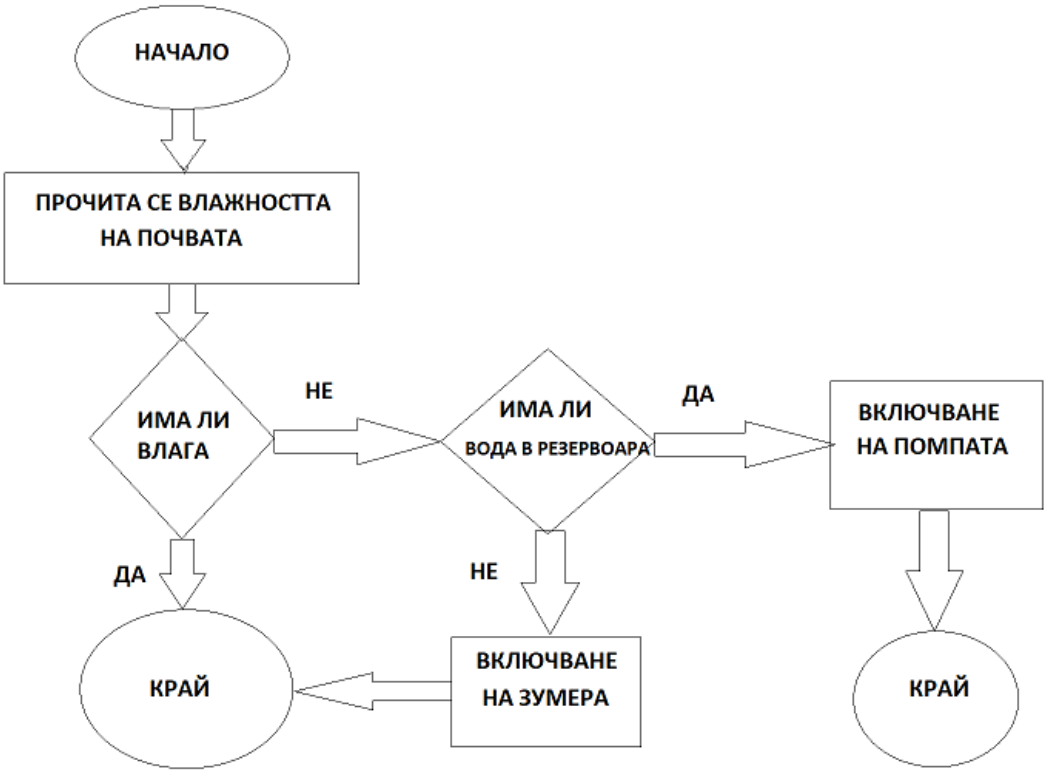
Софтуерът е проектиран като монолитно приложение, изпълняващо се в основния цикъл (loop()) на Arduino. Не се използва операционна система в реално време (RTOS) като FreeRTOS (въпреки че ESP32 я поддържа), за да се запази простотата на кода за този прототип. Основните компоненти на архитектурата са: инициализация (setup()) и основен цикъл (loop()). В setup()) се инициализират GPIO пиновете (вход/изход), сензорите (DHT, BH1750, I^2C), конфигурира се серийния порт за дебъг и се настройва и стартира Wi-Fi връзка в режим Access Point, стартира се уеб сървърът. В loop() периодично се четат стойностите от всички сензори, обработват се данните (проверка за валидни стойности - isNan), прилага се логиката за автоматично поливане и сигнализация. Използвани са стандартни и външни библиотеки за Wi-Fi връзка (WiFi.h), за уеб сървъра (WiFiClient.h), за DHT сензора (DHT.h), за BHT1750 сензора (BH1750.h), за I^2C комуникация (Wire.h), за асинхронен сървър (ESPAsyncWebServer.h), за файлова система (SPIFFS.h) .Използвани са глобални променливи за съхранение на текущите стойности от сензорите, състоянието на системата и конфигурационни параметри (прагове, пинове).

1.2. Проектиране на алгоритми

При четенето на стойности от сензорите са придвидени следните алгоритми: за DHT11 - извикване на dht.readTemperature() и dht.readHumidity(). Включена е проверка за isNaN за валидиране на прочетените стойности (NFR3). За BH1750 - Извикване на lightMeter.readLightLevel(). Библиотеката управлява IC комуникацията. За резистивния сензор - извикване на analogRead(ANALOG_SOIL1). Получената стойност е между 0-4095 (за ESP32 ADC) и се използва директно или може да се обвърже към процентна скала след калибриране.

Логика за автоматично поливане (в loop()) – първо се прочита влажността на почвата (soilMoisture), после се прочитане на състоянието на сензора за ниво(WATER_LEVEL_PIN). Условието за стартиране на помпата е, ако soilMoisture е над определен праг (тъй като по-висока аналогова стойност често означава по-суха почва при този тип сензори) и WATER_LEVEL_PIN показва наличие на вода следва включване на помпата (digitalWrite(IN1_PIN, HIGH), analogWrite(ENA_PIN, 255) - или директно управление на ENA, ако не се ползва PWM). Предварително е зададен таймер за продължителност на поливането (3 секунди). Условието за спиране на работа е, ако помпата работи и времето за

поливане е изтекло или WATER_LEVEL_PIN показва липса на вода, следва изключване на помпата (digitalWrite(IN1_PIN, LOW), analogWrite(ENA_PIN, 0)). Ако нивото на водата е под определена граница се включва зумера при активиране на функцията tone(BUZZER_PIN, frequency). По-долу е показана проста блок схема на алгоритъма (Фиг. 21.).



Фиг. 21. Блок схема на алгоритъма

1.3. Проектиране на комуникацията

Системата работи като Access Point (AP). Това означава, че ESP32 създава собствена Wi-Fi мрежа с дефинирани SSID и парола. Потребителите се свързват директно към тази мрежа. Така не се изисква съществуваща Wi-Fi инфраструктура или интернет връзка, но пък като недостатък потребителското устройство губи връзка с интернет, докато е свързано към ESP32. Уеб сървърът използва WiFiServer библиотеката на Arduino ESP32 и протокол HTTP. Сървърът слуша на порт 80. Комуникацията между HTML и JavaScript се осъществява основно чрез манипулиране на Document Object Model (DOM). Когато уеб браузърът зареди index.html, той конструира DOM – дървовидна структура в паметта, представяща всички

HTML елементи. JavaScript файлът script.js, получава достъп до този DOM и може да взаимодейства с неговите елементи.

Изпълнението на JavaScript кода започва едва след като събитието DOMContentLoaded е задействано. Това гарантира, че целият HTML е анализиран и DOM е напълно изграден, преди JavaScript да се опита да взаимодейства с него. В рамките на функцията, асоциирана с това събитие, JavaScript използва метода document.getElementById() за да "хване" специфични HTML елементи, идентифицирани чрез техните уникални id атрибути, като например елементите, предназначени да показват температурата () или предупреждения (<p id="tempWarning">). Веднъж "хванати", тези DOM елементи се съхраняват в JavaScript променливи, което позволява последващата им манипулация.

JavaScript динамично променя съдържанието и външния вид на тези HTML елементи. Например, във функции като updateTemperatureUI (Фиг. 22.), свойството .textContent на съответния DOM елемент се използва за задаване на новата температурна стойност, получена от ESP32. По същия начин, свойството .style (например .style.backgroundColor или .style.color) се използва за промяна на CSS стиловете на елементите, за да се осигури визуална обратна връзка – например, фонът на температурния дисплей може да се промени в зависимост от това дали температурата е ниска, нормална или висока. Всички тези промени, направени от JavaScript върху DOM обектите, се отразяват незабавно от браузъра, така че потребителят вижда актуализираната информация и визуални индикации на уеб страницата.

За извличане на данни от ESP32 сървъра и изпращане на команди, JavaScript използва функцията fetch(). Тези HTTP заявки към пътища като /data (за получаване на всички сензорни данни). Това означава, че JavaScript не блокира изпълнението на останалия код, докато чака отговор от сървъра. След получаване на данните, JavaScript извиква съответните UI обновяващи функции (updateTemperatureUI, updateHumidityUI и т.н.), за да покаже новата информация на HTML страницата. Функцията setInterval(getAllSensorData, 5000) осигурява периодично извикване на getAllSensorData на всеки пет секунди, което поддържа данните на уеб страницата сравнително актуални, без да е необходимо потребителят ръчно да презарежда страницата. По този начин се постига динамичен и интерактивен потребителски интерфейс, където HTML осигурява структурата, а JavaScript управлява данните, взаимодействията и динамичните актуализации.

1.4. Проектиране на потребителския интерфейс

Проектирането на потребителския интерфейс за системата е насочен към осигуряване на ясна, интуитивна и леснодостъпна визуализация на данните от сензорите. Интерфейсът е реализиран като уеб страница, генерирана и обслужвана директно от ESP32 микроконтролера, което елиминира нуждата от външни сървъри или специализирани мобилни приложения и осигурява локален достъп.

Структурата на потребителския интерфейс е дефинирана в index.html файла. Страницата е проектирана да бъде адаптивна (responsive) чрез използването на meta тага viewport, което позволява коректното й изобразяване на различни размери екрани, включително мобилни устройства. Основното съдържание е организирано в контейнер ( ), който центрира елементите и подобрява визуалната организация. Включено е лого на проекта (logo.webp) и заглавие "GREENIT Контролен Панел", които ясно идентифицират предназначението на интерфейса.

Основната част от интерфейса е посветена на визуализацията на данните от сензорите, групирани в секция с клас sensor-data. Всеки параметър – температура, влажност на въздуха, осветеност, влажност на почвата и ниво на водата в резервоара – е представен с параграф (<p>), наименованието на параметъра и елемент с уникален id (напр.), предназначен за динамично обновяване на стойността чрез JavaScript.

Стилизирането на интерфейса се осъществява чрез външен CSS файл (style.css). Дизайнът цели да бъде чист и функционален, с използване на фоново изображение, свързано с тематиката на растенията (backgroundPlants.avif), и полупрозрачен фон за основния контейнер, за да се подобри четимостта. Цветовата схема е избрана така, че да бъде приятна за окото, като заглавията и ключовите елементи са акцентирани. Елементите, показващи стойностите от сензорите, са стилизирани така, че да променят цвета на фона и текста си в зависимост от измерените стойности (например, различни цветове за ниска, нормална и висока температура или влажност), осигурявайки бърза визуална обратна връзка на потребителя.

```
function updateTemperatureUI(tempStr) { // Очакваме стринг от JSON
    if (temperatureElem && tempIconElem && tempWarningElem) { // Проверка дали елементите съществуват
        if (tempStr !== null && tempStr !== undefined && tempStr !== "Error") {
            const temp = parseFloat(tempStr);
            temperatureElem.textContent = temp.toFixed(1); // ESP32 изпраща с 1 знак
            temperatureElem.style.color = '#333';          // Нулиране на цвета на текста
            temperatureElem.style.backgroundColor = 'transparent'; // Нулиране на фона
            tempIconElem.className = '';                   // Нулиране на иконите
            tempWarningElem.textContent = '';              // Нулиране на предупреждението

            if (temp < 15) {
                temperatureElem.style.backgroundColor = '#ADD8E6'; // Светло синьо за студено
                temperatureElem.style.color = '#0000CD';       // Тъмно син текст
                tempIconElem.className = 'fas fa-snowflake';
                tempWarningElem.textContent = 'Внимание: Ниска температура!';
            } else if (temp >= 15 && temp <= 30) {
                temperatureElem.style.backgroundColor = '#90EE90'; // Светло зелено за нормално
                temperatureElem.style.color = '#006400';       // Тъмно зелен текст
                // Няма специфична икона за нормална температура, освен ако не искаш
            } else if (temp > 30 && temp <= 40) {
                temperatureElem.style.backgroundColor = '#FFFFE0'; // Светло жълто за топло
                temperatureElem.style.color = '#B8860B';       // Тъмно жълт/кафяв текст
                tempIconElem.className = 'fas fa-exclamation-triangle'; // Икона за внимание
                tempWarningElem.textContent = 'Внимание: Висока температура!';
            } else if (temp > 40) {
                temperatureElem.style.backgroundColor = '#FFCCCB'; // Светло червено за много горещо
                temperatureElem.style.color = '#A52A2A';       // Кафяво-червен текст
                tempIconElem.className = 'fas fa-fire-alt';    // Или fa-fire за Font Awesome 6
                tempWarningElem.textContent = 'ОПАСНОСТ: Много висока температура!';
            }
```

Фиг. 22. Реализирана функция в script.js

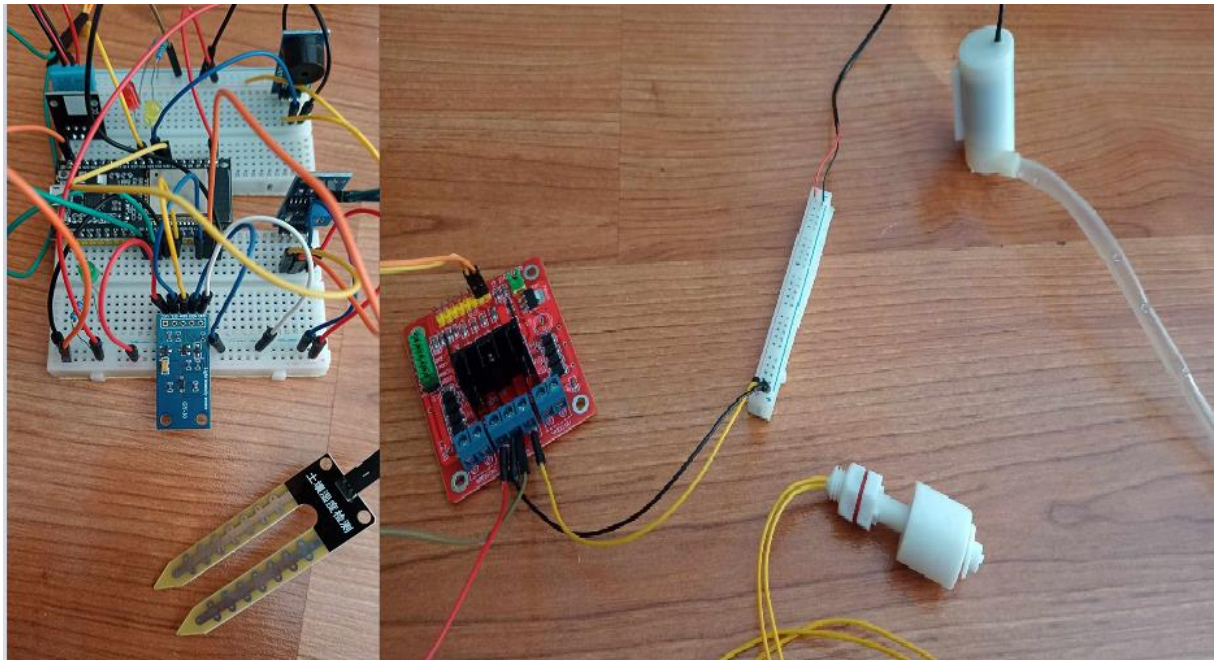
V. РЕАЛИЗАЦИЯ И ИМПЛЕМЕНТАЦИЯ

Този раздел описва практическите стъпки, предприети за изграждане на функционалния прототип на системата за мониторинг и автоматизирано поливане. Описани са процесът на сглобяване на хардуера и процесът на разработка и имплементация на софтуера (firmware) за ESP32 микроконтролера.

1. Сглобяване на хардуерния прототип

Сглобяването на хардуера е извършено на етап прототипиране, използвайки беззапойна прототипна платка (breadboard) за лесно свързване и тестване на компонентите. Развойната платка ESP32 е поставена централно на прототипната платка. Сензорът DHT11 е свързан към захранване (+3.3V/5V), земя (GND) и сигнален пин към GPIO 26 на ESP32. Сензорът BH1750 (на модулна платка) е свързан към захранване (+3.3V), земя (GND), SDA пин към GPIO 21 и SCL пин към GPIO 22 на ESP32. Резистивният сензор за влажност на почвата е свързан към захранване (+3.3V/5V), земя (GND) и аналоговия му изходен пин към GPIO 32 на ESP32. Драйверната схема L298N е свързана към ESP32 с пинове IN1 и ENA

към съответните GPIO 18 и 19. Логическото захранване на L298N (VCC и GND) е свързано към 5V (Vin) и GND на ESP32. Захранването за мотора (VMS и GND) е свързано към отделен 5V източник (с обща земя). Водната помпа е свързана към изходните клеми OUT1 и OUT2 на L298N. Пасивният зумер е свързан с единия извод към GPIO 33, а другия към GND. Светодиодите (LEDs) са свързани с анодите си (по-дългия крак) през токоограничаващи резистори (напр. 220 Ω) към съответните GPIO пинове (2, 27, 25), а катодите им (по-късия крак) са свързани към GND. ESP32 е захранен през USB. Отделното захранване за L298N VMS е включено в контактната мрежа. Всички GND връзки са проверени за непрекъснатост.



Фиг. 23. Моментно състояние на проекта (към 10.05.2025)

1.1. Разработка на софтуера

Разработката на софтуера е извършена в средата Arduino IDE, използвайки езика C++ и Arduino Framework за ESP32. Кодът е структуриран около стандартните setup() и loop() функции на Arduino. В началото на файла са дефинирани константи за номерата на пиновете, към които са свързани сензорите и драйверната схема. Дефинирани са глобални променливи за съхранение на прочетените стойности от сензорите (temperature, humidity, lightIntensity, soilMoisture) и състоянието на зумера (buzzerOn). Включени са необходимите заглавни файлове (#include) за използваните библиотеки (WiFi, DHT, Wire, BH1750, и др.). Създадени са инстанции на обектите за DHT сензора и BH1750 сензора. Дефинирани са SSID и парола за AP режима и е създаден обект WiFiServer.

Във функция setup() се инициализира се серийната комуникация (Serial.begin(9600)), също и сензорите (dht.begin(), Wire.begin(), lightMeter.begin()). Конфигурират се режимите на GPIO пиновете с pinMode() (OUTPUT за LEDs, зумер, L298N; INPUT или INPUT_PULLUP за сензора за ниво). Стартира се Wi-Fi в режим AP с WiFi.softAP(ssid, password). Извежда се IP адресът на AP в серийния монитор (Serial.println(WiFi.softAPIP())). Стартира се уеб сървърът (server.begin()). Задават се първоначални състояния на изходите (напр. помпата и LEDs са изключени).

Във функция loop() се извикват се функциите за четене от всеки сензор и резултатите се записват в глобалните променливи. Добавена е проверка isNaN() за DHT сензора. Имплементирана е логиката за управление на LEDs и зумера според дефинираните прагове за температура и светлина. Променливата buzzerOn се използва за управление на състоянието на зумера с tone() и noTone(). Имплементирана е базова логика за автоматично поливане, проверяваща стойността от почвения сензор и състоянието на сензора за ниво. Извиква се server.available() за проверка на нови клиенти. По – долу е представен кода за комуникацията на HTML и JavaScript през микроконтролера.

```
    // Routes за статични файлове
    server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
        if (SPIFFS.exists("/index.html")) request->send(SPIFFS, "/index.html",
"text/html");
        else request->send(404, "text/plain", "index.html not found");
    });
    server.on("/style.css", HTTP_GET, [](AsyncWebServerRequest *request) {
        if (SPIFFS.exists("/style.css")) request->send(SPIFFS, "/style.css",
"text/css");
        else request->send(404, "text/plain", "style.css not found");
    });
    server.on("/script.js", HTTP_GET, [](AsyncWebServerRequest *request) {
        if (SPIFFS.exists("/script.js")) request->send(SPIFFS, "/script.js",
"text/javascript");
        else request->send(404, "text/plain", "script.js not found");
    });
    server.on("/logo.webp", HTTP_GET, [](AsyncWebServerRequest *request){
        if (SPIFFS.exists("/logo.webp")) request->send(SPIFFS, "/logo.webp",
"image/webp");
        else request->send(404, "text/plain", "logo.webp not found");
    });
    server.on("/backgroundPlants.avif", HTTP_GET, [](AsyncWebServerRequest
*request){
         if (SPIFFS.exists("/backgroundPlants.avif")) request->send(SPIFFS,
"/backgroundPlants.avif", "image/avif");
```

```
        else request->send(404, "text/plain", "backgroundPlants.avif not found");
    });

    // Route за данни от сензори
    server.on("/data", HTTP_GET, [](AsyncWebServerRequest *request) {

        StaticJsonDocument<320> jsonDocument;

        if (isnan(temperature)) {
            jsonDocument["temperature"] = nullptr;
        } else {
            jsonDocument["temperature"] = String(temperature, 1); // Един знак
след десетичната запетая
        }

        if (isnan(humidity)) {
            jsonDocument["humidity"] = nullptr;
        } else {
            jsonDocument["humidity"] = String(humidity, 1);
        }

        if (lightIntensity < 0) { // Проверка за валидна стойност от BH1750
            jsonDocument["lightIntensity"] = nullptr;
        } else {
            jsonDocument["lightIntensity"] = String(lightIntensity, 0);
        }

        jsonDocument["soilMoisture"] = soilMoisture; // Суровата стойност от
почвения сензор
        jsonDocument["waterLevelPercent"] = mapSensorValue(val); // Мапнатата
стойност (0-100%)
        jsonDocument["waterLevelRaw"] = val;  // Суровата аналогова стойност

        String jsonString;
        serializeJson(jsonDocument, jsonString);
        request->send(200, "application/json", jsonString);
    });

    server.begin();
    Serial.println("HTTP server started");
}


void managePump() {
     if (soilMoisture > SOIL_DRY_THRESHOLD) {
          Serial.println("Pump ON");
          digitalWrite(IN1_PIN, HIGH);
          digitalWrite(ENA_PIN, HIGH);
```

```
            delay(3000);
            digitalWrite(IN1_PIN, LOW);
            analogWrite(ENA_PIN, LOW);
            Serial.println("Pump OFF");
            delay(3000);
        }
}
```

1.2. Първоначална конфигурация и настройка

Кодът е компилиран и качен на ESP32 платката чрез Arduino IDE, като предварително е избран правилният борд (ESP32 Dev Module) и COM порт. След качването, е отворен серийният монитор в Arduino IDE, за да се наблюдават дебъг съобщенията, включително IP адреса на създадената Access Point мрежа. От лаптоп/смартфон е осъществено свързване към Wi-Fi мрежата, създадена от ESP32 (със SSID и парола, дефинирани в кода). След това в уеб браузър е отворен IP адресът на ESP32 (обикновено 192.168.4.1). Проверено е дали уеб страницата се зарежда коректно и показва актуални стойности от сензорите (Фиг. 24.).

Фиг. 24. Уеб интерфейс

VI. ТЕСТВАНЕ И ОЦЕНКА НА РЕЗУЛТАТИТЕ

Този раздел описва процеса на тестване на разработения прототип на системата за мониторинг и автоматизирано поливане. Представена е методологията на тестване, дефинирани са конкретни тестови сценарии и случаи, и са представени и анализирани събраните експериментални данни с цел оценка на съответствието на системата с функционалните и нефункционалните изисквания.

1. Методология на тестване

За да се гарантира правилната работа както на отделните компоненти, така и на цялостната интегрирана система, тестването е проведено на няколко нива. Първоначално се извършва тестване на компоненти, известно още като Unit Testing, което включва проверка на функционалността на всеки сензор и актуатор поотделно. Това се осъществи чрез използване на прости тестови програми, написани в средата Arduino IDE, преди тези компоненти да бъдат интегрирани в основната система. Следващият етап е интеграционното тестване, което има за цел да провери взаимодействието между различните компоненти след сглобяването на прототипа. Тук се следи например дали данните от сензорите се четат коректно от основния програмен код и дали командите, изпращани към L298N драйвера, управляват водната помпа правилно. Накрая, е проведено системно тестване на цялостната функционалност на системата, съпоставяйки я с дефинираните функционални и нефункционални изисквания, включително проверката на работата на уеб интерфейса, механизмите за автоматизация и системите за сигнализация.

Всички тестови опити са проведени в домашни условия. За симулиране на различни експлоатационни условия са използвани разнообразни подходи: чаша с вода и източник на топлина или студ са използвани за тестване на DHT11 сензора; различни източници на светлина, като лампа и пряка слънчева светлина, служат за тестване на BH1750 сензора; почва с различна степен на влажност – суха, умерено влажна и напълно мокра – е използвана за оценка на резистивния сензор; а резервоар с вода, в който нивото се променя ръчно, позволи тестването на цифровия сензор и работата на помпата. Като инструмент за наблюдение по време на тестовете, серийният монитор на Arduino IDE е използван за следене на дебъг съобщения и сурови данни от сензорите, докато уеб интерфейсът е достъпван от лаптоп и смартфон за проверка на визуализацията на данните и функционалността на контролните елементи.

1.1. Тестови сценарии и случаи

За целите на проверката на ключовите изисквания са дефинирани конкретни тестови сценарии и случаи. При тестването на сензорите, DHT11 сензорът е излаган на известни температури и нива на влажност, като показанията му бяха сравнявани с тези на референтен термометър. Наблюденията, както в серийния монитор, така и в уеб интерфейса, показват, че стойностите са в рамките на очакваната точност от ±2°C за температура и ±5% за влажност, като данните се обновяваха коректно. Сензорът BH1750, изложен на различни нива на осветеност – от тъмнина до пряка светлина – демонстрира адекватно вариране на стойностите в луксове. Резистивният сензор, поставян в почва с различна влажност, показва значителна промяна в аналоговите стойности, които корелираха с очакваната влажност, като по-високи стойности съответстваха на по-суха почва. Цифровият сензор, при ръчно преместване в резервоар с вода, отчита коректно наличието или липсата на вода.

Тестването на уеб интерфейса и контролните му функции показа, че свързването към Wi-Fi Access Point мрежата, създадена от ESP32, е успешно от различни устройства като лаптоп и смартфон. Уеб страницата се зарежда в браузъра при въвеждане на IP адреса 192.168.4.1 за по-малко от пет секунди. Всички измерени параметри се показват коректно на страницата и се обновяват, като данните съответстват на тези от серийния монитор с незначително закъснение.

При тестването на автоматизираното поливане, поставянето на почвения сензор в суха почва, при наличие на вода в резервоара, води до автоматично включване на помпата за зададения период от три секунди. Когато обаче в резервоара няма вода, дори и почвата да е суха, помпата не се включва, демонстрирайки коректната работа на защитния механизъм.

Тестовете на сигнализацията потвърждават, че светодиодите (LEDs) се задействат според дефинираната логика при промяна на температурата или светлината около зададените прагове, както и при стартиране на сървъра. Зумерът също издава звук според програмираните условия.

2. Постигнати резултати

Въз основа на проведените тестове и събраните резултати може да се направи следната оценка - всички дефинирани функционални изисквания са изпълнени от разработения прототип. Системата успешно измерва параметрите, предоставя уеб интерфейс, управлява помпата ръчно и автоматично, има защита за нивото на водата и

осигурява сигнализация. Изискването за достъпност е изпълнено, тъй като са използвани евтини и достъпни компоненти, също са изпълнени и следните нефункционални изисквания: уеб интерфейсът е базов, но функционален и лесен за разбиране. Свързването е стандартно през Wi-Fi, прототипът демонстрира добра стабилност. Необходими са по-дългосрочни тестове за пълна оценка, особено по отношение на деградацията на сензорите. Системата работи коректно при захранване на ESP32 през USB и отделно захранване за L298N. Разработеният прототип успешно покрива повечето от поставените изисквания. Функционалността е реализирана според спецификацията. Нефункционалните изисквания за достъпност, леснота на ползване, време за реакция, захранване и локална работа са изпълнени.

2.1. Сравнение с аналогични системи

Спрямо комерсиални продукти системата е значително по-евтина и по-гъвкава (отворен код и хардуер). Ключово предимство е локалният контрол без зависимост от облак, което липсва при много комерсиални решения (Gardena, Netro, Edyn). Функционалността (брой сензори, автоматизация) е сравнима с някои продукти (Parrot Pot), но без полирания дизайн и мобилно приложение. Спрямо Mi Flora, системата предлага автоматизация и локален интерфейс.

2.2. Идентифицирани проблеми и ограничения на системата

По време на разработката и тестването са идентифицирани следните ограничения и потенциални проблеми: DHT11 има леки забележки относно точността, а резистивният сензор дава относителни показания и се нуждае от калибриране за различни типове почва. Също продължителната работа на резистивния сензор във влажна среда може да доведе до деградация на защитното му покритие. Прототипът е на breadboard и не е защитен от външни влияния (вода, прах). Липсва специализиран корпус. Системата не е оптимизирана за работа на батерии; ESP32 и Wi-Fi консумират значителна енергия.

2.3. Принос на разработката и бъдещи подобрения

Приносът на тази дипломна работа се състои в проектирането и реализацията на цялостна, интегрирана система за мониторинг на пет ключови параметъра и автоматизирано поливане, която използва достъпни компоненти, което я прави рентабилно решение. Важен аспект на приноса е имплементацията на локален уеб интерфейс, работещ в режим Wi-Fi Access Point, което осигурява на потребителите независимост от интернет и облачни услуги,

като същевременно повишава леснотата на използване и гарантира поверителността на данните. Освен това, разработката успешно демонстрира практическата приложимост на комбинацията от микроконтролер ESP32, разнообразни сензори и уеб технологии за създаването на полезно и достъпно решение в сферата на домашната автоматизация. Не на последно място, извършеното систематизиране на информация за съществуващи решения допринася за идентифицирането на специфична ниша за подобрени локални системи, насочени към грижата за растенията, което отваря възможности за бъдещи разработки и усъвършенствания. За бъдеща разработка се предвижда внедряването на MQTT (Message Queuing Telemetry Transport) протокола, което би трансформирало системата от чисто локално решение към компонент на по-голяма IoT екосистема. MQTT е лек протокол за съобщения тип "публикуване/абониране" (publish/subscribe), идеален за устройства с ограничени ресурси като ESP32. Предимствата на този подход са многобройни. Например, той позволява данните от системата да бъдат достъпни за множество други приложения и услуги. Това отваря врати за интеграция със системи за домашна автоматизация като Home Assistant, OpenHAB или Node-RED, където могат да се създават по-сложни правила за автоматизация, да се съхраняват исторически данни и да се генерират нотификации. Също се обмисля и разработка на специализирано мобилно приложение за подобрено потребителско изживяване. Мобилното приложение би могло да се свързва към MQTT сървъра (ако е имплементиран) или директно към ESP32 (например чрез Wi-Fi в локалната мрежа, ако ESP32 работи в режим Station, или дори чрез Bluetooth Low Energy за директна връзка). Приложението ще визуализира данните от сензорите по атрактивен и интуитивен начин, използвайки графики за показване на исторически тенденции. Освен визуализация, мобилното приложение ще предоставя удобни контроли за ръчно управление на помпата и други потенциални актуатори. Една от най-ценните функции би била възможността за получаване на push нотификации. Например, приложението може да известява потребителя при критично ниско ниво на водата в резервоара, при екстремни температури, или когато влажността на почвата падне под определен праг и растението се нуждае от поливане. Допълнително, мобилното приложение може да позволи на потребителя да конфигурира параметрите на системата, като например праговете за автоматично поливане, продължителността на поливането, или дори да създава персонализирани профили за грижа за различни видове растения. Може да се интегрира и база данни с информация за нуждите на популярни стайни растения, предоставяйки съвети и препоръки.

VII. ЗАКЛЮЧЕНИЕ

В рамките на настоящата дипломна работа е успешно проектирана, реализирана и тествана IoT базирана вградена система за мониторинг на ключови параметри (температура и влажност на въздуха, влажност на почвата, интензитет на светлина, ниво на вода в резервоар) и автоматизирано поливане на стайни растения. Системата използва микроконтролер ESP32, набор от сензори, и предоставя интуитивен локален уеб интерфейс за визуализация и контрол, достъпен през Wi-Fi в режим Access Point. Извършен е теоретичен обзор и анализ на съществуващи решения, дефинирани са изисквания, проектирани са хардуерната и софтуерната архитектура, сглобен е хардуерен прототип и е разработено програмното осигуряване. Проведените тестове потвърждават функционалността и стабилността на системата в домашни условия. Разработеният прототип успешно демонстрира концепцията за достъпна, локално управляема система за интелигентна грижа за растенията, която интегрира мониторинг на множество параметри и автоматизация. Използването на ESP32 с вграден Wi-Fi модул и работата в режим Access Point се оказва ефективно решение за осигуряване на лесен локален достъп без зависимост от външна инфраструктура или услуги. Комбинацията от избраните сензори, макар и с известни компромиси в точността (DHT11), предоставя достатъчно информация за базовите нужди на домашния потребител. Локалният уеб интерфейс, макар и базов, е функционален и отговаря на изискването за леснота на използване.

Тази разработка не представлява завършен комерсиален продукт, а по-скоро солидна основа, върху която могат да се надградят множество съществени подобрения и разширения. Потенциалът за развитие е значителен и включва както усъвършенстване на текущата функционалност, така и внедряване на изцяло нови възможности. Подобрения в енергийната ефективност и разработването на компактен и естетически издържан корпус ще повишт практическата приложимост на системата.

Дипломната работа успешно адресира реална потребност и демонстрира как съвременните вградени системи и IoT технологии могат да бъдат приложени за създаване на интелигентни, достъпни и полезни решения, които улесняват ежедневието и насърчават по-близката връзка с природата в домашни условия.

VIII. ИЗПОЛЗВАНА ЛИТЕРАТУРА

1. Stoyanov, Stanimir & Popchev, Ivan. (2017). Интернет на нещата. 3(37).

2. Фархи, Овид & Николов, Емил. (2008). ВГРАДЕНИ ТЕХНОЛОГИИ И СИСТЕМИ ЗА УПРАВЛЕНИЕ (EMBEDDED SYSTEMS) - ОБЗОР (7-18)

3. Maini, Anil. (2007). Microcontrollers. 10.1002/9780470510520.ch14.

4. Espressif Systems, "ESP32 Series Datasheet Version 4.9" https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf [Достъпено на: 25.05.2025].

5. Arduino, “Arduino Uno R3” (май 2025) https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf [Достъпено на: 26.05.2025]

6. Raspberry Pi, “Raspberry Pi Model B” (март 2024) https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf [Достъпено на: 26.05.2025]

7. Hrisko, Joshua. (2020). Capacitive Soil Moisture Sensor Theory, Calibration, and Testing. 10.13140/RG.2.2.36214.83522.

8. Mouser Electronics, “DHT11 Humidity & Temperature Sensor” https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf?srsltid=AfmBOooqryYwge74RM3tBsMhMGngqPDwkl1uD1qdHaBlHmjAgK7ubkuJ [Достъпено на: 28.05.2025]

9. Handson Technology, “L298N Dual H-Bridge Motor Driver” http://handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf [Достъпено на 26.05.2025]

10. ROHM Semiconducor, “Digital 16bit Serial Output Type Ambient Light Sensor IC” https://www.mouser.com/datasheet/2/348/bh1750fvi-e-186247.pdf?srsltid=AfmBOopirhUGyZQ-dgPbFy_WEPt0pWO0E8QoUcEUqXnvADO2pPfOOIL4 [Достъпено на: 28.05.2025]

11. Daud, Sharipah & Hairul, Muhammad & Abdul Rahman, Rahimah. (2025). IoT-BASED SMART AGRICULTURE MONITORING SYSTEM.