Пловдивски Университет “Паисий Хилендарски”

Факултет по Математика и Информатика

Катедра “Компютърна Информатика”

Дипломна работа

за придобиване на образователно-квалификационна степен „**бакалавър**”

на Валентин Иванов Романов, фак. № **1601321027**,

специалност „Софтуерно инженерство”

редовна форма на обучение

на тема

Разработка на система за смесване на течности

Научен ръководител: докт-т. Михаил Петров

Пловдив 2020 г.

Увод

Въведение в проблематиката

Днес селско стопанство не може без модерни технологии. Фермерите, които не използват най-новите постижения във ветеринарната наука в областта на храненето на животните, рискуват да останат без печалба и да загубят в борбата за оцеляване на пазара.

Минералните добавки са неразделна част от диетата на говеда, птици и други селскостопански животни. Минерален комплекс прибавя към водата за пиене на животните, има благоприятен ефект върху бактериите на стомашно-чревния тракт. Подобрено храносмилане, насърчава ефективното хранене на животните и води до увеличаване на ежедневната тежест на животното. Той подобрява общото състояние на животните и осигурява устойчивостта на животните към всички видове болести. Предотвратява образуването на опасни бактерии. Поддържа баланса на течности в тялото.

Цели и задачи на дипломна работа

Целта на проекта е создаване на контролера кой да осигури автоматичен подход към храненето с минерали, елиминирайки ръчния труд и човешките грешки.

От поставената цел произтичат следните подзадачи:

* Да се проучи съществуващи решения на пазара
* Да се избира технологии за реализация на проекта
* Да се разработи клиентска и серверна част на приложение
* Да се разработи вграден софтуер за контроллер

Структура на дипломната работа

Дипломната работа, описва както теоретичните основи на проблемна област така и реализацията на протопипната система.

В Глава първа – Съществуващи решения, се описват основни съществуващи решения на пазара. Целта на първа глава е да разберем необходимостта от контролер и неговата конкурентоспособност.

В Глава втора – Хардуерна част на контролера, да се разгледа електромеханичната и електронната част на контролера

В Глава трета – Софтуерна част на контролера, да се разгледа основните технологии, използвани при разработването

В Глава четвърта – Прототип на приложение за контроллер

Съществуващи решения

A close up of a mans face

Description automatically generated

Фигура 1 Контекстна диаграма

Контролерът се състои от няколко части.

Електромеханична част (Фигура 2):

* Спирателен вентил за отваряне / затваряне на водопровода
* Сензор за воден поток
* Помпи за подаване на минерални добавки от контейнери към вода

Електронна част (Фигура 2):

* Arduino за електро-механичен контрол и трансфер на данни към / от облака
* ESP8266 за свързване с интернет чрез WiFi

A close up of text on a white background

Description automatically generated

Фигура 2 Електромеханична и електронна част

Софтуерна част:

* Програма Arduino
* Приложение за управление и получаване на отчети за работата на контролера за мобилни телефони
* Уеб страница за управление и получаване на отчети за ефективността на контролера

За осигуряване функционирането на контролера се използват редица услуги на трети страни.

Firebase Auth е услуга, която може да удостоверява потребителите, използвайки само клиентски код. Той поддържа доставчици на социални влизания Facebook, GitHub, Twitter и Google (и Google Play Games). В допълнение, тя включва система за управление на потребители, която позволява на разработчиците да активират удостоверяване на потребителя, използвайки имейл за вход и парола, съхранени във Firebase.

Firebase предоставя база данни в реално време и бекенд като услуга. Тази услуга предоставя на разработчиците на приложения API, който ви позволява да синхронизирате данните на приложенията между клиенти и да ги съхранявате в облака на Firebase. Компанията предоставя и клиентски библиотеки, които позволяват интеграция с приложения за Android, iOS, JavaScript / Node.js, Java, Objective-C, Swift. Базата данни е достъпна и чрез REST API и обвързване с множество JavaScript скриптове, като AngularJS, React, Ember.js и Backbone.js. API REST използва дневник на събития със сървър, който е интерфейс за създаване на HTTP връзки за получаване на push съобщения от сървъра. Разработчиците, които използват база данни в реално време, могат да защитят данните си с правила за сигурност, приложени на сървъра.

Netlify е хостинг и без сървърна back-end услуга за статични сайтове.

* Атомно внедряване с незабавно публикуване и връщане назад. Разполаганията са неизменни. Тоест, всяко внедряване води до версия на сайта, която никога не се променя. Последващите актуализации създават нови копия на сайта, за да заменят предишни версии (които са тихо благодарни за тяхната поддръжка и се оттеглят, но не се унищожават). Това означава, че можете да се върнете към всяка предишна компилация по всяко време с едно щракване на бутон в администраторската конзола или чрез API.
* Известия и постоянни връзки. Netlify ви позволява да персонализирате известия въз основа на различни събития за внедряване. Можете да определите кой ще бъде уведомен за събития като началото на разполагане или когато успешното разполагане се провали, бъде блокирано или деблокирано.
* Разгръщане на клонове и поддомейни. Netlify ви дава известен контрол върху вашето внедряване. Можете да изберете дали да разгърнете само производствения си клон, всички клонове или няколко имена на клонове.
* A / B тестване, многовариантно тестване или разделно тестване. Хората се отнасят до A / B тестване по няколко термина, но Netlify го нарича „разделно тестване“, защото прави точно това: разделя трафика към вашия сайт между всички филиали, които посочите. Можете да разделите трафика си на толкова клонове, колкото искате, като посочите какъв процент от трафика ви трябва да отиде там.
* Команди за сглобяване на контекст. Не само можете да внедрите различни клонове, но също така можете да персонализирате съдържанието и средите във вашите внедрявания, за да отговарят на различни контексти като подреждане, тестване и производство.
* SSL управление и безплатен SSL от Let’s Encrypt. Netlify улеснява настройването на HTTPS на персонализирани домейни. Имате възможност за управляван SSL, персонализиран SSL и дори специален IP SSL за тези бизнеси, които го изискват.
* Извършване на тестове с непрекъсната интеграция на Netlify. Едно от нещата, които Netlify дава е, че освен специално създаден CDN за хостване на вашите сайтове, той също така осигурява контейнеризирана среда за изпълнение на вашите компилации. Това означава, че всяка компилация, която стартирате във вашата локална среда за компилация или на сървър за непрекъсната интеграция, може да се изпълнява директно в Netlify.
* Обработка на формуляри. Ако имате нужда от формуляр на вашия сайт, който приема съдържание, изпратено от вашите посетители, Netlify може да го хоства вместо вас. Чрез просто добавяне на атрибут към HTML на вашия формуляр, Netlify ще създаде подходяща крайна точка за формуляра и ще направи всички данни достъпни за вас чрез администраторския интерфейс и API.
* Пренасочване, пренаписване и прокси. Не оставяйте URL адреса зад себе си! \_redirects file \_redirects в разположената директория, можете да получите достъп до разширени опции за конфигуриране за пренасочване и пренаписване. Това се случва в крайните възли на CDN, което ги прави невероятно бързи и ефективни. Също така имате възможност да посочите HTTP кода за отговор във файла \_redirects, създавайки няколко други неща като персонализирани 404s и дори проксиране на други ресурси.
* Персонализиран контрол на заглавката. Това е за всеки, който е хоствал своя сайт на Github Pages и е преследвал онзи перфектен показател Lighthouse или Page Speed ​​Insights. Правилно сте го направили, но трябва да контролирате своите HTTP заглавки за кеширане, за да получите последната част от оптимизацията на производителността, която ви липсва ... за съжаление нямате толкова мощност.

Arduino

Arduino е електронна платформа с отворен код, базирана на лесен за използване хардуер и софтуер. Arduino дъските могат да четат входове - светлина върху сензор, пръст върху бутон или съобщение в Twitter - и го превръщат в изход - активиране на двигател, включване на светодиод, публикуване на нещо онлайн. Можете да кажете на вашия съвет какво да прави, като изпратите набор от инструкции до микроконтролера на дъската. За целта използвате езика за програмиране на Arduino (базиран на Wiring) и Arduino Software (IDE), базиран на Processing.

През годините Arduino е мозъкът на хиляди проекти, от ежедневни обекти до сложни научни инструменти. Световна общност от производители - студенти, хобисти, художници, програмисти и професионалисти - се е събрала около тази платформа с отворен код, техните приноси са добавили към невероятно количество достъпни знания, които могат да бъдат от голяма полза както за начинаещи, така и за експерти.

Ардуино е роден в Ivrea Interaction Design Institute

като лесен инструмент за бързо създаване на прототипи, насочен към студенти без опит в електрониката и програмирането. Веднага след като достигна до по-широка общност, бордът на Arduino започна да се променя, за да се адаптира към новите нужди и предизвикателства, диференцирайки своето предложение от прости 8-битови дъски до продукти за IoT приложения, носими, 3D печат и вградени среди. Всички дъски на Arduino са напълно отворен код, даващи възможност на потребителите да ги изграждат независимо и в крайна сметка да ги адаптират към техните специфични нужди. Софтуерът също е с отворен код и се разраства чрез приноса на потребителите по целия свят.

Благодарение на своето просто и достъпно потребителско изживяване, Arduino е използван в хиляди различни проекти и приложения. Софтуерът Arduino е лесен за използване за начинаещи, но достатъчно гъвкав за напреднали потребители. Работи се на Mac, Windows и Linux. Учителите и студентите го използват за изграждане на нискотарифни научни инструменти, за доказване на принципите на химията и физиката или за започване на програмиране и роботика. Дизайнерите и архитектите изграждат интерактивни прототипи, музикантите и художниците го използват за инсталации и да експериментират с нови музикални инструменти. Създателите, разбира се, го използват за изграждането на много от проектите, изложени на Faire Maker. Arduino е ключов инструмент за научаване на нови неща. Всеки - деца, любители, артисти, програмисти - може да започне да се занимава само следвайки стъпка по стъпка инструкциите на комплект или да споделя идеи онлайн с други членове на общността на Arduino.

Има много други микроконтролери и микроконтролерни платформи, достъпни за физически изчисления. Parallax Basic Stamp, BX-24 на Netmedia, Phidgets, MIT's Handyboard и много други предлагат подобна функционалност. Всички тези инструменти вземат разхвърляните подробности за програмирането на микроконтролери и ги увиват в лесен за използване пакет. Arduino също опростява процеса на работа с микроконтролери, но предлага известно предимство за учители, студенти и заинтересовани любители пред други системи:

* Евтин - Arduino платките са сравнително евтини в сравнение с други платформи за микроконтролери. Най-малко скъпата версия на модула Arduino може да бъде сглобена на ръка и дори предварително сглобените Arduino модули струват по-малко от $ 50
* Крос-платформа - Софтуерът Arduino (IDE) работи на операционни системи Windows, Macintosh OSX и Linux. Повечето микроконтролери са ограничени до Windows.
* Проста, ясна програмна среда - Софтуерът Arduino (IDE) е лесен за използване за начинаещи, но достатъчно гъвкав и за напреднали потребители, за да се възползва от тях. За преподавателите удобно се основава на програмната среда за обработка, така че студентите, които се учат да програмират в тази среда, ще бъдат запознати с начина, по който работи Arduino IDE.
* С отворен код и разширим софтуер - Софтуерът Arduino се публикува като инструменти с отворен код, достъпни за разширение от опитни програмисти. Езикът може да бъде разширен чрез C++ библиотеки и хората, които искат да разберат техническите подробности, могат да направят скока от Arduino към езика за програмиране на AVR C, на който е базиран. По същия начин можете да добавите AVR-C код директно във вашите програми на Arduino, ако искате.
* Отворен код и разтегателен хардуер - Плановете на платките на Arduino са публикувани под лиценз Creative Commons, така че опитни дизайнери могат да направят своя собствена версия на модула, като го разширят и подобрят. Дори сравнително неопитни потребители могат да създадат версията на модула, за да разберат как работи и да спестят пари.

В проекта исползван Arduino Uno R3

A circuit board

Description automatically generated

Arduino Uno е микроконтролерна платка, базирана на ATmega328P. Той има 14 цифрови пина за вход / изход (от които 6 могат да се използват като PWM изходи), 6 аналогови входа, 16 MHz керамичен резонатор (CSTCE16M0V53-R0), USB връзка, захранващ жак, ICSP заглавка и бутон за нулиране.

Информационен лист

|  |  |
| --- | --- |
| Microcontroller | [ATmega328P](http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf) |
| Operating Voltage | 5V |
| Input Voltage (recommended) | 7-12V |
| Input Voltage (limit) | 6-20V |
| Digital I/O Pins | 14 (of which 6 provide PWM output) |
| PWM Digital I/O Pins | 6 |
| Analog Input Pins | 6 |
| DC Current per I/O Pin | 20 mA |
| DC Current for 3.3V Pin | 50 mA |
| Flash Memory | 32 KB (ATmega328P) of which 0.5 KB used by bootloader |
| SRAM | 2 KB (ATmega328P) |
| EEPROM | 1 KB (ATmega328P) |
| Clock Speed | 16 MHz |
| LED\_BUILTIN | 13 |
| Length | 68.6 mm |
| Width | 53.4 mm |
| Weight | 25 g |

Езикът за програмиране на устройството Arduino се основава на C / C ++ и е свързан с библиотеката AVR Libc и ви позволява да използвате която и да е от неговите функции. Езикът Arduino може да бъде разделен на четири раздела оператори, данни (променливи и константи), функции и библиотеки.

Основните оператори(функции):

setup() - извиква се, когато скицата започне. Използва се за инициализиране на променливи, определяне на режимите на работа на щифтове, стартиране на използваните библиотеки и т.н. Функцията за настройка работи само веднъж, след всяко включване или нулиране на платката Arduino.

loop() - прави точно това, което подсказва името му, и се върти наоколо, позволявайки на вашата програма да прави и реагира на изчисления.

Поддържат се всички контролни, аритметични, логически оператори на езика C. От основните разлики от C бих искал да отбележа специалните константи и входно / изходните функции.

Константи, характеризиращи нивото на напрежението на щифтовете, HIGH и LOW

Когато работите с цифрови щифтове, има само две стойности, които те могат да изведат или прочетат: HIGH и LOW.

HIGH (както се прилага към щифт) може да се различава леко в зависимост от това дали щифтът е конфигуриран като вход (INPUT) или като изход (OUTPUT). Ако щифтът е конфигуриран като INPUT с помощта на функцията pinMode, тогава при четене на данни от него (функция digitalRead), микроконтролерът ще отговори HIGH, когато щифтът е 3V или повече.

Също така е възможно щифтът да е конфигуриран като INPUT от функцията pinMode, след което функцията digitalWrite го задвижва високо. В този случай към щифта ще бъде свързан вътрешен 20K издърпващ резистор, което ще го направи ВИСОКО. При четене стойността HIGH ще се задържи, докато външните вериги на щифта генерират ниско ниско ниво. Ето как работи режимът INPUT\_PULLUP.

Ако pinMode е конфигуриран като OUTPUT и digitalWrite е HIGH, щифтът ще бъде настроен на 5V. В този режим може да бъде източник на ток и например да светне светодиод, свързан последователно през резистор към земя или към друг изход с НИСКО ниво.

LOW също има различни значения в зависимост от това дали изходът е конфигуриран като вход (INPUT) или изход (OUTPUT). Ако щифтът е конфигуриран като вход (INPUT) с помощта на функцията pinMode, тогава при четене на данни от него с функцията digitalRead, микроконтролерът ще реагира НИСКО, ако напрежението на щифта не надвишава 2V.

Ако щифтът е конфигуриран като ИЗХОД от функцията pinMode и функцията digitalWrite е LOW, щифтът ще бъде настроен на 0V. В този режим той може да приеме потъващия ток, например от светодиод, свързан чрез резистор до + 5V или към друг изход с ВИСОКО ниво.

Константи, характеризиращи цифрови щифтове, INPUT, INPUT\_PULLUP и OUTPUT

Щипки, конфигурирани като INPUT

Пиновете на Arduino (ATmega), конфигурирани с функция pinMode () като входове (INPUT), са в състояние на висок импеданс. Това е еквивалентно на свързване към щифта на резистор от серия 100 MΩ, така че има малко или никакво изискване за веригите, свързани към тези щифтове. Този режим е удобен за четене на сигнали от сензори, но не е приемлив за захранване на светодиоди.

Трябва да се отбележи, че входовете INPUT понякога са свързани към земята чрез издърпващ резистор (резистор към земята).

Щипки, конфигурирани като INPUT\_PULLUP

Микроконтролерът ATmega в Arduino има вътрешни издърпващи резистори (резистори, свързани към захранването вътре в микросхемата), които могат да бъдат контролирани. Ако предпочитате да ги използвате вместо външни резистори, свързани към земята, използвайте параметъра INPUT\_PULLUP във функцията pinMode (). Това ще обърне поведението на външния сензор, свързан към щифта: HIGH означава, че е деактивиран, а LOW означава, че е активиран. Вижте пример за използване на INPUT\_PULLUP със серийна комуникация.

Изходи, конфигурирани като OUTPUT

Пиновете, конфигурирани с pinMode () като OUTPUT, са в състояние на нисък импеданс. Това означава, че те могат да захранват външни вериги с относително висок ток. Микроконтролерът ATmega може да подава (положителен ток) или да приема (отрицателен) ток до 40 mA (милиампера) от външни устройства / вериги. Този режим е удобен за захранване на светодиоди, но безполезен при четене на сигнали от сензори. Щифтовете, конфигурирани като изход, също могат да бъдат повредени от късо съединение към земята или 5V. Освен това изходният ток на ATmega е недостатъчен за захранване на повечето релета и двигатели, изискващи допълнителни интерфейсни вериги.

pinMode () конфигурира поведението на посочения щифт като вход или изход.

digitalWrite () изпраща HIGH или LOW стойност към цифров щифт.

Ако функцията pinMode () е конфигурирана като изход (OUTPUT), тогава когато се изпълни функцията digitalWrite (), нейното напрежение ще бъде променено на съответната стойност: 5 V (или 3.3 V за платки, работещи от 3.3 V) при изпращане на HIGH, 0 V (земя ) - при НИСКО.

Ако щифтът е конфигуриран като INPUT, изпращането на digitalWrite () HIGH стойност ще свърже вътрешния 20K издърпващ резистор (вижте ръководството за цифрови щифтове). Записването на НИСКА стойност ще деактивира изтеглянето. Вътрешен издърпващ резистор може да осигури само слаба LED светлина. Следователно, ако светодиодът свети, но е много слаб, най-вероятната причина е издърпващият резистор. За да разрешите този проблем, трябва да поставите съответния щифт в изходния режим, като използвате функцията pinMode ().

digitalRead () отчита ВИСОКОТО или НИСКО ниво на сигнала от посочения цифров щифт.

analogReference (type) задава еталонното напрежение, използвано при четене на аналогов сигнал (с други думи, задава максималната стойност на входния диапазон). На разположение са следните стойности за избор на еталонно напрежение:

* DEFAULT: референтно напрежение по подразбиране от 5V (на 5V Arduino платки) или 3.3V (на 3.3V Arduino платки)
* INTERNAL: Вътрешно референтно напрежение, равно на 1,1 V в микроконтролерите ATmega168 и ATmega328, или 2,56 V в микроконтролера ATmega8 (не се предлага в Arduino Mega)
* INTERNAL1V1: Вътрешен референтен 1.1V (само за Arduino Mega)
* INTERNAL2V56: Вътрешен референтен 2.56V (само за Arduino Mega)
* EXTERNAL: Напрежението, приложено към щифта AREF, ще се използва като еталонно напрежение (0 до 5V)

analogRead () чете стойността на напрежението от посочения аналогов щифт. Arduino съдържа 6-канален (8-канален - в Mini и Nano, 16 - в Mega) 10-битов аналогово-цифров преобразувател, който преобразува входното напрежение от 0 - 5 V в целочислени стойности в диапазона от 0 до 1023, съответно ... Разделителната способност на ADC е: 5 V / 1024 стойности или 0,0049 V (4,9 mV) на стойност. Входният обхват и разделителната способност могат да се променят с помощта на функцията analogReference ().

Отчитането на стойност от аналогов вход отнема около 100 микросекунди (0,0001 s), така че максималната скорост на изписване на изхода е приблизително 10 000 пъти в секунда.

analogWrite () генерира определеното аналогово напрежение на ПИН като ШИМ сигнал. Може да се използва за промяна на яркостта на светодиода или контрол на скоростта на мотора. След извикване на analogWrite (), щифтът непрекъснато ще генерира ШИМ сигнал с посочения работен цикъл до следващото извикване на analogWrite () (или докато digitalRead () или digitalWrite () бъде извикан на същия пин). PWM честотата е приблизително 490 Hz.

На повечето платки Arduino (базирани на микроконтролери ATmega168 или ATmega328) функцията analogWrite () работи на щифтове 3, 5, 6, 9, 10 и 11. На Arduino Mega функцията работи на щифтове 2 до 13. На по-старите версии на Arduino (на базирана на микроконтролера ATmega8), функцията analogWrite () работи само на щифтове 9, 10 и 11.

Arduino Due поддържа analogWrite () за изводи 2 до 13 и за изводи DAC0 и DAC1. За разлика от PWM щифтовете, DAC0 и DAC1 са щифтове от цифрови към аналогови преобразуватели, така че когато е извикан analogWrite (), те се държат като нормални аналогови изходи.

При работа с analogWrite () не е необходимо предварително извикване на функцията pinMode (), за да превключите щифтовете в изходен режим.

AnalogWrite () няма нищо общо с аналоговите щифтове и analogRead ().

Проектът активно използва функцията map () за преобразуване на стойностите от датчика за дебита на водата в необходимото време за работа на водната помпа за подаване на минерални добавки.

map (value, fromLow, fromHigh, toLow, toHigh) преобразува стойността на променлива от един диапазон в друг. Тези. стойност отLow ще бъде преобразувана в toLow, а fromHigh ще бъде преобразувана вHigh. Всички междинни стойности се мащабират спрямо новия диапазон [toLow; toHigh].

Функцията не ограничава стойността на променлива до посочените граници, тъй като нейните стойности извън посочения диапазон понякога носят полезна информация. За да ограничите обхвата, трябва да използвате функцията constrain () преди или след функцията map ().

Имайте предвид, че долните граници на посочените диапазони (отLow, toLow) могат да бъдат числено по-големи от горните граници (отHigh, toHigh). В този случай функцията map () може да се използва за създаване на обратен диапазон от числа, например:

y = map (x, 1, 50, 50, 1);

Функцията може да обработва отрицателни числа, така че този пример

y = map (x, 1, 50, 50, -100);

също работи правилно.

Функцията map () използва изчисления с цяло число, така че не връща дробни стойности, както понякога се очаква. В този случай дробната част на числото просто се изхвърля, без закръгляване или изчисляване на средните стойности.

Сензорът за воден поток (дебит), както подсказва името, е устройство за измерване на водния поток. Сензорът за воден поток, използван в този проект, е показан на фигурата 3.

A picture containing indoor, table, food, cup

Description automatically generated

Фигура 3

Той има пластмасов корпус на клапана с ротор и верига на сензор на Хол. Сензорът има три проводника, а именно + 5V (червен), GND (черен) и изход (жълт). Тъй като работи на + 5V, той може да бъде свързан с всеки микроконтролер като Arduino. В горната част има етикет с важна информация за сензора. Информацията на този етикет е както следва:

Модел: YF-S201

Работен обхват: 1 - 30 л / мин

Водно налягане: по-малко от 1,75Mpa

Дъното на сензора е маркирано със стрелка, показваща желаната посока на воден поток през сензора. Поради това е важно, когато свързвате този сензор към тръбата, не забравяйте да следвате тази посока.

Сензорът за водния поток е изпълнен с помощта на сензор за ефект на Хол. По този начин работата на сензора за водния поток може лесно да бъде разбрана, ако сте запознати със сензора на Хол. По принцип сензорът за воден поток YF-S201 се състои от въртящо се колело (или турбинно колело), ​​което се върти, докато водата преминава през сензора. В центъра на това колело е прикрепен магнит.

Сега да преминем към сензора на Хол. Сензорът на ефекта на Хол, използван в този датчик за воден поток, е 460S IC в пакет TO-92.

Работата на сензора 460S на Хол е следната: когато южният полюс на магнита е насочен към микросхемата, изходът на сензора е нисък, а когато северният полюс на магнита е насочен към микросхемата, тогава изходът е висок. Имайки това предвид, когато водата тече през сензора за воден поток YF-S201, колелото на клапата се върти под силата на водата и в резултат на това магнитът, прикрепен към него, също ще се върти. В резултат на това магнитното поле в близост до сензора за ефект на Хол обръща полярността, докато колелото на клапата се върти и изходът на сензора (жълт проводник на изходния щифт) ще бъде импулсен. Чрез наблюдение на броя на импулсите от изхода на сензора за воден поток можете лесно да изчислите количеството вода, преминаващо през сензора и в резултат на това, дебита на водата.

A picture containing clock

Description automatically generated

Фигура 4 Пример за свързване на сензора

Според документацията на сензора за воден поток YF-S201, тъй като изходът на сензора е импулсен, чрез изчисляване на честотата на импулсите можем да изчислим обема на водата, преминаваща през сензора. Честотата на импулсите в Hz е 7,5 \* Скорост на потока (в литри в минута). По този начин количеството вода в литри на час = честота на импулсите \* 60 / 7,5.

По-долу е даден примерен код за изчисляване на дебита на водата.

const int watermeterPin = 2;

volatile int pulse\_frequency;

unsigned int literperhour;

unsigned long currentTime, loopTime;

byte sensorInterrupt = 0;

void setup()

{

pinMode(watermeterPin, INPUT);

Serial.begin(9600);

attachInterrupt(sensorInterrupt, getFlow, FALLING);

currentTime = millis();

loopTime = currentTime;

}

void loop ()

{

currentTime = millis();

if(currentTime >= (loopTime + 1000))

{

loopTime = currentTime;

literperhour = (pulse\_frequency \* 60 / 7.5);

pulse\_frequency = 0;

Serial.print(literperhour, DEC);

Serial.println(" Liter/hour");

}

}

void getFlow ()

{

pulse\_frequency++;

}

Воден насос

A close up of a device

Description automatically generated

спецификация:

материал на помпата: ABS

размер: 38x30x24 мм

работно напрежение: dc 12v

консумация на енергия: 3.6w

работен ток: 450ma

водоустойчиво ниво: IP68

най-високо повдигане: 350 сантиметра

максимален дебит: 350l / h

професионален живот: > 20 000 часа

децибел: < 35 децибела

работна температура: 0 - 75 ℃

Приблизителна схема на свързване:

A screen shot of a city

Description automatically generated

React Native

React Native е рамка за мобилни приложения с отворен код, създадена от Facebook. Използва се за разработване на приложения за Android, iOS, Web и UWP, като дава възможност на разработчиците да използват React заедно с възможностите на родната платформа.

React Native позволява на разработчиците да създават кросплатформени приложения, които изглеждат и се чувстват изцяло Native, тъй като използва JavaScript компоненти, които са изградени както на iOS, така и на Android компоненти. Това спестява много време и пари, тъй като разработчиците не трябва да създават едно и също приложение за множество платформи. Освен това е по-лесно да се поддържа приложение, създадено с React Native, тъй като има само една кодова база. React Native използва основни блокове за Android и iOS, за да компилира Native приложения и за двете платформи в JavaScript. Това прави работата с кодовата база по-лесна, а добавянето на нови функции също е опростено, дори докато приложението работи.

A picture containing monitor

Description automatically generated

Нека да разгледаме работния процес на React Native, както следва:

Първоначално UI нишката започва да зарежда JS пакети и изпълнения в приложението. Тези две нишки никога не комуникират директно и блокират взаимно пътя си в резултат на синхронизация.

UI нишката и JS нишката никога не споделят и не работят върху едни и същи данни. Те предпочитат да обменят сериализирани съобщения.

Докато започва визуализацията, Reconciler започва "диференциация", когато генерира нов V-DOM и модифицираното оформление се изпраща към shadow нишка, която генерира shadow nodes.

След изчисляване на оформлението, той изпраща параметъра към нишката на потребителския интерфейс, защото единственият основен поток може да изобрази нещо на екрана.

Потребителският интерфейс ще се представи, когато нишката на сенките ще изпрати генерирано оформление към нишката на потребителския интерфейс.

Има 4 вида нишки в реагиращите местни приложения:

В реагиращото местно приложение нишките играят основна роля, като позволяват отделни задачи да се изпълняват заедно, докато една нишка изпълнява една задача, друга нишка може да изпълнява някаква друга отделна задача по паралелен начин.

* Основна нишка: Известена е също като UI нишка, която се използва за естествено изобразяване на iOS или Android UI. Той се създава автоматично в началото на приложенията, тъй като основната нишка ще породи всички долни нишки. UI нишката е и последната нишка, която завършва изпълнението, като изпълнява различни задачи за затваряне.
* JavaScript нишка: Той ще стартира основния логически код на JavaScript, включително сензорни събития, API обаждания и много други. Всички актуализации на собствените изгледи се пакетират и в края на всеки цикъл на събитието в нишката на JavaScript ще бъдат изпратени директно към основната страна. За по-добра ефективност JavaScript нишката изпраща пакетните актуализации в потребителския интерфейс, преди следващия кадър да представи крайния срок. Има и едно изключение от JS нишката, че родният изглед, който работи в нишката на потребителския интерфейс, не е блокиран поради бавна js нишка. напр .: ScrollView работи в UI нишка.
* Native Modules нишка: Native модулите обикновено се разпространяват като npm пакети, с изключение на типичните JavaScript файлове и се използват за достъп до специфичния за платформата API.
* Render нишка: Това е специална нишка, която изпраща команди към графичния процесор и използва за генериране на действителни OpenGL команди за изготвяне на потребителски интерфейс.

Нов проект React Native може да бъде лесно започнат как с помоща на Expo CLI, так и с React Native CLI.

Expo е рамка и платформа за универсални React приложения. Това е набор от инструменти и услуги, изградени около React Native и родните платформи, които ви помагат да развивате, изграждате, разгръщате и бързо да итератирате в iOS, Android и уеб приложения от същата JavaScript / TypeScript кодова база.

Тази дипломна работа исползва React Native CLI.

Трябва ни предварително да имаме Node.js, Watchman, React Native CLI, Xcode, JDK и Android Studio (проектът разработен на macOS за iOS и Android).

Инсталиране.

Homebrew менеджер на пакетите за macOS и Linux.

/bin/bash -c "$(curl -fsSL <https://raw.githubusercontent.com/Homebrew/install/master/install.sh>)"

Node.js е JavaScript среда, изградена на двигателя на JavaScript Chrome V8.

brew install node

Watchman е инструмент на Facebook за гледане на промени във файловата система.

brew install watchman

Xcode – интегрирана среда за разработка софтуер за macOS, iOS, padOS, AppleTV.

Най-лесният начин за инсталиране на Xcode е чрез Mac App Store. Инсталирането на Xcode ще инсталира и iOS Simulator и всички необходими инструменти за изграждане на приложението ви за iOS.

CocoaPods е изграден с Ruby и той ще бъде инсталируем с стандартния Ruby, наличен на macOS. CocoaPods е мениджър на зависимости за проекти на Swift и Objective-C Cocoa.

A screenshot of a computer

Description automatically generated

React Native има вграден интерфейс за команден ред. С npx react-native <command> текущата стабилна версия на CLI ще бъде изтеглена и изпълнена по време на изпълнение на командата.

npx react-native init SmartFarmMobile

Java Development Kit

<https://www.oracle.com/java/technologies/javase-downloads.html>

Android Studio

<https://developer.android.com/studio/index.html>

A screenshot of a computer screen

Description automatically generated

Код на компонент Thermometer

import React, {useRef} from 'react';

import {View, Text, StyleSheet} from 'react-native';

import Animated, {

Value,

cond,

lessOrEq,

greaterOrEq,

} from 'react-native-reanimated';

import {PanGestureHandler} from 'react-native-gesture-handler';

const CONTAINER\_HEIGHT = 200;

const CONTAINER\_WIDTH = 20;

const CONTROL\_WIDTH = 5;

const FLOW\_DIFF = 0.1;

const KNOB\_SIZE = 40;

export default ({

mineral,

currentVolume,

flow,

isDashboard,

hasControl,

onFlowChanged,

}) => {

const contentHeight =

((mineral.maxVolume - currentVolume) \* CONTAINER\_HEIGHT) /

mineral.maxVolume -

1;

const currentVolumeLabel = currentVolume.toLocaleString('en-US', {

maximumFractionDigits: 0,

});

const flowDiff = mineral.maxFlow \* FLOW\_DIFF;

const scaleHightInMl = mineral.maxFlow - mineral.minFlow + 2 \* flowDiff;

const scaleDiffHeightInPts = (flowDiff \* CONTAINER\_HEIGHT) / scaleHightInMl;

const scaleWorkHeightInMl = mineral.maxFlow - mineral.minFlow;

const scaleWorkHeightInPts =

(scaleWorkHeightInMl \* CONTAINER\_HEIGHT) / scaleHightInMl;

const startKnobPosition =

(((flow || mineral.defaultFlow) - (mineral.minFlow - flowDiff)) \*

CONTAINER\_HEIGHT) /

scaleHightInMl -

KNOB\_SIZE / 2;

let touchY = useRef(new Value(startKnobPosition)).current;

let minY = useRef(new Value(-KNOB\_SIZE / 2)).current;

let maxY = useRef(new Value(CONTAINER\_HEIGHT - KNOB\_SIZE / 2)).current;

const getTopValue = () => {

return cond(

lessOrEq(touchY, minY),

minY,

cond(greaterOrEq(touchY, maxY), maxY, touchY),

);

};

const onGestureEvent = event => {

const {nativeEvent} = event;

const y = nativeEvent.y - KNOB\_SIZE / 2;

touchY.setValue(y);

const currentKnobPositionInPts =

y <= -KNOB\_SIZE / 2

? 0

: y >= CONTAINER\_HEIGHT - KNOB\_SIZE / 2

? CONTAINER\_HEIGHT

: y + KNOB\_SIZE / 2;

const invertedY = CONTAINER\_HEIGHT - currentKnobPositionInPts;

const currentKnobPositionInMl =

(invertedY \* scaleHightInMl) / CONTAINER\_HEIGHT +

(mineral.minFlow - flowDiff);

onFlowChanged(currentKnobPositionInMl);

};

return (

<View style={styles.containerWrapper}>

<View style={styles.tankLabelWrapper}>

<Text style={styles.tankLabel}>{mineral.name}</Text>

</View>

<View style={styles.container}>

<View style={styles.tankWrapper}>

<Text style={[styles.textMark, {marginTop: contentHeight - 15}]}>

{currentVolumeLabel}

</Text>

<View

style={[

styles.tankContainer,

{backgroundColor: mineral.contentColor},

]}>

<View style={[styles.tankContent, {height: contentHeight}]} />

</View>

</View>

{hasControl && (

<View style={styles.controllerContainer}>

<View style={styles.controllerMarksContainer}>

<Text

style={[

styles.textMark,

{marginTop: scaleDiffHeightInPts - 15},

]}>

{mineral.maxFlow}

</Text>

{mineral.minFlow !== mineral.maxFlow && (

<Text

style={[

styles.textMark,

{marginTop: scaleWorkHeightInPts - 15},

]}>

{mineral.minFlow}

</Text>

)}

</View>

<PanGestureHandler onGestureEvent={onGestureEvent}>

<Animated.View style={styles.controllerScaleContainer}>

<View

style={[styles.dangerZone, {height: scaleDiffHeightInPts}]}

/>

<View

style={[styles.workZone, {height: scaleWorkHeightInPts}]}

/>

<View

style={[styles.dangerZone, {height: scaleDiffHeightInPts}]}

/>

<Animated.View

style={[

styles.knob,

{

top: getTopValue(),

},

]}>

<Text style={styles.textMark}>

{flow || mineral.defaultFlow}

</Text>

</Animated.View>

</Animated.View>

</PanGestureHandler>

<View />

</View>

)}

</View>

</View>

);

};

const styles = StyleSheet.create({

containerWrapper: {

width: '100%',

alignItems: 'center',

},

tankLabelWrapper: {

marginBottom: 10,

},

tankLabel: {

fontSize: 20,

},

container: {

flexDirection: 'row',

},

tankWrapper: {

flexDirection: 'row',

},

textMark: {

fontWeight: 'bold',

textAlign: 'right',

},

tankContainer: {

width: CONTAINER\_WIDTH,

height: CONTAINER\_HEIGHT,

padding: 0,

marginLeft: 5,

borderBottomWidth: 1,

borderLeftWidth: 1,

borderRightWidth: 1,

borderBottomLeftRadius: 5,

borderBottomRightRadius: 5,

},

tankContent: {

position: 'absolute',

backgroundColor: 'white',

width: '100%',

},

controllerContainer: {

flexDirection: 'row',

},

controllerMarksContainer: {

marginLeft: 10,

},

controllerScaleContainer: {

width: CONTROL\_WIDTH,

height: CONTAINER\_HEIGHT,

padding: 0,

borderWidth: 1,

marginLeft: 5,

},

dangerZone: {

backgroundColor: 'red',

},

workZone: {

backgroundColor: 'green',

},

knob: {

position: 'absolute',

left: -KNOB\_SIZE / 2,

width: KNOB\_SIZE,

height: KNOB\_SIZE,

borderRadius: KNOB\_SIZE / 2,

borderWidth: 1,

backgroundColor: 'yellow',

alignItems: 'center',

justifyContent: 'center',

},

});

Освен React Native този компонент ползва библиотеките react-native-reanimated (подобрена версия Animated) и react-native-gesture-handler.

React Native Reanimated осигурява по-изчерпателна абстракция на ниско ниво за API за анимирана библиотека, която трябва да бъде изградена върху и следователно позволява много по-голяма гъвкавост, особено когато става въпрос за взаимодействия, базирани на жестове.

React Native Gesture Handler предоставя API-та за управление на жестове, задвижвани от родния компютър, за изграждане на възможно най-добри базирани на допир преживявания в React Native.

С тази библиотека жестовете вече не се контролират от системата за отговор на JS, а вместо това се разпознават и проследяват в нишката на потребителския интерфейс. Това прави взаимодействията с докосване и проследяването на жестове не само плавни, но и надеждни и детерминирани.

A drawing of a face

Description automatically generated

Фигура 5 Компонент Slider

Код на основния екран:

import React, {useState, useEffect} from 'react';

import {

View,

Text,

SafeAreaView,

TextInput,

TouchableNativeFeedback,

Alert,

} from 'react-native';

import {Thermometer} from '../components/Thermometer';

import styles from './Main.styles';

const mineral\_1 = {

name: 'MultiMix',

contentColor: 'magenta',

minFlow: 200,

maxFlow: 250,

defaultFlow: 220,

maxVolume: 60,

};

const mineral\_2 = {

name: 'X-BLUE',

contentColor: 'blue',

minFlow: 250,

maxFlow: 250,

defaultFlow: 250,

maxVolume: 60,

};

const mineral\_3 = {

name: 'X-ROC',

contentColor: 'cyan',

minFlow: 200,

maxFlow: 400,

defaultFlow: 300,

maxVolume: 60,

};

const Main = () => {

const [volume1, setVolume1] = useState(0);

const [volume2, setVolume2] = useState(0);

const [volume3, setVolume3] = useState(0);

const [volumeInput1, setVolumeInput1] = useState('');

const [volumeInput2, setVolumeInput2] = useState('');

const [volumeInput3, setVolumeInput3] = useState('');

const [isWaterOpen, setIsWaterOpen] = useState(false);

const [flow1, setFlow1] = useState(mineral\_1.defaultFlow);

const [flow2, setFlow2] = useState(mineral\_2.defaultFlow);

const [flow3, setFlow3] = useState(mineral\_3.defaultFlow);

const toggleWater = () => setIsWaterOpen(!isWaterOpen);

const clearAll = () => {

setIsWaterOpen(false);

setVolume1(0);

setVolume2(0);

setVolume3(0);

};

const changeVolume1 = () => {

setVolume1((Number.parseInt(volumeInput1, 10) || 0) + volume1);

setVolumeInput1('');

};

const changeVolume2 = () => {

setVolume2((Number.parseInt(volumeInput2, 10) || 0) + volume2);

setVolumeInput2('');

};

const changeVolume3 = () => {

setVolume3((Number.parseInt(volumeInput3, 10) || 0) + volume3);

setVolumeInput3('');

};

const tank1Speed = 1;

const tank2Speed = 0.25;

const tank3Speed = 0.4;

useEffect(() => {

let interval = null;

if (isWaterOpen) {

interval = setInterval(() => {

setVolume1(v1 => v1 + tank1Speed + tank2Speed + tank3Speed);

if (volume2 !== 0) {

setVolume2(v2 => v2 - tank2Speed);

} else {

Alert.alert('Add to tank 2');

setIsWaterOpen(false);

}

if (volume3 !== 0) {

setVolume3(v3 => v3 - tank3Speed);

} else {

Alert.alert('Add to tank 3');

setIsWaterOpen(false);

}

}, 1000);

} else if (!isWaterOpen && volume1 !== 0) {

clearInterval(interval);

}

return () => clearInterval(interval);

}, [isWaterOpen, volume1, volume2, volume3]);

const onFlow1Changed = newFlow => {

setFlow1(Number.parseInt(newFlow, 10));

};

const onFlow2Changed = newFlow => {

setFlow2(Number.parseInt(newFlow, 10));

};

const onFlow3Changed = newFlow => {

setFlow3(Number.parseInt(newFlow, 10));

};

return (

<SafeAreaView>

<View style={styles.contentContainer}>

<View style={styles.titleContainer}>

<Text style={styles.text}>SmartFarmDemo</Text>

</View>

<View style={styles.termContainer}>

<View style={styles.termCol}>

<Thermometer

mineral={mineral\_1}

currentVolume={volume1}

hasControl

onFlowChanged={onFlow1Changed}

flow={flow1}

/>

<TextInput

value={volumeInput1}

onChangeText={text => setVolumeInput1(text)}

onBlur={changeVolume1}

style={styles.textInput}

/>

</View>

<View style={styles.termCol}>

<Thermometer

mineral={mineral\_2}

currentVolume={volume2}

hasControl

onFlowChanged={onFlow2Changed}

flow={flow2}

/>

<TextInput

value={volumeInput2}

onChangeText={text => setVolumeInput2(text)}

onBlur={changeVolume2}

style={styles.textInput}

/>

</View>

<View style={styles.termCol}>

<Thermometer

mineral={mineral\_3}

currentVolume={volume3}

hasControl

onFlowChanged={onFlow3Changed}

flow={flow3}

/>

<TextInput

value={volumeInput3}

onChangeText={text => setVolumeInput3(text)}

onBlur={changeVolume3}

style={styles.textInput}

/>

</View>

</View>

<View style={styles.buttonsContainer}>

<TouchableNativeFeedback onPress={toggleWater}>

<View style={styles.button}>

<Text style={styles.text}>{`${

isWaterOpen ? 'Close' : 'Open'

} water`}</Text>

</View>

</TouchableNativeFeedback>

<TouchableNativeFeedback onPress={clearAll}>

<View style={styles.button}>

<Text style={styles.text}>Reset</Text>

</View>

</TouchableNativeFeedback>

</View>

</View>

</SafeAreaView>

);

};

export default Main;

HTML

HTML е език за маркиране на хипертекст. Той позволява на потребителя да създава и структурира секции, параграфи, заглавия, връзки и блокове за уеб страници и приложения. HTML не е език за програмиране, което означава, че той няма способността да създава динамични функции. Вместо това ви позволява да организирате и форматирате документите си по подобен начин на Microsoft Word. Когато работим с HTML, ние използваме прости кодови структури (тагове и атрибути), за да маркираме страницата на уебсайта. Например, можем да създадем абзац, като поставим прикачения текст в оригиналния <p> таг и затварящия </p> таг.

HTML е изобретен от Тим ​​Бърнърс-Ли, физик в изследователския институт на CERN в Швейцария. Той дойде с идеята за интернет хипертекстова система. Хипертекст означава текст, който съдържа връзки към други текстове, до които зрителите могат да получат незабавен достъп. Той публикува първата версия на HTML през 1991 г., състояща се от 18 HTML тагове. Оттогава всяка нова версия на HTML езика се появява с маркиране на нови тагове и атрибути (модификатори на тагове). Според справочника за HTML елементи от мрежата за разработчици на Mozilla, в момента има 140 HTML тагове, въпреки че някои от тях вече са оттеглени (не се поддържат в съвременните браузъри). Поради бързия си растеж на популярност, HTML вече се счита за официален уеб стандарт. Спецификациите на HTML се поддържат и разработват от World Wide Web Consortium (W3C). Можете да проверите най-новото състояние на езика по всяко време на уебсайта на W3C.

Най-голямата актуализация на езика беше въвеждането на HTML5 през 2014 г. Няколко нови семантични маркера са добавени към маркирането, които показват значението на тяхното собствено съдържание, като <article>, <header> и <footer>.

CSS

Как работи HTML?

HTML документите са файлове, които завършват с разширение .html или .htm. Можете да го видите с всеки уеб браузър (като Google Chrome, Safari или Mozilla Firefox). Браузърът чете HTML файла и показва неговото съдържание, така че потребителите в Интернет да могат да го преглеждат.

Обикновено средният уебсайт включва няколко различни HTML страници. Например: началните страници, обикновените страници и страниците за контакти ще имат отделни HTML документи.

Всяка HTML страница е съставена от набор от тагове (също наречени елементи), които можете да наречете градивните елементи на уеб страниците. Те създават йерархия, която структурира съдържанието в секции, параграфи, заглавия и други блокове от съдържание.

Повечето HTML елементи имат отворени и затворени, които използват синтаксиса <tag> </tag>.

По-долу можете да видите пример на кода, с който можете да структурирате HTML елементи:

<div>

<h1> Основна позиция </h1>

<h2> Подпозиция </h2>

<p> Точка 1 </p>

<img src = "/" alt = "Изображение">

<p> Елемент втори с хипервръзка <a href="https://example.com"> </a> </p>

</ div>

Най-важният елемент е просто разделение (<div> </div>), което можете да използвате, за да маркирате големи секции от съдържание. Той съдържа заглавие (<h1> </h1>), подзаглавие (<h2> </h2>), два абзаца (<p> </p>) и изображение (<img>).

Вторият параграф съдържа връзка (<a> </a>) с атрибут href, който съдържа целевия URL адрес.

Името на изображението също има два атрибута: src за пътя на изображението и alt за описанието на изображението.

Javascript

Преглед на най-използваните HTML маркери

HTML маркерите са от два основни типа: блокове и вградени маркери.

Елементите на ниво блок заемат цялото свободно пространство и винаги започват нов ред в документа. Заглавията и параграфите са чудесни примери за блокови тагове.

Вградените елементи заемат толкова място, колкото е необходимо, и не стартират нов ред на страницата. Те обикновено се използват за форматиране на вътрешното съдържание на елементи на ниво блок. Връзки и подчертани линии са добри примери за вградени маркери.

Тагове на ниво блок

Три тагове на ниво блок, които всеки HTML документ трябва да съдържа: <html>, <head> и <body>.

<html> </html> тагът е елемент от най-високо ниво, който обхваща всяка HTML страница.

Тегът <head> </head> съдържа мета информация като заглавие на страницата и кодиране.

И накрая, тагът <body> </body> съдържа цялото съдържание, показано на страницата.

<html>

<head>

<! - МЕТА ИНФОРМАЦИЯ ->

</ head>

<body>

<! - СЪДЪРЖАНИЕ НА СТРАНИЦА ->

</ body>

</html>

Заглавията са 6 нива в HTML. Те варират от <h1> </h1> до <h6> </h6>, където h1 е заглавието с най-високо ниво и h6 е най-ниското.

Параграфите са приложени <p> </p>, докато коментарите за блокове използват маркера <blockquote> </blockquote>.

Разделите са по-големи секции от съдържание, които обикновено съдържат множество абзаци, изображения, понякога блокират записи и други по-малки елементи. Можем да ги маркираме с помощта на <div> </div> тага. Елемент div може да съдържа друг div tag вътре в него.

Можете също да използвате маркерите <ol> </ol> за подредени списъци и <ul> </ul> за списъци, които не са подредени. Отделните елементи от списъка трябва да бъдат затворени в маркера <li> </li>. Например, това изглежда основен неупореден списък в HTML:

<ul>

<li> Списък с елемент 1 </li>

<li> Списък на списък 2 </li>

<li> Списък на списък 3 </li>

</ul>

Вградени (inline) маркери

Много вградени маркери се използват за форматиране на текст. Например, етикетът <strong> </strong> превръща маркирания елемент с удебелен шрифт, докато маркерите <em> </em> го правят курсив.

Хипервръзките също са вградени елементи, които изискват <a> </a> тагове и href атрибути, за да посочат дестинацията на връзката:

<a href="https://example.com/"> Кликнете тук! </a>

Изображенията също са вградени елементи. Можете да добавите такъв с <img> без затварящ маркер. Но също така ще трябва да използвате атрибута src, за да определите пътя на изображението, например:

<img src = "/ images / example.jpg" alt = "Примерно изображение">

CSS

CSS е официален език, използван за описание на външния вид на документ, създаден с помощта на език за маркиране (HTML, XHTML, XML). Името идва от английските таблици за каскадни стилове, което означава „каскадни таблици със стилове“.

Целта на CSS е да отдели това, което определя външния вид на страницата от нейното съдържание. Ако документ е създаден с помощта само на HTML, той определя не само всеки елемент, но и начина на показване (цвят, шрифт, позиция на блока и т.н.). Ако са свързани каскадни таблици със стилове, тогава HTML описва само реда на обектите. И CSS е отговорен за всичките им свойства. В HTML е достатъчно да напишете клас, без да изброявате всички стилове всеки път.

Подобна технология:

* осигурява сравнително проста и бърза разработка, тъй като веднъж създаден, дизайнът може да се приложи към много страници;
* увеличава гъвкавостта и използваемостта на редактирането - достатъчно е да редактирате CSS, за да промените дизайна навсякъде;
* прави кода по-прост, като намалява повторението на елементите. По-лесно е да се чете за програмисти и ботове за търсене;
* Ускорява се времето за зареждане, защото CSS може да се кешира при първо отваряне, а в следващите се четат само структура и данни;
* увеличава броя на визуалните решения за представяне на съдържание;
* предоставя възможност за лесно прилагане на различни стилове към един и същ документ (например, създаване на персонализирана версия за мобилни устройства или специални стилове за хора със зрителни увреждания).

Тоест, каскадните таблици служат не само за изпълнение на дизайна, но и променят радикално подхода към изграждането на сайта, опростявайки работата на разработчиците и осигурявайки гъвкавост при внедряване. За това е CSS.

Необходимостта от развитие на CSS беше призната от W3C през 90-те години. През 1996 г. беше приет стандартът CSS1, който ви позволява да промените параметрите на шрифта, цвета, текстовите атрибути, подравняването и подплънките. През 1998 г. CSS2 беше пуснат, добавяйки възможността за използване на блоково оформление, звукови таблици, генерирано съдържание, указатели, медийни страници. Версията CSS3 значително увеличи възможностите за стилове: стана достъпна за създаване на анимирани елементи, без да се използва JavaScript, появи се поддръжка за антилизинг, сенки, градиенти и др. Спецификацията беше разделена на модули, всеки от които започна да се развива отделно. От 2011 г. се разработват CSS4 модули. Възможностите все още са описани в чернови версии.

CSS можно охарактеризовать простыми словами как набор правил, описывающих, как должен выглядеть элемент.

Правило состоит из селектора и блока объявлений.

Селекторът казва към кой елемент ще бъдат приложени свойствата на стила, описани в CSS. Всеки маркер, който има форматиране (размер, цвят и т.н.), може да се използва като селектор. Ако трябва да зададете различни стилове за маркер или да приложите такъв за различни елементи, използвайте класове и запис на формата "таг.клас {свойство: стойност;}". Името на класа е посочено с латински букви и може да съдържа подчертаване или тире. Ако не посочите маркер, но започнете да записвате с ".сlass", тогава можете да използвате правилото за всеки маркер. Ако изброите няколко класа за един маркер, всички описани стилове ще бъдат приложени към него. Идентификаторът указва уникално име за елемента, който трябва да бъде оформен или скриптиран. Записът "#Идентификаторът {свойство: стойност;}". Името на идентификатора се състои от букви от латинската азбука, разрешени са тирета и подчертаването. За да приложите идентификатор към конкретен маркер, посочете неговото име, след това без интервал и чрез фунта подпишете името на идентификатора.

Блокът за деклариране се състои от свойство: стойностни двойки (винаги последвани от двоеточие), затворени в къдрави скоби. Записите завършват със запетая. CSS е нечувствителен към раздели, интервали, регистър. Изборът на начина на писане (в колона с отстъпи или просто в ред) остава по преценка на разработчика. Ако различни стойности за едно и също свойство са регистрирани за един и същ селектор, тогава приоритетът се дава на долния запис.

Универсален селектор

\* {

**margin**: 0;

**padding**: 0;

}

Тагов селектор

**p** {

**font-family**: arial, helvetica, **sans-serif**;

}

Селектор на класа

.**note** {

**color**: **red**;

**background-color**: **yellow**;

**font-weight**: **bold**;

}

Селектор на идентификатора

#**paragraph1** {

**margin**: 0

}

Селектор на атрибута

**a**[**href**="http://www.somesite.com"] {

**font-weight**: **bold**;

}

Контекстен селектор

**div**#**paragraph1** **p**.**note** {

**color**: **green**;

}

CSS може да бъде свързан с HTML по няколко начина:

* когато описанието на стила е в отделен файл, то може да бъде свързано с документа чрез <link> елемента, включен в елемента <head>:

<!DOCTYPE html>

<**html**>

<**head**>

.....

<**link** rel="stylesheet" type="text/css" href="style.css">

</**head**>

<**body**>

.....

</**body**>

</**html**>

* когато файл със стил е поставен отделно от неговия родителски документ, той може да бъде свързан с документа с оператора @import в елемента <style>:

<!DOCTYPE html>

<**html**>

<**head**>

.....

<**style** media="all">

@**import** **url**(**style**.**css**);

</**style**>

</**head**>

<**body**>

.....

</**body**>

</**html**>

* когато стиловете са описани в документ, те могат да бъдат включени в елемента <style>, който е включен в елемента <head>:

<!DOCTYPE html>

<**html**>

<**head**>

.....

<**style**>

**body** {

**color**: **red**;

}

</**style**>

</**head**>

<**body**> ..... </**body**>

</**html**>

* когато стиловете са описани в тялото на документа, те могат да бъдат разположени в атрибутите на отделен елемент

<!DOCTYPE>

<**html**>

<**head**>

.....

</**head**>

<**body**>

<**p** style="font-size: 20px; color: green; font-family: arial, helvetica, sans-serif">

.....

</**p**>

</**body**>

</**html**>

Javascript

JavaScript е междуплатформен, обектно-ориентиран, скриптов език. JavaScript е малък и лек език; не е предназначен за създаване на цялостни, цялостни приложения, а за лесно интегриране в други продукти и приложения като уеб браузъри. В рамките на основното приложение JavaScript може да се свързва с обекти в това приложение и да упражнява програмен контрол над тях.

Основният JavaScript съдържа набор от основни обекти, като масив, дата и математика и набор от езикови елементи: оператори, контролни структури и изрази. Ядрото на JavaScript може да бъде разширено за различни цели с добавки, например:

* Client-side JavaScript разширява ядрото на езика, като предоставя контролни обекти за браузъра (Навигатор или друг уеб браузър) и неговия обектен модел на документ (DOM). Например разширенията от страна на клиента позволяват на приложението да излага елементи в HTML форма и да реагира на потребителски събития като кликвания на мишката, въвеждане на формуляр и навигация на страница.
* Server-side JavaScript разширява ядрото на езика, като предоставя обекти за стартиране на JavaScript на сървъра. Например разширенията от страна на сървъра позволяват на приложението да се свърже с релационна база данни, да осигури непрекъснатост на информацията между обажданията към приложението или да изпълнява файлови операции на сървъра.

Използвайки функциите на JavaScript LiveConnect, можете да позволите на Java кода и JavaScript да комуникират помежду си. Можете да създадете екземпляр на Java обект от JavaScript и да осъществите достъп до неговите публични методи и свойства. От Java можете да получите достъп до обекти, свойства и методи на JavaScript.

JavaScript е използван за първи път в браузърите Netscape.

React (понякога React.js или ReactJS) е JavaScript библиотека с отворен код за разработване на потребителски интерфейси. React се разработва и поддържа от Facebook, Instagram и общността на отделни разработчици и корпорации. React може да се използва за разработване на едностранични и мобилни приложения. Целта му е да осигури висока скорост, простота и мащабируемост. Като библиотека за разработване на потребителски интерфейси, React често се използва с други библиотеки като MobX, Redux и GraphQL.

* Еднопосочно предаване на данни. Свойствата се предават от родителски компоненти на деца. Компонентите получават свойства като набор от неизменяеми стойности, така че компонентът не може директно да модифицира свойствата, но може да извиква промени чрез функции за обратно извикване. Този механизъм се нарича "свойства надолу, събития нагоре".
* React използва виртуалния DOM. React създава структура в кеш паметта, която ви позволява да изчислите разликата между предишното и текущото състояние на интерфейса, за да актуализирате оптимално DOM на браузъра. По този начин програмистът може да работи със страницата, приемайки, че тя се актуализира като цяло, но библиотеката сама решава кои компоненти на страницата трябва да бъдат актуализирани.
* JavaScript XML (JSX) е разширение на синтаксиса на JavaScript, което ви позволява да използвате подобен на HTML синтаксис, за да опишете структурата на интерфейса. Като правило компонентите се пишат с помощта на JSX, но е възможно да се използва и обикновен JavaScript. JSX прилича на друг език, създаден от Facebook за разширението PHP, XHP.
* Методите на жизнения цикъл позволяват на разработчика да изпълнява код на различни етапи от жизнения цикъл на компонента. Например: **shouldComponentUpdate** - ви позволява да предотвратите преначертаване на компонент, като върнете false, ако не е необходимо преначертаване; **componentDidMount** - извиква се след първото изобразяване на компонента. Често се използва за задействане на извличане на данни от отдалечен източник чрез API; **render** е най-важният метод в жизнения цикъл. Всеки компонент трябва да има този метод. Обикновено се извиква, когато данните на компонента се променят, за да се преначертаят данните в интерфейса.
* React се използва не само за рендиране на HTML в браузъра. Например, Facebook има динамична графика, която се изобразява в маркери <canvas>. Netflix и PayPal използват изоморфни изтегляния, за да изобразяват идентичен HTML на сървъра и клиента.

A close up of a map

Description automatically generated

Фигура 6 Главна страница SPA

Код на React SPA (demo):

import React, { useEffect, useState } from "react";

import "./App.css";

import \* as firebase from "firebase/app";

import "firebase/database";

import { TextField } from "@material-ui/core";

var firebaseConfig = {

apiKey: "AIzaSyAL2tyMYzB4Ykhh55ggdddYQDAl8VuvzVtcJWA",

authDomain: "smartfarm-b56cd.firebaseapp.com",

databaseURL: "https://smartfarm-b56cd.firebaseio.com",

projectId: "smartfarm-b56cd",

storageBucket: "smartfarm-b56cd.appspot.com",

messagingSenderId: "503665889197",

appId: "1:503665889197:web:b4c937bbf22fe42e595b24",

measurementId: "G-KXWKTCKRVW",

};

firebase.initializeApp(firebaseConfig);

// Get a reference to the database service

var database = firebase.database();

function App({ classes }) {

const [mineral1, setMineral1] = useState(0);

const [mineral2, setMineral2] = useState(0);

const [mineral3, setMineral3] = useState(0);

useEffect(() => {

const subscriber1 = database

.ref("/user1/mineral1")

.on("value", (snapshot) => {

setMineral1(snapshot.val());

});

const subscriber2 = database

.ref("/user1/mineral2")

.on("value", (snapshot) => {

setMineral2(snapshot.val());

});

const subscriber3 = database

.ref("/user1/mineral3")

.on("value", (snapshot) => {

setMineral3(snapshot.val());

});

return () => {

subscriber1();

subscriber2();

subscriber3();

};

}, []);

return (

<div className="root">

<div className="minerals\_container">

<div className="mineral\_container">

<TextField

id="mineral1"

label="MultiMix"

variant="outlined"

value={mineral1}

/>

</div>

<div className="mineral\_container">

<TextField

id="outlined-basic"

label="X-BLUE"

variant="outlined"

value={mineral2}

/>

</div>

<div className="mineral\_container">

<TextField

id="outlined-basic"

label="X-ROC"

variant="outlined"

value={mineral3}

/>

</div>

</div>

<div className="uabg\_logo" />

<div className="bio\_logo" />

</div>

);

}

export default App;

Литература

<https://www.arduino.cc/>

<https://firebase.google.com/>

<https://reactnative.dev/>

<https://www.wikipedia.org/>

<https://wiki.rookee.ru/>