ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

ДИПЛОМНА РАБОТА

Тема: Автоматизиран диспенсър за хапчета

Дипломант:
Роберта Нецова

Научен ръководител:
маг. инж. Росен Витанов

СОФИЯ
2019

УВОД

В днешния свят, който предлага множество от възможности, хората непрестанно търсят начини да създават повече удобство за себе си и околните, достъпвайки се до технологиите.

С появата на 5G мрежата по-голямата част от населението ще има изключително бърз интернет, което поставя солидни основи за развитието на IoT (Internet of Things). Възможността предмети от бита да бъдат контролирани от разстояние не само улеснява значително вършенето на различни задължения, а ги прави даже приятни. От контрол на по-обикновени предмети като домашно осветление, до модернизирани системи за заключване на дома, вариантите са неограничени.

Друга сфера, която е засегната в дипломната работа, покрива проблем срещан при по-голямата част от по-възрастните представители на населението. Грижата за здравето в домашни условия не винаги е лесна, но с помощта на новите технологии, методиките, които досега не са били предразположени към развитие, могат да бъдат направени много по-ефективни за хората, нуждаещи се от тях.

Вдъхновена от вече съществуващи решения, направената разработката представя един различен начин за изпълняване на лекарствени рецепти.

1.

Обзор на съществуващи решения

1.1. Въведение в проблема, с който се среща дипломната

С напредване на възрастта, човешкият организъм често започва да страда от здравословни проблеми (нестабилно кръвно налягане, сърдечно съдови заболявания и пр.). Медикаментите са ефективен начин за поддържане на здравословното състояние на по-възрастните хора. Поради големият брой, лекарства вземани дневно, един на четирима пенсионери пропуска редовната си дозировка необходими лекарства. Проучванията показват, че пациенти в напреднала възраст, които не следват предписаната медикаментозна доза, имат 76 % по–голям шанс да влошават здравето си в сравнение с хората, които спазват точно рецептите си. **[1]**

1.2. Решения на проблема

MDS & MCA

Multi compartment Compliance Aids (MCA) са така познатите на много хора диспенсъри за лекарства , а Monitored Dosage Systems (MDS) са известни още под името блистерни пакети. Главната цел и на двата метода е да раздели лекарствата почасово на дозировки. По този начин се намалява стресът на пациентите от незнанието дали правилната дозировка е взета.

фиг 1.1
Monitored Dosage System

фиг 1.2
Multi compartment Compliance Aid

На **фиг 1.1** и **фиг 1.2** са показани решения, които се използват в много домове днес. С напредването на технологията, обаче тези начини за разпределение на медикаменти се развиват, улеснявайки пациентите и хората , отговарящи за техните дозировки (болногледачи, близки хора). Проучване, направено през 2014 година, поставя под наблюдение 96 човека, които в продължение на 12 месеца приемат медикаментите си чрез автоматизирано диспенсърно устройство. 94% процента от тях казват, че устройството е лесно за ползване, а 95% го намират за надежден продукт. В краят на годината 85% споделят, че биха използвали подобна машина в бъдеще и определено диспенсърът им е помогнал да се чувстват по- спокойни относно правилната дозировка на лекарствата си. **[2]**

Автоматизирани диспенсъри

- Pivotell® Mk3-11 Диспенсър

Pivotell е една от марките, произвеждащи по-модернизирани диспенсъри на сравнително достъпни цени. Този продукт напомня на потребителя чрез аларма и мигаща светлина когато трябва да бъде взето лекарство. Веднъж зареден, уредът пуска нужната дозировка в определения час (през деня или нощта), а останалите деления на вътрешния съд остават заключени.

фиг 1.3
Pivotell Mk3-11 Диспенсър

Диспенсърът съдържаща 29 секции **фиг.1.3.** (28 с дозировка и 1 празна, която стои на отвора на горния корпус при първо използване на презареден контейнер) Максимален брой настройки на алармата е 28 като периодът, преди ново презареждане зависи от дозировките дневно – при една доза дневно пълнителят издържа 28 дена, докато при 7 дозировки дневно на всеки 4 дни е нужно презареждане. **[3]**

- Pivotell Advance GSM диспенсър за хапчета с Приложение за известяване

Продуктът, е с подобен дизайн **(фиг 1.4)** и механизъм на работа с оптимизацията, че е добавена Roaming SIM карта, чрез която диспенсърът може да праща есемеси до 3 номера, както и 1 имейл. **[5]**

фиг 1.4
Pivotell Advance GSM

- Friendly Locked e-pill MedSmart VOICE Automatic Pill Dispenser

Продуктът на E-pill също съдържа 28 разделения. Различното е, че алармата е под формата на съобщения като „Време е да приемете медикамента си“ и „Благодаря Ви“, което придава персоналност на уреда. Приложение, свързващо се приложение, достъпно за Android и IOS, чрез което по лесен начин може да бъде настроен уредът, както и да се следи кога е имало дозировки. **[6].**

1.3. Видове механизми за вземане на дозировка

Разделение по дозировки

Примерите, посочени в точка 1.2, имат сходството, че медикаментите в диспенсъра са подредени по дозировки, следвайки принципа на MDS & MCA (точка 1.1). В нужния час се пуска дозировката, която предварително е сложена в специален контейнер. За момента този метод е наложилият се на пазара и е използван от всички фирми, предлагащи диспенсъри.

Разделение по медикаменти

Друг начин, по който могат да се изпълняват дозировки е чрез разпределение по хапчета. В нужния час се преброява и пуска по едно хапче от всяко предназначено лекарство.

Методът може да бъде осъществен чрез засмукване посредством вакуум технология, която успешно се справя с различни форма и размери предмети, включително и медикаменти. На пазара все още няма диспенсъри използващи подобна технология, но е силно разпространени в други среди.

Съществуващи решения, използващи метода “Разделение по медикаменти”

- Вендинг машина за сладолед

Вендинг машините за сладолед започват като startup проект още през 1995 г и са смятани за първите проявления на вакуумните роботизирани диспенсинг технология. Тези машини се използват и до днес и разликата с останалите по-често срещани вендинг автомати, които отпускат

фиг 1.6
Вакуумно захващане

желания продукт чрез спираловидна шина, тази технология използва вакуумно засмукване, за да вземе желания сладолед. **(фиг 1.6)**

QR код, показващ начина на работа

1.4. Кратко описание на дипломната работа

В текущата дипломна работа ще бъде разгледана подробно разработката на проект за автоматизиран диспенсър. Използваният механизъм за дозировка е „разделение по медикаменти“. Този метод беше избран с идеята да се улесни презареждането на апарата и да се увеличи периода преди презареждане в случаи на повече от 4 дневни дозировки. Нестандартните дизайни и размери на хапчетата доведоха до осъществяването метода е използван вакуумен механизъм (повече информация в глава 2). Хапче по хапче диспенсърът може да изпълнява различни рецепти с максимум 6 различни медикамента.

2.

Използвани технологии и развойни среди

В тази глава са описани главните характеристики и предимства на използваните за разработката на дипломната работа развойни среди и технологии.

2.1. Избор на технологии

- Node.js

Node.js е JavaScript среда за изпълнение. Благодарение на него оригиналният JavaScript може да работи като самостоятелна апликация. Предимствата на Node.js са, че е събитийноконтролиран и използва неблокиращи входове и изходи (Non-blocking I/O), което го прави изключително олекотен и ефикасен. Тъй като JavaScript не е много подходящ за задачи, изискващи няколко нишки, се използва Non-blocking I/O. Пакетната му екосистема, npm, е една от най-големите open-source екосистеми с библиотеки в света.

- npm

npm (Node package manager) е пакетен мениджър, използван за Node.js. Въпреки, че създава някои от структурооорганизационните директории, най-важната характеристика е, че организира dependencies и packages. Това позволява всички дефинирани във файла package.json dependencies на проекта да бъдат добавени заедно като се изпълни командата npm install. По този начин се спестява време при по-големи проекти и се улеснява процеса на споделяне на разработката. Друго предимство е, че в проекта могат да бъдат задавани конкретни версии на всяко dependency, като така предотвратяват проблеми в следствие на актуализации. **[6]**

- Express

Express е олекотен framework за уеб приложения, който спомага за организирането на приложението в MVC (Model-View-Controller) архитектура на сървърната страна. Зареждането на страниците става динамично, а езикът, на който се пише е темплейтен (в разработката беше използван Jade). Чрез Express лесно управляване на всичко от routes до управляване на requests и views. **[7]**

- Bootstrap

Bootstrap е open-source front-end уеб framework. Чрез него лесно може да се изработва дизайн на responsive приложения (благодарение на 12-колоново решетъчно разделение на страниците). Предлага се широк списък от стилизирани компоненти (като падащи менюта). Много от тях са интерактивни, заради JavaScript plugin-и, които са вградени в пакета на Bootstrap. Не на последно място, framework има подробна и ясна документация, благодарение на която работата се улеснява значително.

- C/C++

C е процедурен език за програмиране (няма обекти). Макар Разработен е първоначално за системно програмиране за писане на операционни системи се използва и за **компилиращ се development**. Основните характеристики включват достъп до паметта на ниско ниво, прост набор от ключови думи и чист стил. Много по-късни езици са заимствали синтаксис/функции директно или индиректно от C. Примери са C++, Java, PHP, JavaScript и много други. **[8]**

C++ е обектно ориентиран програмен език, работещ на ниско ниво. Това отваря на девелопъра голям контрол над компютърните ресурси. Изключително мащабируем е , което го прави удобен за разработка на приложения, които са много ресурсоемки. Като статично типизиран език, C ++ обикновено е по-ефективен от динамично типизираните езици, защото кодът се проверява преди да бъде изпълнен. **[9]**

- MQTT

MQTT е опростен протокол за съобщения, който е изключително енергоспестяващ. Това го прави много разпространен в IoT разработките, където често се среща устройствата да работят на батерии. MQTT позволява да се изпращат съобщения за контрол на изходи на контролери, да се чете или публикува информация. Съобщенията се изпращат от publisher и се получават от subscribers. Всяко съобщение се изпраща на определена тема (topic). Голямо предимство е, че в случай на разпадната връзка, MQTT брокера преизпраща съобщението докато не бъде получено.

Важно е да се спомене, че publisher и subscribers не знаят много информация един за друг (порт, на който работят, информация на потребителите в случай, че има такъв). Именно брокерът ги обединява.

2.2. Избор на развойни среди

- Arduino IDE

Интегрираната среда за разработка на Arduino - или Arduino Software (IDE) - съдържа текстов редактор за писане на код, област за съобщения, текстова конзола, лента с инструменти за общи функции и серия от менюта. Той се свързва с хардуера на Arduino и Genuino, за да качва програми и комуникира с тях. (1) Средата поддържа и NodeMCU борда, използван за разработката. Поддържаните програмни езици са C и C++ и Java.

- Sublime text

Sublime Text е междуплатформен софтуер и текстов редактор с приложно-програмен интерфейс (API), написан на Python. Функционалността му може да бъде допълвана от потребителите чрез плъгини. Повечето от допълненията са с лиценз за свободен софтуер и се разработват и поддържат от потребителите.

Две от основните характеристики на Sublime Текст 3 са символно индексиране и управление на прозореца. Символното индексиране позволява Sublime Текст, да сканира файлове и да изгражда индекс за улесняване функции Goto Definition и Goto Symbol в

проекта. Pane managemant позволява на потребителите да се движат между прозорците чрез клавишни комбинации (hotkey).

- Сървър

Уеб сървърът е Ubuntu 18.04 server. За да е сполучлива връзката интернет от външни източници, портът 80 е свързан към външната мрежа.

3.

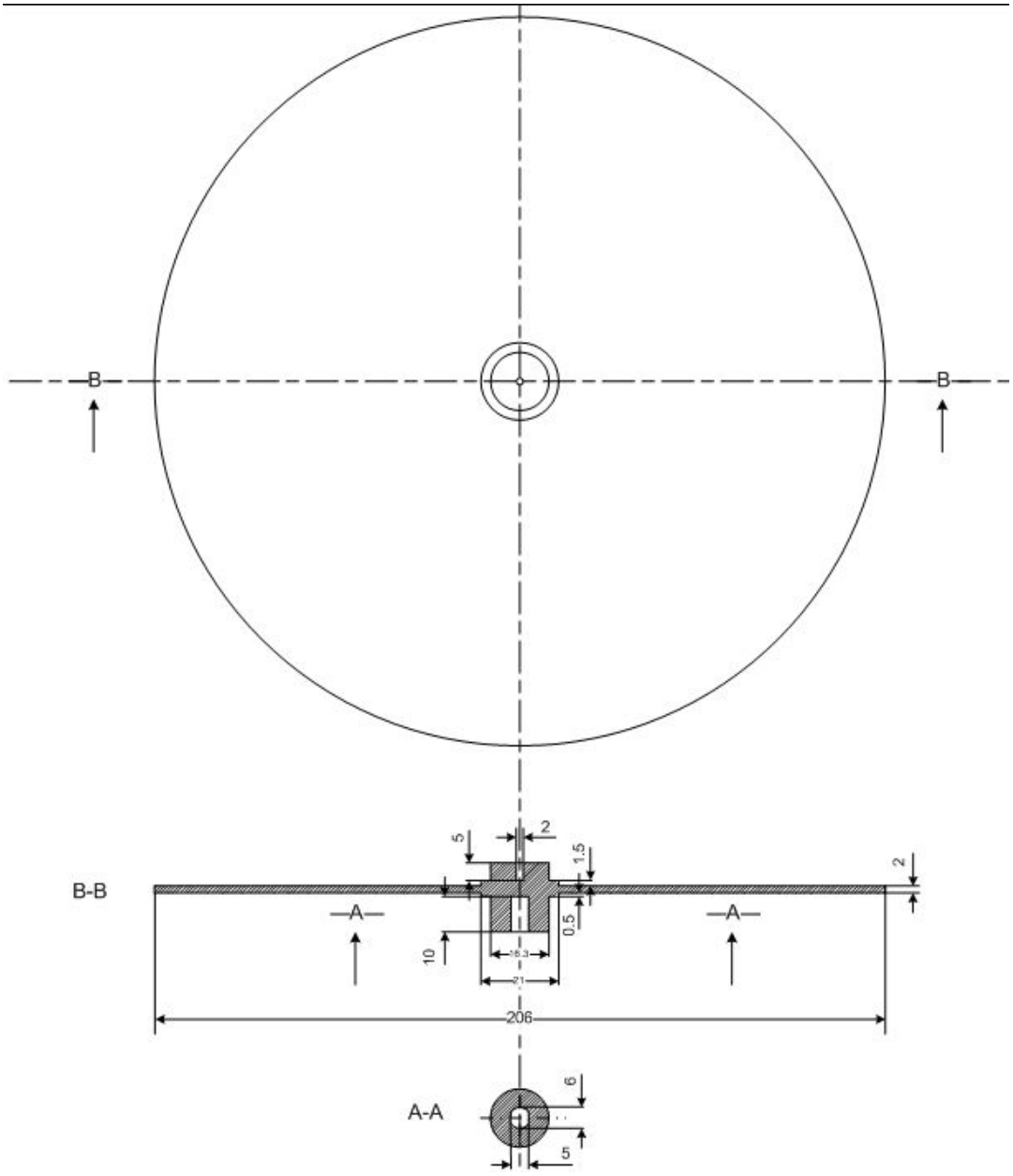
Принципни механически и блокови схеми

В тази глава са обяснени и представени нагледно основните логически блокови схеми. Включени са също описания и чертежи на механичните части.

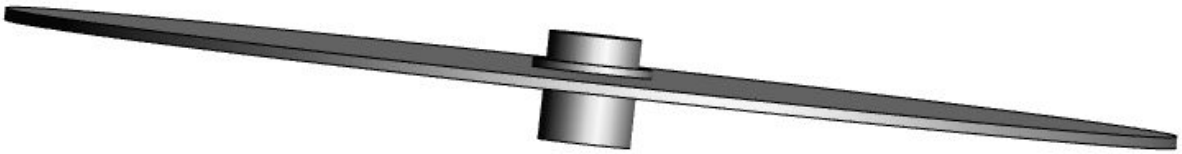
3.1. Механически схеми

Механиката на разработката съдържа няколко основни компонента.

На **фиг 3.1** е показано ротационното подвижно дъно на диспенсъра (върти се по посока на часовниковата стрелка). На изгледа, показващ сечение по оста B-B, се вижда дебелината на плочата, както и дълбочината на отвора, в който се захваща постояннотоковият двигател. На разреза по оста A-A се вижда формата на захват на мотора.



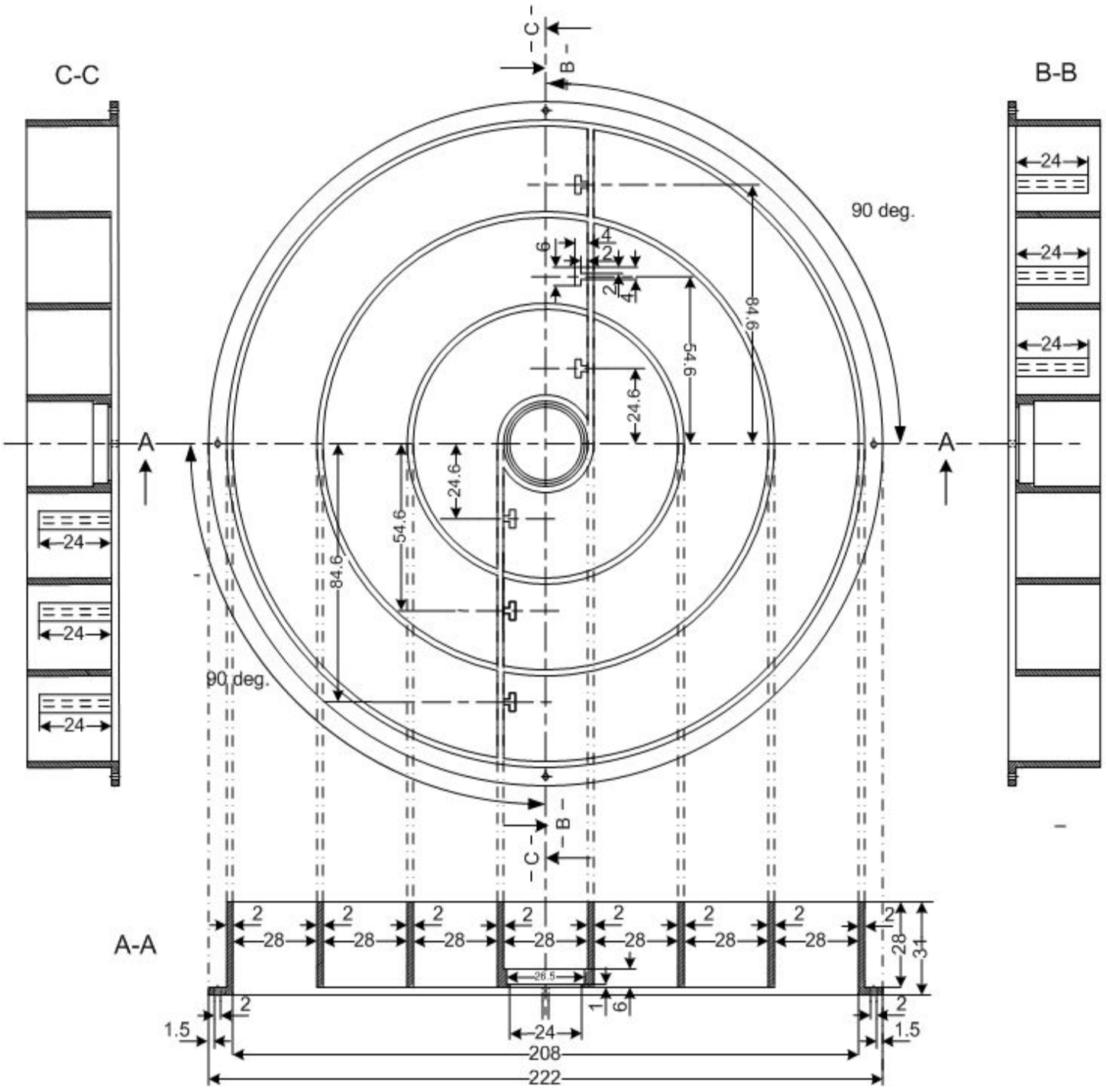
фиг 3.1
Въртящо се дъно



фиг 3.2
Въртящо се дъно - 3D модел

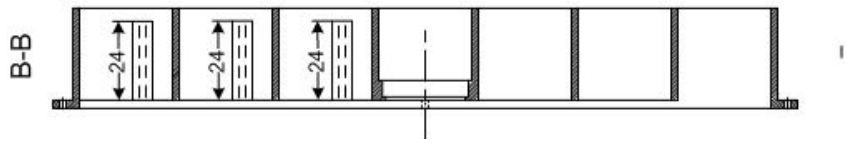
Върху ротационната основа стои разделителят, посредством който лекарствата се разпределят по видове.

На **фиг 3.3** е показана конструкцията на преградите, отделящи различните видове медикаменти. Максималният брой видове е 6 (осъществено е с помощта на 3 концентрични кръга, всеки от които е разделен на 2). Центърът е точката, около чиято ос се върти целият основата на механизма. Сечения С-С и В-В представят размерите на т-образните захвати за допълните елементи, докато А-А - размерите на контейнерите и на гнездото за лагера.**(фиг 3.3)**



фиг 3.3

Разделителни прегради



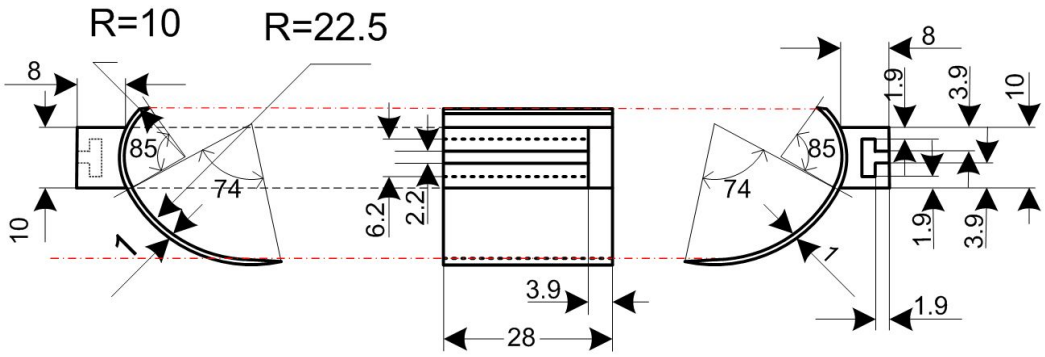
фиг 3.4
Дължини на преградите

Важно е да се отбележи, че и при трите сечения средните прегради са с 3мм по-къси от крайните (на **фиг 3.4** е показан пример с разрез B-B). Благодарение на това, дъното (**фиг 3.1**) може да се върти успешно, без опасност лекарство да премине под преградата.

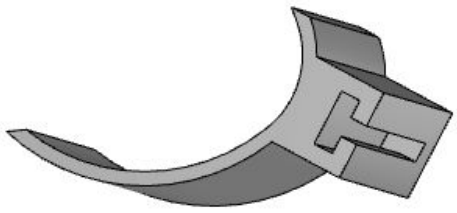
На **фиг 3.5** е показан 3D модел на преградите. Със стрелка е посочен изходният контейнер. Той служи за начална позиция на вакуумната сонда. Вземането на всеки медикамент започва и завършва от това място (при завършване, избраното хапче е пуснато в контейнера, за да достигне до пациента).

фиг 3.5
Прегради

Т-образните захвати (общо 6) са разпределени по правите, разделящи окръжностите. В двете половини на окръжностите са поставени общо 6 т-образни захвата, така че след поставяне на детайлите от **фиг 3.6**, **фиг. 3.7** медикаментите от различните контейнери да се събират в 1 права (диаметъра на окръжностите).



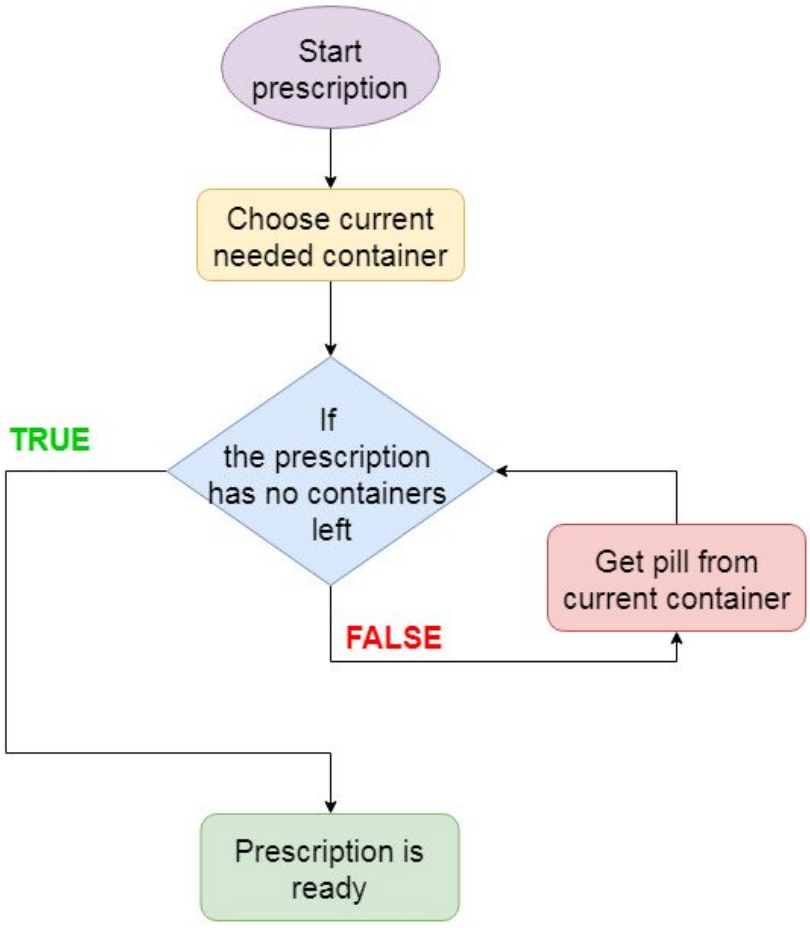
фиг 3.6
Приставки с т-образен захват



фиг 3.7
Приставки с т-образен захват - 3D модел

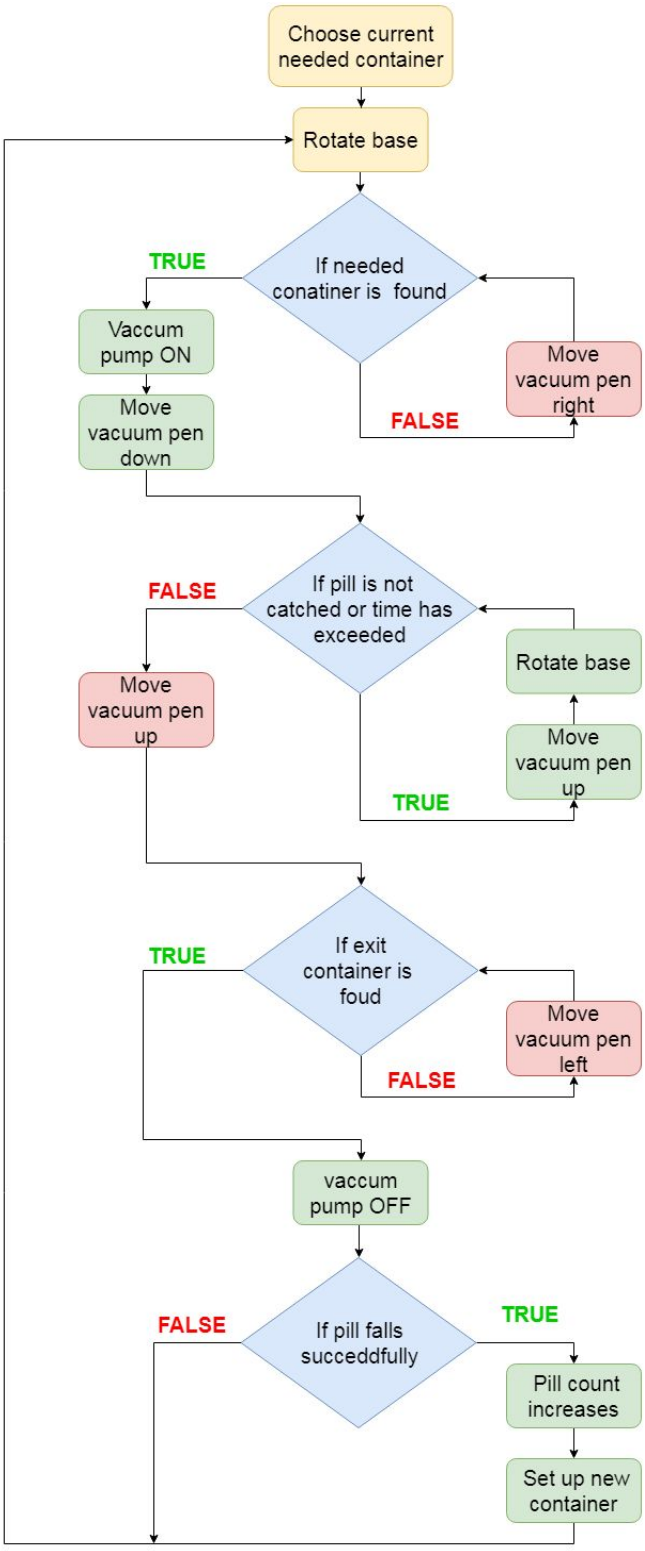
Линейното движение на сондата (хоризонтално и вертикално) е реализирано посредством ходов винт (шпилка) и стоманен стабилизиращ водач. За намаляване на триенето, двата края на винта са прикрепени чрез лагери към съответните статични опори.

3.2. Логически схеми



фиг 3.8
Главна функционалност

На **фиг 3.8** е показана основната логическа схема за изпълнение на рецепти. Всяка рецепта съдържа даден брой контейнери (във всеки контейнер се се слага различен вид лекарство). Рецептата се изпълнява докато не бъдат обходени успешно всички контейнери (докато не се изпълни успешно вземането на едно хапче за всеки един контейнер в рецептата).



фиг 3.9
Вземане на едно хапче

На **фиг 3.9** е представена логическа схема на алгоритъма за вземане на едно хапче. Първо, се избира с кой от шестте контейнера ще се работи (една рецепта представлява поредица от контейнери, всеки от които трябва да се обходи). Винаги преди начало на изпълнение се завърта основата, за да е сигурно, че контейнерите ще имат хапче в правилната позиция.

Вакуумната сонда се движи по хоризонталата докато не бъде достигнат нужния контейнер. Когато условието е изпълнено, тя започва да засмуква въздух и се спуска по вертикалата (надолу).

В случай, че хапче не бъде засмукано, механизмът се издига нагоре и основата се завърта (по този начин лекарство ще застане на нужната позиция). След успешно вземане на лекарство засмукващият накрайник се издига.

Започва движение по хоризонталата (в посока обратна на досегашното движение, за да се достигне до изходния контейнер). Когато е в правилната позиция, вакуумната помпа се изключва, за да бъде пуснато хапчето. Ако стъпката е успешна, контейнерът се счита за успешно обходен и се преминава на следващия.

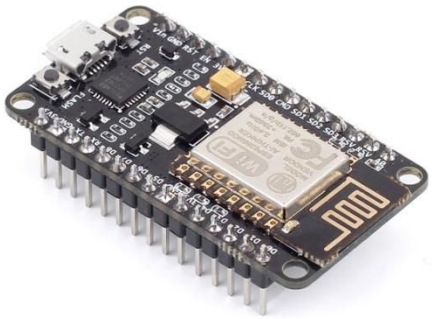
Алгоритъмът се повтаря докато не бъде изпълнена цялата рецепта. **(фиг 3.8).**

4.

Избор на елементи

NodeMCU ESP8266 development board

NodeMCU е изграден с ESP8266 базиран WiFi модул ESP-12E. Бордът има 10 GPIO порта, 4MB Flash, два бутона, micro USB конектор и вградена PCB антена. GPIO портовете могат да се използват и като PWM, 1-Wire и I2C.



фиг 4.1

Платката е с размери 48mm x 25mm, което я прави практична за монтаж. Захранва се с напрежение от USB 5V или от външен захранващ източник между 5V и 12V. Работното напрежение на GPIO портовете е 3,3V.

Модулът се програмира по сериен интерфейс с помощта на интегрирания USB-сериен порт конвертор CP2102. NodeMCU намира приложение при

експериментиране и изграждане на Internet of Things (IoT) проекти и прототипи, което го прави подходящ за разработката, описана в дипломната работа. **[1]**

Шифт регистър 74HC165

фиг 4.2

Модулът NodeMCU не предлага достатъчно пинове за изпълнението на разработката. Начинът, по който могат да се добавят GPIO портове е чрез шифт регистри. Поради нуждата от четене на информация от оптрони и сензори е избран 8-битовият 74HC16. Нужното захранващо напрежение е между 2V и 6V. Входовете използват паралелно предаване на данни, а изходите - серийно. Предимствата на този метод за увеличаване на броя на изходите на програмен модул са, че е евтино, лесно се имплементира и най-важното - заради начина на връзване на регистрите (погледнете 5 глава) могат да се използват много регистри с използването на само 4 GPIO пина **[2]**

Оптрон

фиг 4.3

За разработката бяха използвани **RPI-579** оптрони с въздушна междина и с изход npn фототранзистор. Моделът има вграден филтър за видими лъчи (светлоизточникът е инфрачервен диод, което позволява работата на оптрона да не се влияе от външна светлина). **[3]**

Микроключе

фиг 4.4

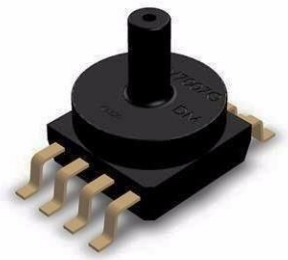
Като крайни изключватели са използвани микроключета, свързани в нормално отворено състояние на контактите.

Реле THA-3

фиг 4.5

Беше избрано постояннотоково реле THA-3 с напрежение на управляващата верига 3.3V, а на работната - 5V.

Сензор за налягане



фиг 4.6

В разработката е използван MPXA6115A сензор за налягане, чрез който се следи потокът на въздух, засмукващ хапчетата. Според промяната на въздушната струя се следи дали предмет е засмукан или не. Елементът работи на напрежение от 5V. Устойчив е на влажност и работи в широк температурен интервал (между - 40 °C и +125 °C). Сензорът има добра точност (в среда между 0 °C и 85 °C максималната грешка е 1.5%). **[4]**

DC мотори

- Редуктор GM9 (143:1)

фиг 4.7

Моторът работи с напрежение 6V, при което изпълнява 78 RPM (обороти в минута, когато е ненатоварен). Консумацията на ток е 50mA (ненатоварен) и достига до 700mA (при запълване). Тягата на мотора е 5500 гр*см **[5]**

- Мотор без редуктор(6)

фиг 4.8

Използван е моделът RM2, може да замени стандартния използван при редукторитеGM2/3/8/9 като ще увеличи скоростта тройно, тягата двойно, но и ще консумира четири пъти повече енергия. **[6]**

Драйвер за DC мотор

фиг 4.9

TB6612FNG е интегрална схема, контролираща поведението на DC мотори. Захранва се с максимално напрежение от 15V. Начинът на работа е в готовност (икономично използване на мощност). Схемата поддържа и автоматично изключване при прегряване. Драйверът изпълнява следните функционалности:

- Контрол на мотора по часовниковата стрелка (CW)
- Контрол обратно на часовниковата стрелка (CCW)
- Спиране (stop function)
- Една схема поддържа контрола на максимум 2 мотора. **[7]**

Вакуумна помпа

фиг 4.10

В разработката е използван модел **LY370CPM,** който работи при напрежение 6V. Входният ток е под 420mA. Максималното налягане е над 400mmHg, а минималният

постигнат вакуум е -250mmHg. При 30 см разстояние шумът, излъчван от помпата е под 50dB. Дължината на живот е 100 000 завъртания на мотора. **[8]**

5.

Принципна електрическа схема и печатна платка

5.1. Принципна електрическа схема

Шифт регистри

В приложената принципна електрическа схема отбелязано с U4 е NodeMCU борда. За единственият аналогов пин е свързан сензорът за налягане.

С U1 и U22 са номерирани шифт регистрите SN74HC165. За управление се използват 4 GPIO пина от контролера:

- **SH/LD** (пин 1) - При ниско ниво (LOW) паралелните данни от входовете D0 до D7 се зареждат в регистъра асинхронно. Когато SH/LD е във високо ниво (HIGH), данните влизат в регистъра серийно на входния QH като данните от всеки вход изместват предходно записаната поредица с 1 позиция.
- **CLK INH** (пин 15) - позволяване на часовника. За да може сигналът да бъде предаден серийното към NodeMCU борда, този пин трябва да е в ниско ниво (LOW). Когато се изпълнява shift на информация е добра практика пинът да бъде във високо ниво .
- **CLK** (пин 2) - часовник - При преминаването от ниско във високо ниво се изпълнява серийно предаване на данните, т.нар. shift. (едно преминаване изпълнява shift на 1 бит информация, което отговаря на един паралелен вход на регистъра.)
- **QH** (пин 9) - изходен сериен пин - Информацията се предава предава серийно към NodeMCU борда чрез този пин. При нуждата от използване на няколко регистъра QH се връзва към SER (пин 10) - "входен сериен пин". По този начин

Информацията се предава серийно през всички регистри докато стигне до първия (регистъра, чийто QH пин е свързан към програмния модул).

Повече информация за начинът на работа на шифт регистъра може да бъде видяна на таблицата на **фиг (5.1)**.

INPUTS			FUNCTION
SH/$\overline{LD}$	CLK	CLK INH	
L	X	X	Parallel load
H	H	X	No change
H	X	H	No change
H	L	↑	Shift(1)
H	↑	L	Shift(1)

(1) Shift : Content of each internal register shifts towards serial output Q_H. Data at SER is shifted into the first register

фиг 5.1
Функционалности на шифт регистър

Оптрони

Фототранзисторите, използвани при направата на оптрони, работят като оптоелектронен преобразувател (при тях базата не е свързана към електрически източник, а преобразува светлинен сноп в ток).

В приложената схема 7 npn фототранзистора са със свързан колектор към 5V (през 300**Ω** резистор). Всеки емитер е свързан към 2К**Ω** pull down резистор . Така се предотвратява висящо състояние, когато транзисторът е запушен и подаването на неточен сигнал към шифт регистъра. Всеки транзистор отговаря на един контейнер и в нормално състояние е запушен. Чрез инфрачервен диод, който е прикачен към механизма, движеш сондата по хоризонталата, при достигане до нужния контейнер, транзисторът бива осветен, а на съответния пин от регистъра се прочита 1.

За проверка дали хапче е паднало успешно през изходния контейнер, са използвани 6 последователно свързани оптрона (емитерът на един транзистор е свързан към колектора на следващия). По този начин един засегнат оптрон е достатъчен, за да бъде подадена 0 на съответния пин от регистъра (пин D0 на U22).

Микроключета

В разработката са използвани две микроключета (SW1, SW2), които служат за проверка на пространственото положение на сондата при вертикално движение. Свързани са в нормално отворено състояние съответно към пиновете D7 на U1 и D1 на U22.

Драйвери за мотори

Трите DC мотора се контролират с помощта на два драйвера за мотори (U5, U6). Пиновете за контрол, свързани към NodeMCU борда са следните:

- STBY - (пин 19) - Standby - Чрез този пин моторите на дадения драйвер преминават в Standby състояние (в готовност). Моторите могат да работят само когато Standby е във високо ниво. За тази разработка беше решено STBY пинът да бъде свързан към постоянно напрежение 5V.
- AIN1,AIN2 - (пин 21, 22) - Това са входни пинове, на които отговаря съответно AO1 и AO2.
- BIN1, BIN2 - (пинове 17, 16) - Аналогични на горепосочения, но подават състояния на съответно BO1 и BO2.
- PWMA, PWMB - (пинове 23, 15) - Всеки от трите свързани Pulse Width Modulation пина са свързани към различни GPIOs от NodeMCU борда. По този начин се работи само с желания мотор.(повече информация за PWM в **6. Глава**).

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

фиг 5.2

Функционалности на драйвер за мотор

Освен изходните пинове , за които се свързват моторите, желаното работно напрежение за двигателите се подава съответно през VM пиновете (2.5 V~13.5 V) и PGND пиновете.

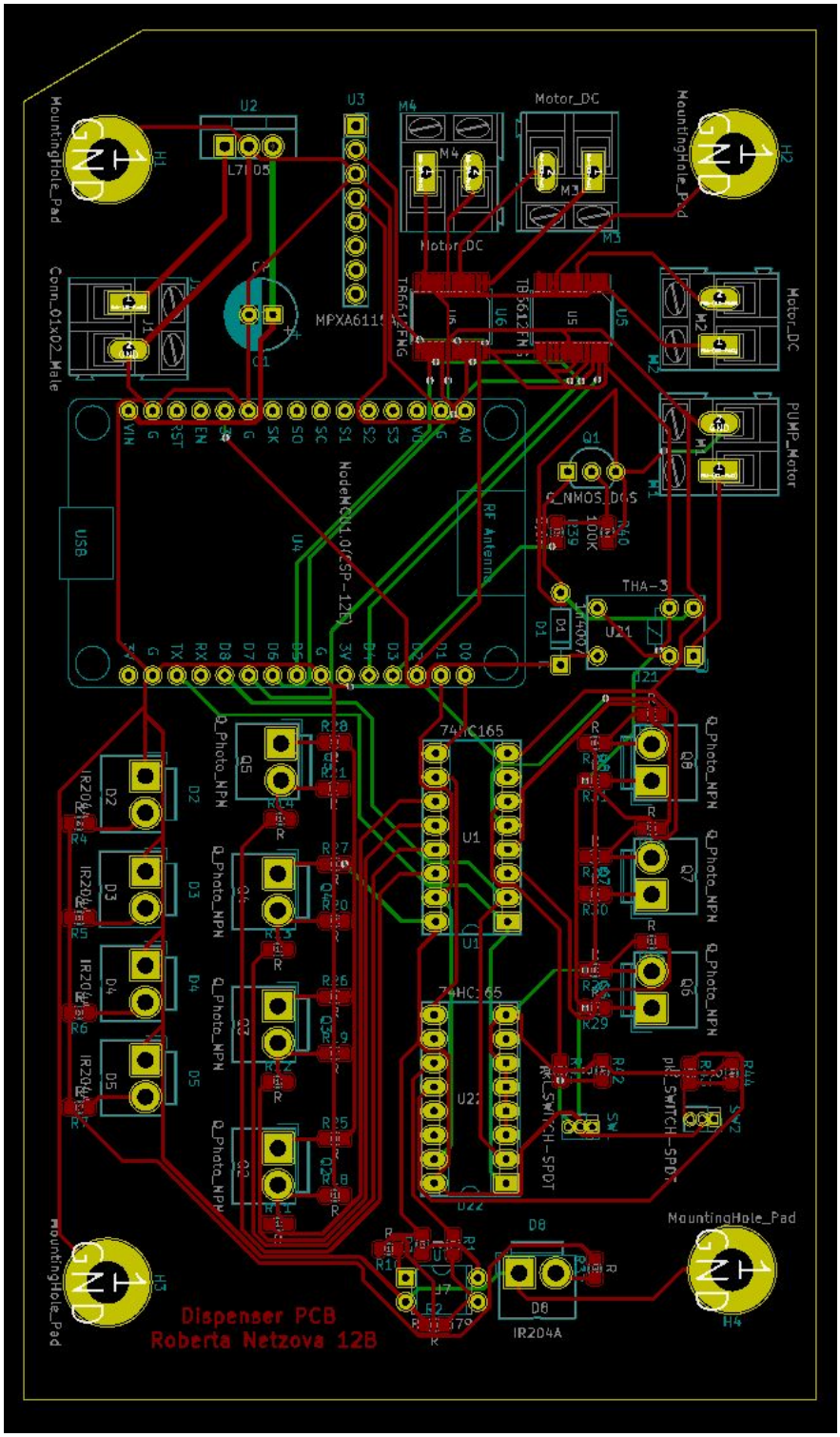
Забележка: Начинът на свързване на драйверите не позволява два мотора да работят едновременно, поради което вакуумната помпа е управлявана чрез реле и транзистор.

Управление на вакуумна помпа

За управлението на вакуумната помпа бяха използвани NMOS транзистор (Q1) и постояннотоково реле (U21). Паралелно към намотката е свързан диод, който предпазва от преминаване на обратни токове по време на комутация. Работната част на транзистора (дрейнът) е свързана последователно на намотката. При подаване на високо ниво от пина D3 на NodeMCU борда към гейта, през дрейн-сорса протича ток, веригата NMOS транзистор-управляваща намотка на релето се затваря, в следствие на което се подават 5V на статора на помпата

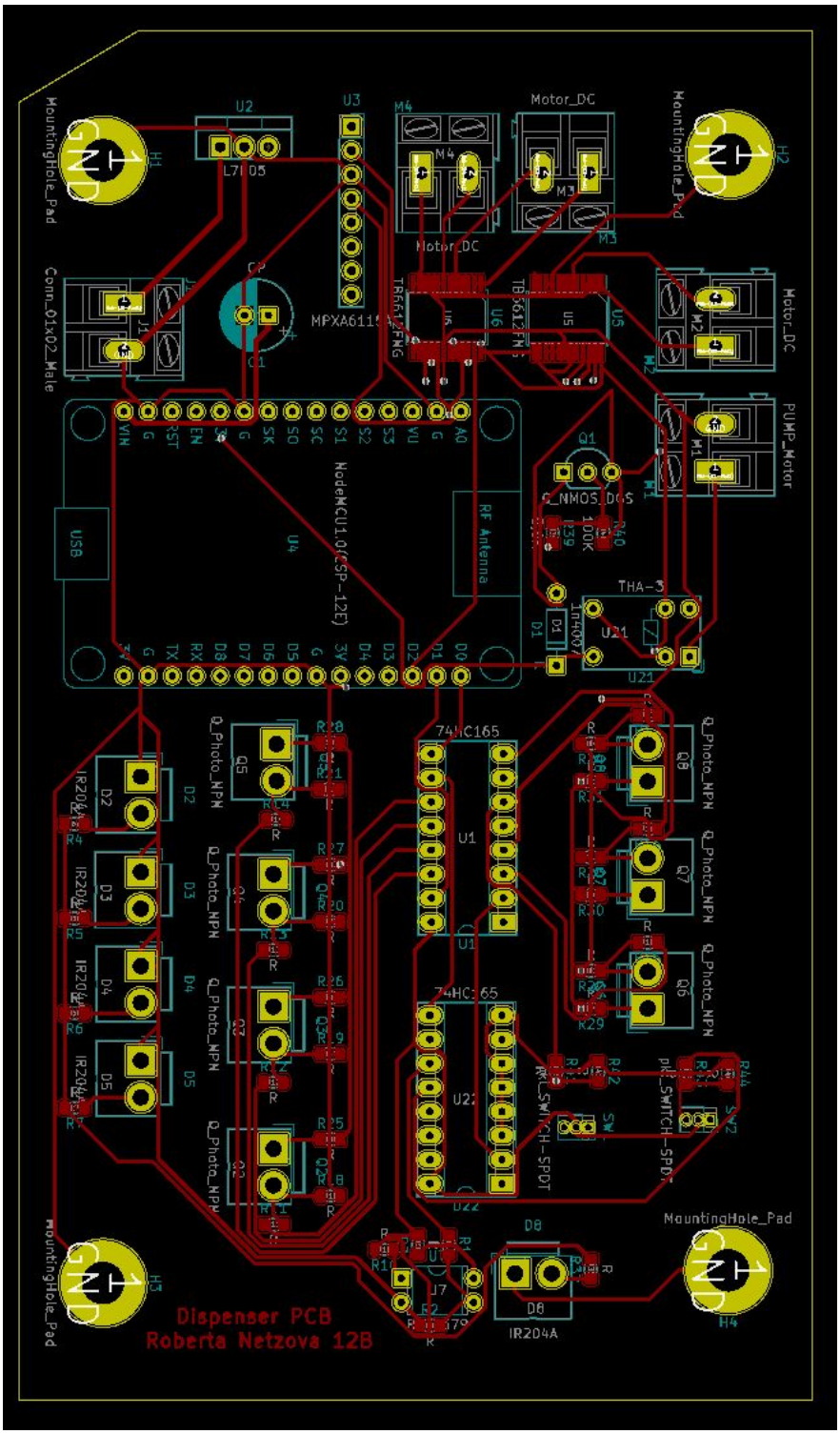
5.2. Печатна платка

На **фиг. 5.3** е изобразена печатната платка, използвана за разработката. Както се вижда бяха използвани два слоя. Пистите, отбелязани в червен цвят отговарят на горния слой, докато тези в зелено - на долния. За улеснение при по-нататъшна работа беше решено почти всеки елементи да бъде THT. За предотвратяване от объркване при работа, едно от ъгловите очертания е скосено. Тъй като някои елементи, изобразени на принципната електрическа схема, са разпределени на различни места по машината, е използван footprint на терминали. Съответно е съобразено те да бъдат разпределени близо до ръбовете на платката. В четирите ъгъла са добавени монтажни отвори с големина M3. На **фиг.5.4** е показан само горният слой, който видимо е по-често използван. На **фиг.5.5** е изобразен долният слой.



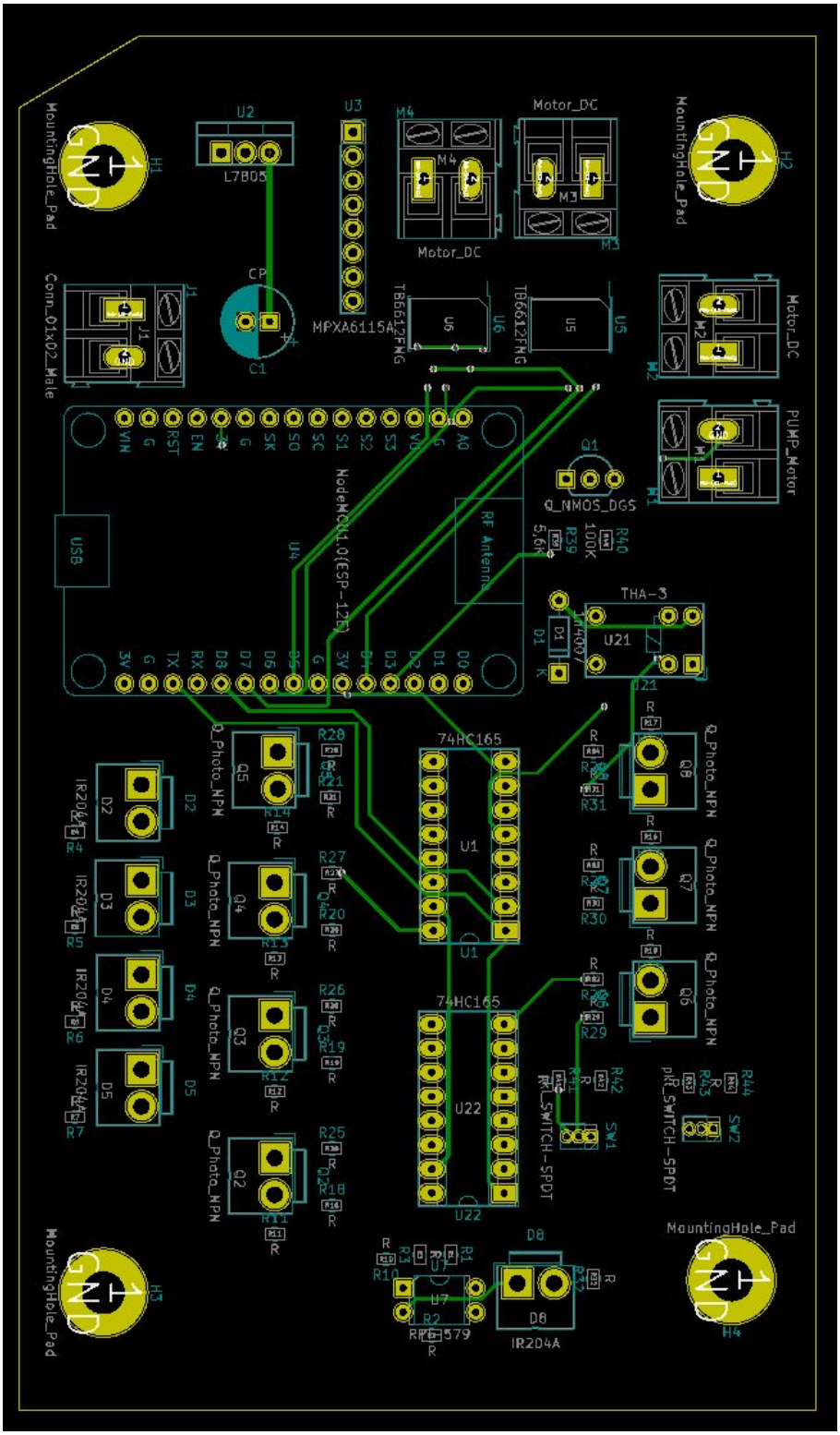
фиг 5.3

Печатна платка - двуслойна



фиг 5.4

Печатна платка - горен слой



фиг 5.5

Печатна платка - долен слой

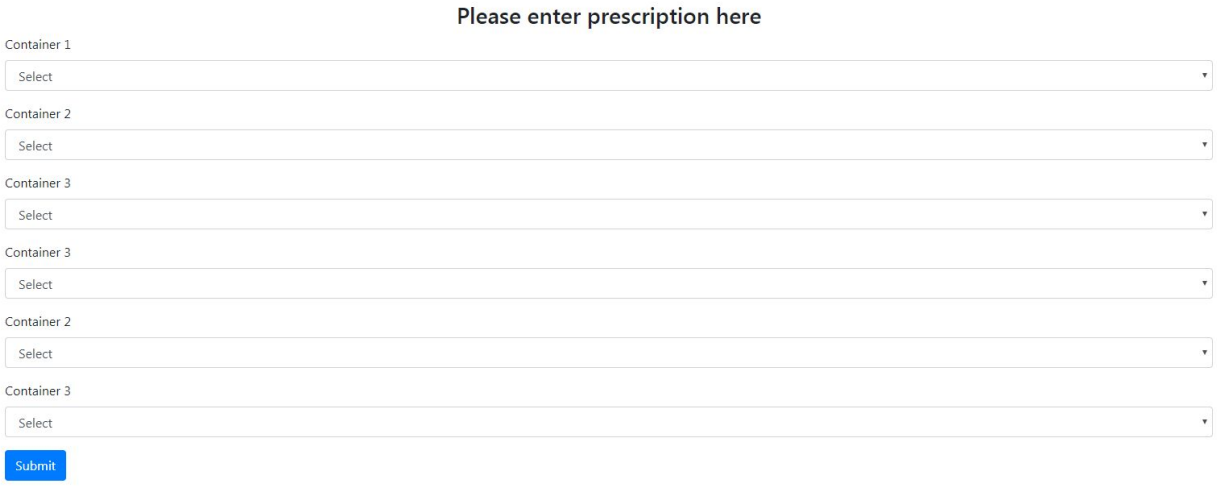
6.

Софтуерно обезпечаване

В тази глава са разгледани ключовите софтуерни моменти от разработката.

6.1. Уеб приложение

В индексната jade страница index.jade беше написана форма за създаване на рецепта. Тя представлява 6 менюта с избираем отговор (за всеки контейнер). Страницата е показана на **(фиг 6.1.)**



фиг 6.1
Изглед на формата за рецепти

За разработката беше прието, че един тип лекарство може да бъде поръчано в максимално количество 3 хапчета.

```
form(method='get', action='/getPrescription')
           .form-group
                label Container 1
                select.form-control(name = 'cont1')
                     option(value = '0') Select
                     option(value = '1') Count: 1
                     option(value = '2') Count: 2
                     option(value = '3') Count: 3
```

фиг 6.2
Jade полета на избираемото меню

На **фиг 6.2** може да се види, че стойностите на option полетата съвпадат с броя искани хапчета. Формата се повтаря за всичките 6 контейнера.

След попълването й се изпраща GET request на разклонението /getPrescription, който се обработва от route файл (/routes/index.js).

```
router.get('/getPrescription', function(req, res) {
     console.log (req.query);

     console.log (req.query.cont1);

     // creating new child process
    const { execSync } = require('child_process');

    let container1 = execSync(
     '/usr/bin/mosquitto_pub -t dispenser_server -m "cont:1:' +
      req.query.cont1 + '"');
```

фиг 6.3
Метод за обработване на изпратените данните.

Главната функционалност на метода от **фиг 6.3** е изпращане на съобщение към Mosquitto брокера. За да се задават команди на операционната система, беше създаден child process. Функцията за изпълнение на командите, execSync(), е синхронизирана, с цел последователно изпълнение на функциите за шестте контейнера).

Променливата container1 е декларирана с let, като по този начин се предотвратява преизползване на променливата в други блокове в състава router.get функция.

Командата за публикуване изисква mosquitto_pub с флаг за комуникационната тема (dispenser_server). Съобщението е съставено от номера на контейнера и полученото query от полето на формата с име cont1. Например ако бъде избрано да се вземат 2 хапчета от контейнер 1, execSync() ще публикува следното съобщение: **/usr/bin/mosquitto_pub -t dispenser_server -m "cont:1:2"**

Всеки контейнер изпраща отделно съобщение със своята информация.

6.2. Комуникация на NodeMCU с приложението

Свързване с интернет

За осъществяване на интернет връзка е използвана библиотеката **WiFiManager**. Предимството при нея е, че е удобна за създаване на връзка с WiFi без нужда за статично въвеждане на мрежовите параметри (SSID и парола). По този начин се предотвратява постоянно пренаписване и качване на нов код на ESP8266 (в случай на смяна на мрежата за връзка).

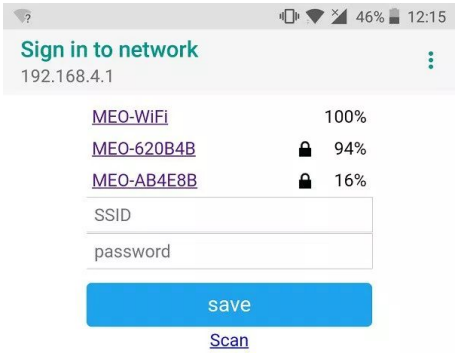
```
WiFiManager wifiManager;
wifiManager.autoConnect("AutoConnectAP");
```

фиг 6.4

Създаване на автоматична връзка

На **фиг 6.4** е показана функцията за създаване на автоматичен достъп.Чрез нея при стартиране на NodeMCU борда, ESP чипът се връзва с вече конфигурирана и достъпна мрежа. В случай, че такава не е намерена, се създава нова точка за достъп (Access Point - AP) - в случая с име **“AutoConnectAP”.**

Веднъж свързали се към новата точка за достъп можем да се отиде на дефолтно генерирания IP адрес. Отваря се уеб страница с достъпни мрежи. **(фиг 6.5)**



фиг 6.5
Достъпни мрежи

Въвеждат се данните на желаната и по този начин е създадена новата точка за достъп с мрежата. Всяка връзка се пази в паметта на чипа и и след рестарт на борда.(т.е този процес на конфигурация се изпълнява само веднъж) конфигурационни данни и се добавя нова връзка и ESP се свързва с нея. **[17]**

Mosquitto връзка

На **фиг 6.6** е показан потокът на предаване на съобщения между диспенсъра и уебсайта. Работи се по зададеното тема (“dispenser”). Приложението играе ролята на publisher, изпращайки информация по MQTT протокол (използвана е функцията execSync(), описана в **точка 6.1**). Mosquitto брокера публикува съобщението. Диспенсърът е subscriber - той слуша за публикувани съобщения на същата тема.

фиг 6.6
Връзка между диспенсър и приложение

За осъществяване на комуникация между Mosquitto брокера и NodeMCU борда е нужна следната информация за създаване на клиент: хост, порт (при MQTT по подразбиране е 1883), ID на клиента и теми за входни и изходни съобщения. В setup() метода параметрите се подават на setup функцията от класа Mosquitto (**фиг 6.7**)

```
  mosquitto.setup(mqtt_host, mqtt_id, mqtt_port, mqtt_topic_in,
mqtt_topic_out);
```

фиг 6.7
setup() функция

В главния метод, loop(), се изпълнява read() функцията (**фиг 6.8**), която проверява дали връзката между създадения клиент и хоста е осъществена успешно. (**фиг 6.9**).

```
void loop() {
 if(! mosquitto.read()) Serial.println("Failed to read !!!");
}
```

фиг 6.8
loop() функция

read() извиква метода connection_checkup() от Mosquitto.cpp (**фиг 6.9**). При грешка се изпълнява reconnect() (**фиг 6.9**). При успех се вика готовият метод loop() (служи за обработка на входящите съобщения и за поддържане на връзката със сървъра.). В противен случай се връща съобщение “Failed to read !!!”. (**фиг 6.8**).

```
bool Mosquitto::connection_checkup () {
      if ( ! _mqtt_client.connected() ) reconnect();
      if ( ! _mqtt_client.connected() ) return false;
      return true;
  }
bool Mosquitto::read()
{
  if (! connection_checkup()) return false;

  return _mqtt_client.loop();
}
```

фиг 6.9

Функция за проверка на Mosquitto връзка

При пристигане на нови съобщения се извиква Callback функция. Тъй като клиентът използва един буфер за входящи и изходящи съобщения, стойности, нужни на програмата, трябва да се запишат в собствено копие. За целта в разработката е използван методът **mqttCallback (char* topic, byte* payload, unsigned int length).** Посредством съдържанието на съобщението се извиква getPill(int container) с аргумент нужният контейнер. Извиква се толкова пъти, колкото е било получено в съобщението.

6.3. Контрол на хардуерните компоненти

Контрол на мотори

За контролирането на трите DC мотора беше направен клас на C++ (motorDriver.cpp), който да изпълнява главните функционалности, нужни за контролиране на скоростта, продължителността и посоката на движение на двигателите.

Инстанция на класа Motor съдържа 3 променливи (те са дефинирани пинове, свързани с NodeMCU). Добавената в "motorDriver.cpp" библиотеката "Arduino.h" позволява да бъдат използвани готови Arduino функции. Инициализацията на пиновете се случва в конструктора на класа вместо в setup функцията (Този начин на инициализация е използван за контрола на всички хардуерни компоненти).

Пиновете, които се инициализират са In1, In2 (те отговарят за посоката на въртене, която може да е по или обратно на часовниковата стрелка) и PWM (pulse width modulation - чрез поредица от импулси с различна продължителност се определя скоростта на въртене на мотора). PWM се контролира чрез функция от "Arduino.h" (analogWrite(pwmPin,frequency)). Достъпните до главния Arduino файл са методите:

- drive(speed) - включва моторите като посоката може да бъде зададена положителна или отрицателна в зависимост от желаната посока.
- brake() - спира мотора
- drive(speed, duration) - Спира мотора след изтичане на определеното работно време.

Контрол на шифт регистри

Както беше споменато в 4 глава, шифт регистърът се управлява от четири GPIO пина. За създаването на инстанция на класа Shift_165 се извиква конструктор със следните полетата enable, load, clock и data, които отговарят съответно на CLK INH, SH/LD, CLK, и QH на шифт регистъра. За предаване на данните е използвана готовата функция shiftIn(data, clock, MSBFIRST).

Контрол на сензор за налягане

Сензорът за налягане, за разлика от другите компоненти, работи с аналогови данни. Чрез Arduino функцията “analogRead(pin)” се чете състоянието на сензора.

Други интересни методи

Единствено контролът на базовия мотор и помпата могат да бъдат контролиран директно в метода getPill(int container), тъй като нямат нужда от проверки, а работят само за определено време.

За извършване на хоризонтално и вертикално движение в посока нагоре (вертикалното движение използва този метод, защото за спиране на мотора се използва ключе в нормално отворено състояние) е използван методът **void movePen(int speed, int container, Motor motor).** Функцията реализира движение на съответния мотор до прочитане на 1 от страна на регистъра.**(фиг. 6.9)**

```
while (shifted[container] != 1) {
   shifted = shift_register.Shift_bytes();
   Serial.print(shifted[container]);
   yield();
   }
```

фиг 6.10

Проверка за състоянието на регистъра

Функцията yield() на **фиг. 6.10** е част от библиотеките за ESP8266, като използването й е нужно в случаи, когато функционалностите, извършвани във фонов режим (например WiFi свързаност) биват блокирани , което води до нежелано рестартиране на борда. Методът позволява по време изпълнение на блокиращия код, фоновите функции да продължават нормалната си работа.

Хоризонталното движение надолу на сондата е контролирано, благодарение проверка за състоянието на сензора за налягане. При промяна на налягането, се разбира, че хапче е засмукано, и се предприема смяна на посоката на мотора.

Заключене

В представеният проект беше измислена и изработена механика, позволяваща вземането на един медикамент, отговарящ на стандартните изисквания за големина на лекарства. С помощта на програмируем борд беше успешно осъществено засмукването на множество хапчета (изпълнение на рецепти до максимум 6 вида различни медикаменти). За улеснение на потребителя беше добавена връзка към уеб сървър и беше разработен улеснен уеб сайт за поръчване на медикаментозни дозировки в реално време.

За бъдещо развитие е планирано към интерфейсът да бъде добавено мобилно приложение с възможност за създаване на поръчки, отложени във времето. Следващата стъпка е машината да поддържа повече потребители и да изпраща предупреждения в случай на възникнала грешка (пр: Неизпита дозировка).

Съдържание

УВОД **4**

Обзор на съществуващи решения **5**
Въведение в проблема, с който се среща дипломната 5
Решения на проблема 5
MDS & MCA 5
Автоматизирани диспенсъри 6
Pivotell® Mk3-11 Диспенсър 6
Pivotell Advance GSM диспенсър за хапчета с Приложение за известяване 7
Friendly Locked e-pill MedSmart VOICE Automatic Pill Dispenser 7
Видове механизми за вземане на дозировка 8
Разделение по дозировки 8
Разделение по медикаменти 8
Съществуващи решения, използващи метода “Разделение по медикаменти” 8
Вендинг машина за сладолед 8
Кратко описание на дипломната работа 9

Използвани технологии и развойни среди **10**
Избор на технологии 10
Node.js 10
npm 10
Express 11
Bootstrap 11
C/C++ 11
MQTT 12
Избор на развойни среди 12
Arduino IDE 12
Sublime text 12
Сървър 13

Принципни механически и блокови схеми **14**
Механически схеми 14
Логически схеми 21

Избор на елементи **24**
NodeMCU ESP8266 development board 24
Шифт регистър 74HC165 25
Оптрон 26
Микроключе 26
Реле THA-3 27
Сензор за налягане 27
DC мотори 28
Редуктор GM9 (143:1) 28
Мотор без редуктор(6) 28
Драйвер за DC мотор 29
Вакуумна помпа 29

Принципна електрическа схема и печатна платка **31**
Принципна електрическа схема 31
Шифт регистри 31
Оптрони 32
Микроключета 33
Драйвери за мотори 33
Управление на вакуумна помпа 34
Печатна платка 35

Софтуерно обезпечаване **39**
Уеб приложение 39
Комуникация на NodeMCU с приложението 41
Свързване с интернет 41
Mosquitto връзка 42
Контрол на хардуерните компоненти 45
Контрол на мотори 45
Контрол на шифт регистри 45
Контрол на сензор за налягане 46
Други интересни методи 46

Заключене **47**

Съдържание **48**

Използвана литература **51**

Използвана литература

[1] https://www.medscape.com/viewarticle/501879

[2] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4122419/

[3] https://www.pivotell.co.uk/

[4] https://www.epill.com/medsmartvoice.html

[5] http://iconicinventions.com/icecream/

[6] https://stackoverflow.com/questions/31930370/what-is-npm-and-why-do-i-need-it

[7] https://stackoverflow.com/questions/12616153/what-is-express-js

[8] https://www.geeksforgeeks.org/c-language-set-1-introduction/

[9] http://www.bestprogramminglanguagefor.me/why-learn-c-plus-plus

[10] https://erelement.com/wireless/nodemcu

[11] https://www.sparkfun.com/datasheets/IC/SN74HC595.pdf

[12] http://80.93.56.75/pdf/0/4/6/9/4/04694054.pdf

[13] https://www.nxp.com/docs/en/data-sheet/MPXA6115A.pdf

[14] https://www.robotev.com/product_info.php?cPath=1_40_32&products_id=76

[15] http://www.longykj.com/en/productshow.asp?ArticleID=249

[16] https://www.pololu.com/file/0J86/TB6612FNG.pdf

[17] https://randomnerdtutorials.com/wifimanager-with-esp8266-autoconnect