**СОФИЙСКА ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО ЕЛЕКТРОНИКА**

**„ДЖОН АТАНАСОВ“**

**ДИПЛОМЕН ПРОЕКТ**

**Тема: „Система за контрол на достъпа на служители“**

**Практическа част: „Реализиране на система за контрол на достъпа на служители“**

Дипломант: Филип Ивайлов Андонов 12 „В“ клас

Професия: 481030 „Приложен програмист“

Специалност: 4810301 „Приложно програмиране“

Ръководител на дипломен проект: инж. Любица Димитрова

Дипломант:

*/подпис/*

Ръководител на Дипломен проект:

*/подпис/*

София

2022

# **Съдържание**

[**Съдържание** 2](#_Toc101808847)

[**Теоретична част** 3](#_Toc101808848)

[**Увод** 3](#_Toc101808849)

[**Изложение** 5](#_Toc101808850)

[**Планиране** 5](#_Toc101808851)

[**Система за контрол на версиите и ресурсите** 7](#_Toc101808852)

[**Хардуерна архитектура** 9](#_Toc101808853)

[**База данни** 11](#_Toc101808854)

[**Общо за софтуерната архитектура** 14](#_Toc101808855)

[**Софтуерна архитектура на портален модул** 18](#_Toc101808856)

[**Софтуерни класове за порталния модул** 18](#_Toc101808857)

[**Софтуерна архитектура на сървърна апликация** 21](#_Toc101808858)

[**Софтуерни класове за сървърната апликация** 22](#_Toc101808859)

[**Потребителски интерфейс / потребителско преживяване** 24](#_Toc101808860)

[**Софтуерна архитектура – уеб приложение** 25](#_Toc101808861)

[**Заключение** 31](#_Toc101808862)

[**Практическа част** 32](#_Toc101808863)

[**Списък на използвана литература** 32](#_Toc101808864)

[**Други полезни линкове** 32](#_Toc101808865)

# **Теоретична част**

## **Увод**

В днешно време процесът на урбанизация се засилва все повече. Земеделието се модернизира и човешкият труд в сферата вече е все по-малко търсен. Докато средната възраст в провинцията расте, младите трудоспособни хора се преселват в големите градове и започват работа в офиси и предприятия. Според проучване на Националния статистически институт (1) в Българските градове към 31.12.2020 г. пребивават 72.9% от населението. Българското законодателство гласи, че всеки работодател е задължен да подпише трудов договор с наетия служител, като се задължава да изплаща възнаграждение. Тази заплата най-често се формира от положените работни часове. За целта всяка фирма наема счетоводители и икономисти, които да следят приходите, разходите и заплатите. Процесът, изчисляващ възнаграждението, става значително по-сложен при по-голям брой служители. Повечето предприятия нямат възможността да се заемат с точно следене на положените часове труд, затова изплащат заплатата на базата на договорено работно време. Това е нечестно спрямо полагащите повече усилия, защото на тях се изплаща толкова, колкото и на по-малко трудещите се. Непрецизния критерии води до занижаване на стандартите, продуктивността и мотивацията.

Друг актуален проблем е сигурността. Служителите и фирмената собственост имат нужда от защита против злонамерени лица. При липса на подходящи мерки е възможен фирмен шпионаж, оронване на престижа на предприятието и/или служителите, изтичане на ценни данни, кражби и саботаж. Протекцията се случва най-често чрез жива охрана, но в голяма фирма с много служители тя е неефективна и скъпа. Това е причината охранителите да работят в симбиоза със системи за сигурност като видеонаблюдение и системи за контрол на достъпа.

Целта на този дипломен проект е едновременно да спомогне подобряването на сигурността и решаването на проблема с изчислението на възнаграждение.

Системата се състои от четири основни части: портален модул, сървърен модул, база данни и уеб модул.

Порталният (клиентският) модул цели да се предотврати неоторизиран достъп през портала, на който е поставен. Той използва гъвкав заключващ механизъм, блокиращ отварянето на врата, и сензор за радиочестотна идентификация. Чрез сензора се извлича информация от чип, карта или мобилно устройство, която след това се използва при запитване за оторизация. Модулът е сравнително евтин и позволява надграждане. Това дава възможността да се използват повече от един модули с цел пълен контрол върху сградата. По този начин с много по-малко съдействие от човек и по-малък бюджет се осигурява по-сигурен периметър.

Сървърният модул получава и обработва заявки от така наречените клиентски модули. Един сървърен модул може да обслужва паралелно много заявки. Целта му е да провери в базата данни дали служителят е оторизиран да премине през портала, това действие да се запише и да се изпрати обратен сигнал до клиентския модул.

Уеб модулът е достъпен през мрежата, позволява да се анализират данните от порталите и да се извежда статистика за работно време на служителите. Изведената информация е подходяща, за да се включи в алгоритъм за изчисление на заплата.

Базата данни запазва информацията трайно в енергонезависима памет.

От този проект се очаква да намали цената на охранителната дейност, да улесни служителите, обработващи разплащанията, да спомогне за по-справедливо възнаграждение и да повиши нивото на сигурност.

Системата е иновативна за България, защото обединява софтуера и хардуера по подходящ за работодателите начин. На пазара се срещат системи за контрол на достъп и за засичане на работно време, но не се предлага цялостно решение, като описаното в този проект.

## **Изложение**

### **Планиране**

#### **Подбор на тема**

Предвид познанията и ресурсите, които имам, съставих списък с проекти. Започнах елиминирането на потенциални възможности, като анализирах дали са осъществими в дадения срок, какви ресурси ще се изискват, каква практическа стойност имат, колко са сложни и колко са оригинални. Съвкупността от многослоен софтуер и хардуер, присъща на този проект, е изключително интересна и предизвикателна, но операционната система Линукс(2) и общността поддържаща „отворен“ код (open source) позволяват завършването му в поставените срокове. Ресурсите нужни за изпълнението са скромни, защото проектът е замислен така, че да функционира надеждно върху контролери със слаби възможности и да е реализиран с евтини, често срещани компоненти. Системата може да намери реално приложение в повечето офиси и предприятия, без значение от тяхната дейност. Тя е много гъвкава, защото позволява заменяне на периферни части като ключалката, сензора и комуникационния модул с много малки модификации. Избрах системата за сигурност и контрол на достъпа, защото тя обединява две съществуващи системи в трета, която е сравнително иновативна и перспективна.

#### **Съставяне и описване на план**

За успешното изпълнение на един сложен проект е нужен подход, стремящ се към разделянето на задачите в повече на брой, но по-прости подзадачи. За целта е използван Канбан. Той е „lean“ метод, който подобрява организацията и координацията на работна група. Традиционно представлява метод, в който се залепят задачи, записани на лепящи листчета, върху голяма дъска, разделена на колони. Подходящ е за проекти, в които всичко се изпълнява от един човек, защото толерира креативност, гъвкавост и простота.

Съставянето на планa започва с нахвърлянето на много идеи за реализация на системата. Тези идеи бяха записани в текстов документ, бяха редактирани и пренесени в интерактивен план в уебсайта Trello(Фиг. 1). Платформата е подходяща, защото в нея са интегрирани модернизирани Канбан шаблони. Традиционното табло е заменено с виртуално, а листчетата с многофункционални бележки. Регистрацията е лесна, интерфейсът е прост и платформата предоставя възможност за „добавки“ (power-ups/plugins), които позволяват добавянето на интерактивен календар, спомагащ съставянето на график и следенето на срокове. Бележките са разделени в различни колони и могат да се добавят в различни групи.

Graphical user interface, application

Description automatically generatedФигура 1. Екранна снимка от платформата на Trello

Задачите за изпълнението на проекта са разпределени в пет колони:

* „To Do Planning Related“ – Съдържа всички информационни бележки, касаещи планирането, системния анализ, организацията и потенциални нови свойства (feature-и). Тази колона съдържа голяма част от задачите с най-висок приоритет, затова е съответно позиционирана на първо място. Резултат от изпълнението и е план, описващ подробно нужните стъпки за изпълнение на проекта.
* „To Do Developing Related“ – Съдържа всички информационни бележки, касаещи практическите дейности. Колоната ще доведе до изпълнение на плана, съставен чрез задачите от колона „To Do Planning Related“.
* „To Do Testing Related“ – Съдържа всички информационни бележки касаещи компонентното и функционалното тестване. Тази колона е с по-нисък приоритет, защото за нейното изпълнение е нужен готов план и компоненти, които възникват въз основа на предните две колони.
* „In Progress“ – Съдържа всяка информационна бележка, върху която се работи. Целта на колоната е да спомага концентрацията на работата върху една единствена задача, с цел да подобри ефикасността и качеството на изпълнение.
* „Done“ – Колоната съдържа всички свършени задачи. Целта и е да отдели настоящата от свършената работа, да подобри концентрацията и да даде реален поглед върху прогреса на проекта.

Бележките са сортирани в колоните по низходящ приоритет. Всяка бележка притежава категории. Категориите са:

* „Planning“ – В категорията попадат всички бележки, касаещи съставянето на план и график за проекта.
* „System Analysis“ – Категорията отговаря за анализирането на системата. Тук попада анализ за скорост и надеждност както на софтуера, така и на хардуера, анализиране на варианти между различни системни подходи и др.
* „Feature“ – Всички информационни бележки, които представляват идеи за нови добавки, които не са от съществено значение и са с по-нисък приоритет, но биха представлявали интерес, попадат в категория “Feature”
* „Hardware“ – Тук попадат всички бележки, уточняващи задачи за планирането, реализирането и тестването на хардуера.
* „Developing“ – Всяка бележка, служеща за реализацията на вече изготвен план, попада в категория „Developing“.
* „Testing“ – Категорията се отнася за всички задачи, обвързани с тестване на вече реализирани бележки.
* „DevOps“ – В категорията попадат задачите, отговорни за организацията на работата, планирането, реализацията и тестването. Всяка „DevOps“ задача улеснява изпълнението на всички други бележки.

Линк към дъската, използвана за този проект можем да намерим в секцията: „Други полезни линкове“ (3).

### **Система за контрол на версиите и ресурсите**

За целта на проекта е използвана система за контрол на версиите и ресурсите (Source control system). Тя е полезна при големи проекти, защото добавя проследяемост, сигурност и позволява паралелна разработка. Записва всяка промяна по съдържанието на проекта. Записът се нарича ревизия и често е съхраняван на сигурна платформа облак (cloud). Ревизиите могат да бъдат запазвани. В момент на нужда, потребител на системата може да върне проекта в състояние на предишна ревизия. Така при срещане на бъг, породен в нови версии, програмистът може да открие разликите в кода и това да му помогне за решаване на проблема. Този вид системи често предлагат други функционалности като разклоняване (branch), сливане(merge), контрол на достъпа(access control), заявка за промяна(feature request/bug request), документация(wikis) и други.

Разклоняването е процес, в който проектът се клонира в нов клон(new branch), паралелен на главния (master branch). Промените в новия клон няма да се отразят на главния.

Сливането позволява на два клона да се съберат в един, като разликите между клоновете се съгласуват чрез изкуствен интелект или друг механизъм. Когато системата не е способна да избере сама се намесва човек.

Осъществява се контрол чрез оторизиране. Създателят на проекта създава различни роли, притежаващи различни права, и ги възлага на другите участници.

При открит бъг или нова идея, която може да подобри кода, всеки, който има достъп, може да публикува бележка(issue), на която да опише проблема или предложението. Това помага значително на общността, поддържаща отворен код (open source community).

Някои системи предлагат възможност за добавяне на помощни файлове към кода. Те подобряват документацията и поощряват използването на чужд, добре описан код, като позволяват дори преизползване без подробни познания за структурата и съдържанието му (black-box reuse).

#### **Git, Github и TortoiseGit**

За целта на проекта е избрана системата Гит (Git). Тя предоставя всичко описано по-горе и дори надгражда списъка с възможности. Безплатна е и е силно застъпена както в професионалната, така и в любителската среда. В интернет е налична много подробна документация за нея [5].

С Гит системата може да се работи както през конзолен интерфейс(CLI - command-line interface), така и през графичен интерфейс (GUI - graphical user interface). Основни команди в Гит са

* git add – добавя промените в списък
* git commit – създава ревизия с добавените промени на локално ниво
* git push – изпраща списъка с ревизии на сървър, който синхронизира различните устройства
* git fetch – изтегля списъка с промени от сървър
* git pull – изтегля и интегрира списъка с промени от сървър

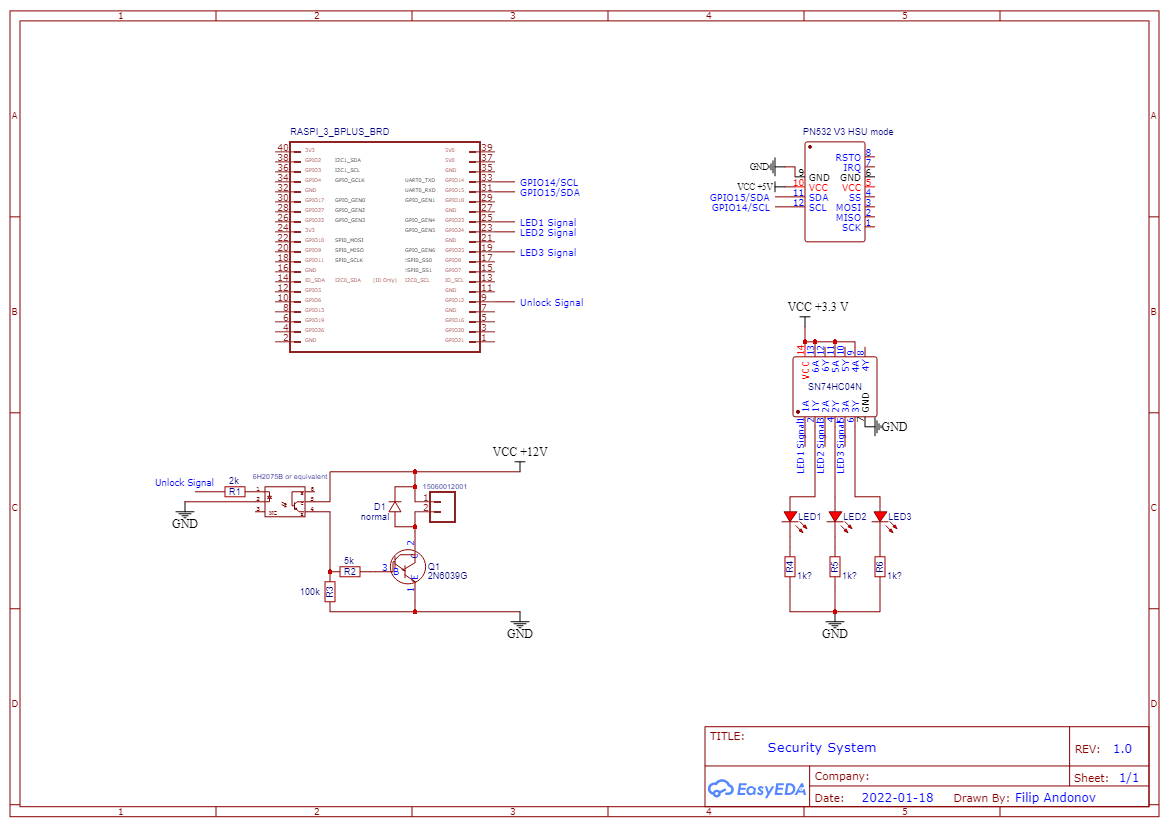
За работа със системата Гит са използвани два софтуера: TortoiseGit и Github.

Програмата TortoiseGit (4) представлява графичен интерфейс, опростяващ работата с Гит. Работата с него е лесна и интуитивна. Софтуерът е поддържан, документиран и безплатен.

Github (5) е уеб базирана платформа, поддържаща сървъра, използван от Гит системата. Платформата предоставя безплатно и сигурно хранилище за ревизиите, достъпно от цял свят през уеб браузър. Линк към Github хранилището можем да намерим в списъка „Други полезни линкове“ (6)

### **Хардуерна архитектура**

#### **Хардуерна схема**

Чертежът на електрониката (фиг.2) е съставен чрез софтуерния инструмент EasyEDA (7). Той е безплатен, служи за чертаене на електронни схеми и има богата библиотека с елементи, обогатявана от своите потребители.

Фигура 2. Схема на електронните компоненти, създадена чрез EasyEDA

До всеки, свързан в системата пин, стои надпис, сочещ съответния пин на външното устройство. Същото важи за всеки модул от схемата. Микроконтролерът, използван в проекта (а именно Raspberry Pi 3 B+) е озаглавен RASPI\_3\_BPLUS\_BRD. Причината за избора на този контролер е, че той е сравнително евтин и поддържа дистрибуция на операционната система Линукс. Съдържа вграден Ethernet и Wi-Fi приемник, което добавя гъвкавост спрямо нуждите на клиента. Четириядреният му процесор BCM2837 с архитектура ARMv7 и четирите гигабайта РАМ позволяват бърза паралелна обработка на данните. В платката са вградени 4 USB порта, които могат да бъдат свързани с USB към TTL конвертори и USB разклонители. С тези допълнителни входове можем да надградим броя на радиочестотните сензори. Платката съдържа 40 пина, 28 от които са GPIO(General Purpose Input/Output) с добавени функционалности, подходящи за интерфейс към електрониката. Два от тях предоставят захранване на 3.3 и други два на 5 волта. Пиновете предназначени за захранване имат завишени характеристики. Енергонезависимата памет на платката е micro-SD карта. Това я прави лесна за подмяна при износване на транзисторите ѝ. Платката поддържа видео и аудио интерфейси, което позволява индивидуални разработки за надграждане на проекта.

Според официалната документация, предоставена от производителите [6], при напрежение от 3.3 волта, максималният ток, който GPIO пиновете на микроконтролера могат да издържат, е 16мА, но това е стресово положение и се препоръчва да не надхвърля 8мА. Ключалките обаче често работят на принципа на индукцията и черпят много повече ток.

За проекта е избрана ключалка 15060012001 [7]. Съотношението цена-качество при нея е много добро. Дори бюджетна ключалка като тази има нужда от 430мА постоянен ток при 12 волта напрежение. За управлението ѝ е нужен механизъм с външно захранване. Този механизъм може да се види на схемата в долния ляв ъгъл. Състои се от български оптрон 6Н2075В, ключалката, диод, транзистор на Дарлингтън, външно захранване и ограничаващи резистори. Принципа на действие е следния: Външен сигнал, изпратен от микроконтролера, захранва слаботоков светодиод, който отпушва фототранзистор. По този начин се създава галванично развързване, което пази слаботоковата електроника от евентуални токови удари. През отпушения транзистор потича по-силен ток, който минава през резистор и транзистор на Дарлингтън, който от своя страна отпушва веригата на ключалката и това води до протичане на много по-голям ток през елемент 15060012001. Диод D1 служи за разсейване на пиковете породени от електромагнитната индукция в бобината на елемента, които биха повредили транзистор Q1. Всички стойности са пресметнати спрямо наръчниците на елементите в схемата и законите на физиката (Закон на Ом и др.). Токът през диода на оптрона е ограничен до 2мА чрез съпротивление R1 със стойност 2100 Ома. Съпротивленията R2(5000 Ома) и R3(100кОма) служат за да ограничат тока във веригата и да предотвратят изгаряне.

За близкополеви комуникации е използван сензорът PN532. Той е подходящ за разработване на проекти, защото съдържа богата документация [8] и много широка гама от възможности. Начините за комуникация със сензора са три: Inner Integrated Circuit (I2C), High Speed UART (HSU) и Serial Peripheral Interface (SPI). За проекта е избран HSU, защото микроконтролерът съдържа няколко USB порта. Всеки порт може да обслужва поне един сензор. Елемент PN532 може да бъде захранван с 3.3 и 5 волта. Максималната консумацията на ток е 150мА. Захранван е от общия ток между 5-волтовите изходи на платката, защото когато са обединени те могат да дадат повече от 300мА, или от USB към TTL конверторите през USB портовете. Неговата земя е свързана към тази на елемент RASPI\_3\_BPLUS\_BRD.

Порталният модул съдържа система за известяване на състоянието си. Тя е осъществена чрез два светодиода на сензор. Те са свързани между инвертираща интегрална схема SN74HC04N и резистор. Резисторите ограничават тока в диодите, а интегралната схема служи за захранване. Логически входове се управляват с напрежение и съответно с малко ток (2мА). Логическите изходи черпят своя ток от захранването на чипа. Това позволява по-голя товар (4мА), управляван от по-слаби сигнали (като тези на контролера). Така се избягват излишни компоненти, служещи за да осигурят по-голям ток на диодите. SN74HC04N черпи захранването си от 3.3-волтовите изходи на микроконтролера, което облекчава общият товар върху GPIO пиновете на достъпна цена (около 50 ст./бр.).

R4, R5 и R6 са по 300 Ома. Пресметнато по закон на Ом токът, протичащ през всеки диод, е 4мА.

#### **Захранване**

Захранването на порталната система се реализира през два адаптера: 220V-5V, предоставящ поне 2100мА, захранващ микроконтролера през micro USB вход и 220V-12V, предоставящ поне 1А на системата с ключалката през букса с размер 2.1x5.5x11. Възможно е захранване на целия модул от един захранващ адаптер, но за целта той ще трябва да бъде с по-голяма мощност и системата трябва да бъде надградена с логически преобразовател и стабилизатор на напрежение от 12 към 5 волта. Интеграцията на общо захранване би увеличила сложността на проекта, но би улеснила крайния потребител.

### **База данни**

#### **Релационна база данни**

Релационната база данни представлява колекция от данни, свързани с предефинирана корелация между таблици с информация. Съществуват различни системи като най-известните са MySQL, PostgreSQL, Microsoft SQL Server, SQLite, MongoDB, Oracle и Firebase.

#### **MySQL**

За целта на проекта е избрана MySQL, защото е безплатна, изключително оптимизирана, широко използвана, добре документирана [9], позволява многонишков (multithreading) достъп и предоставя разработена библиотека за свързване чрез C/C++ апликация. Инсталацията ѝ може да стане по няколко начина, но препоръчителният за Линукс е чрез APT(Advanced package tool). Това е програма, вградена в операционната система, чиято задача е да инсталира и премахва софтуер. Инсталацията се случва чрез въвеждане на следната команда в терминала „sudo apt-get install mysql-server“. Проверка за успешна инсталация можем да осъществим с въвеждане на командата „mysql -V“. Чрез въвеждане на „mysql“ влизаме в интерфейса на приложението. То се контролира от команди наречени запитвания(queries). След като сме се удостоверили успешно можем да създадем нова база данни на име „Access\_Control\_System“ чрез въвеждане на „CREATE DATABASE Access\_Control\_System“.

За проекта са нужни две таблици:

Първата е водеща и е наречена „check\_attempt“. В нея се записват всички опити за преминаване през порталите. Съдържа полетата:

* „attempt\_id“ – Идентификационен номер на всеки опит за достъп. Тази колона е от вид интегрално число и се използва за главен ключ.
* „card“ – Уникален номер на картата или устройството. Колоната е от вид интегрално число и се използва като външен ключ, сочещ към главния ключ на използваното устройство от втората таблица.
* „time“ – Съдържа точното време на опита във вид timestamp
* „device\_name“ – Име на портала (устройството) от вид символен низ
* „status\_code“ – Код, показващ как е отговорила системата от вид интегрално число

Втората таблица „card“ съдържа информация за регистрираните устройства. Състои се от колоните:

* „card\_uid“ – Уникален код, присъщ на всяка карта или устройство. Играе ролята на главен ключ от вид интегрално число и се инкрементира автоматично за всеки нов запис(AUTO INCREMENT).
* „associated\_name“ – Име на служителя, притежаващ картата, от вид символен низ.
* „associated\_phone\_number“ – Телефонен номер на служителя от вид символен низ.
* „granted\_access\_level“ – Ниво на достъп от вид интегрално число.
* „card\_code“ – Записан в картата код, служещ за проверка в базата данни от вид символен низ.

Таблица се създава чрез командата: „CREATE TABLE table\_name (

Field1 type modifier,

Field2 type modifier

);“, като „table\_name“ се замества с името на таблицата Field с името на полето, „type“ е неговия вид (например интегрално число/int или символен низ/varchar), а „modifier“ с различни модификатори като „NOT NULL“ и „PRIMARY KEY“.

След създаването им, таблиците трябва да бъдат свързани чрез външен ключ (foreign key). Това става с: „ALTER TABLE table1\_name ADD FOREIGN KEY (field1) REFERENCES table2\_name(field2);“, където „table1\_name“ и „table2\_name“ са имената на таблиците, които се свързват, а „field1“ и „field2“ са имената на колоните, чрез които двете таблици се синхронизират.

След правилно създаване, таблиците могат да бъдат изпробвани със следната заявка за добавяне на нова карта: „INSERT INTO card (associated\_name, associated\_phone\_number, granted\_access\_level, card\_code) VALUES ( 'Real chip', '00 359 000 000 000', 2, 'c4728803' );“, която записва асоциираното име „Real chip“, телефонен номер '00 359 000 000 000', ниво на достъп 2 и код на картата 'c4728803'. Ръчно може да се добави нов опит за вход: „INSERT INTO check\_attempt (card, time, device\_name, status\_code) VALUES ( 1, CURRENT\_TIMESTAMP, 'SOUTH GATE #2', 0);“. В системата ще бъде записан опит, извършен от карта с идентификационен номер 1. MySQL функцията CURRENT\_TIMESTAMP записва сегашното време, сверено спрямо сървъра. Името на устройството, на което е направен опита за вход, е 'SOUTH GATE #2', а кодът на състоянието е 0.

Ако информацията е въведена правилно чрез заявката „SELECT attempt\_id, check\_attempt.card, card.card\_code, time, device\_name, status\_code, card.associated\_name, card.associated\_phone\_number, card.granted\_access\_level FROM

check\_attempt LEFT JOIN card ON check\_attempt.card = card.card\_uid;“ системата ще покаже таблица с подробна информация за това кой потребител, кога, къде се е удостоверил и какъв е статуса на опита. От MySQL интерфейса се излиза с командата „exit“.

Структурата на базите данни е описана в /others/Database\_model.txt

### **Общо за софтуерната архитектура**

#### **Обектно ориентирано програмиране**

Софтуерът, който се изпълнява на отделните модули е различен, но практиките и стилът са еднакви. Използван е обектно ориентиран (ОО) подход, смесен с извикването на отделни функции. Обектно ориентираното програмиране (ООП) представлява парадигма, в която програмата е разглеждана като сбор от обекти. Обектът е инстанция на клас. Всеки клас съдържа атрибути (attributes) и поведение (behavior), изразяващо се във функции, присъщи на класа и наречени методи (methods).

Един клас може да съдържа друг клас и това се нарича композиция (composition). Чрез класове е възможна абстракция. Например ако разглеждаме човека, той е живо същество. Съответно можем да създадем клас генерализиращ атрибутите и поведението на живите същества и клас, който да наследи тази информация, като я надгради с по-подробни, характерни за човека, черти. Този метод се нарича наследяване (inheritance). По-абстрактния клас се нарича родител (parent), а по-конкретния – дете (child).

Енкапсулацията (encapsulation) позволява ограничаването на определени поведения и атрибути. Полезна е когато искаме да скрием детайли, обвързани с вътрешното поведение на клас. Изразява се в три нива на видимост. Частно (private), когато даденото поведение или атрибут е налично само в рамките на класа, предпазено (protected), когато е достъпно в рамките на класа и на всеки наследяващ го и публично(public), когато няма ограничения относно достъпа.

Концепцията на полиморфизма (polymorphism) ни позволява да разглеждаме един обект като част от група, която наследява. Ако клас куче е част от клас четириноги, а четириногите имат свойството да ходят, то тогава ние можем да извикаме това свойство, защото то е присъщо и на кучето.

Предимствата на обектно ориентираното програмиране са, че позволява абстракция, улеснява тестването и позволява паралелно разработване, но често на цената на повече операции, извършвани от хардуера.

Програмата е разбита на програмни единици (units/classes). Всяка единица се разработва отделно и се разглежда подробно в момента на писане, но след завършването и тестването ѝ започва да се гледа като черна кутия с входове и изходи. Разработването на един клас е представлявано от една бележка в Trello.

#### **UML диаграми**

Структурите на класовете са описани в UML диаграми, а резултатът от тяхното взаимодействие е описан в блок схеми, отговарящи на различните модули. За създаването на диаграмите и схемите е използвано уеб приложението draw (8). Платформата предоставя безплатно богат избор от инструменти за чертаене на най-различни диаграми, схеми и други.

UML (universal markup language) диаграмата е чертеж, предоставящ информация за съдържанието на класовете. Един клас представлява правоъгълник с име и полета за атрибути и методи. Пред името на всеки атрибут или метод се поставя знак, който представлява достъпността му. Знаците са „-“ за частен, „+“ за публичен и „#“ за защитен. След името се записват „:“ и видът на променливата/метода. Свързването на отделните класове и тяхната връзка се отбелязва с линии, като в зависимост от връзката линиите са комбинирани с различни елементи. При наследяване към по-подробния клас (child) е свързан празен ромб или „child-parent“ текст, а при композиция само запълнен ромб. Всяка UML диаграма в проекта съдържа име, автор и дата на последна промяна.

На фиг. 3 е изобразена диаграмата, описваща порталния модул. Тя може да бъде намерена в „/others/RaspberryPi\_UML\_diagram.xml“. Всички UML диаграми в този проект се намират в „/others/“ и завършват с „\_UML\_diagram.xml“. Тук се виждат имената на класовете (първият ред в правоъгълниците), атрибутите им(вторият ред) и методите им (третият ред). Може да се види и че „GPIO“ класът е с шрифт italic, което показва, че той е изцяло абстрактен клас и самостоятелен обект от него не може да съществува.

Diagram

Description automatically generatedФигура 3. UML диаграмата порталния модул

#### **Блок схеми**

Блок схемата е чертеж, предоставящ информация за последователността на действията, изпълнявани от програмата. Елементите, съставящи блок схемата са следните:

* Стрелка – Посочва логическата посока на изпълнение на отделните елементи
* Елипса – Отбелязва начало и край на програмата
* Ромб – Отбелязва логическа проверка, като от ъглите на ромба тръгват връзки, спрямо условието
* Трапец – Отбелязва вход към и изход от програмата
* Правоъгълник – Отбелязва логически елемент

Всяка блок схема в проекта съдържа име, автор и дата на последна промяна.

Diagram

Description automatically generatedНа фиг. 4 е изобразена блок схемата, описваща софтуера на порталния модул. Тя се намира в „/others/“, като всяка блок схема завършва с „\_flow\_diagram.xml“.

Фигура 4. Блок схемата на порталния модул

### **Софтуерна архитектура на портален модул**

Софтуерът за микроконтролера, принадлежащ на порталния модул, се състои от група взаимодействащи си класове. Написан е на C и C++, поради бързодействието и контролът, който езиците позволяват. Програмата е планирана за дълготрайно използване, затова е създаден режим на работа, в който тя не отпечатва информация с цел спестяване на място в паметта и режим, в който отпечатва случващото се, за да спомогне поддръжката в бъдеще. В основата на всеки друг клас стои класът „IOManager“. Композицията позволява извеждането на структурирана информация за състоянието си. Всеки клас е разделен в заглавен файл (header file) и ресурсен файл(source file). В заглавните файлове се намират „header guard“-ове, декларациите на структурите и класа, а в ресурсните файлове се намират дефинициите им.

#### **Верига от инструменти**

За писането на всички програми на C и C++ е използвана интегрираната среда (IDE) за разработка Visual Studio Code(9). Тя е удобна и много гъвкава. Позволява надграждане. При разработка са използвани добавките C/C++ компилатор, „Doxygen“, „Git“ и „Prettier“.

Добавката за компилатора позволява настройка на компилационната команда.

„Doxygen“ е софтуер, позволяващ документирането на кода. Той е изключително полезен, защото освен, че улеснява четенето на код, чрез него може автоматично да се генерира документация. Работи с коментарите в кода, като ги комбинира с анотации(@) и ключови думи.

„Prettier“ добавката е програмирана така, че при всяко запазване на файл, той да бъде форматиран в конвенционален стил.

#### **Конвенции за именуване**

В C/C++ кода е използвана „lower camel case“ конвенция за наименуване. Името на всяка променлива и метод започва с малка буква, като всяка следваща дума е залепена и започва с главна буква. Полетата (fields) на клас започват с префикс „m\_“.

### **Софтуерни класове за порталния модул**

Разпределението и съдържанието на класовете е описано в UML диаграмата, отговаряща за тази част от системата. Те са: „IOManager“, „PN532“, „SocketClient“, „GPIO“, „Led“ и „Lock“.

#### **Клас „IOManager“**

„IOManager“ е отговорен за записването на информация в текстов файл. Поради същността на проблема, класът е съобразен с конкурентно записване в един файл. Неговите полета са:

* „m\_stream“ – „fstream“ обект от стандартната библиотека. Позволява записване и четене на поток от данни във файл. Постига това чрез системно повикване, което от своя страна е начин за използване на „kernel space“ функционалност от „user space“ на операционната система Линукс. „Kernel space“ кодът работи на много ниски нива и има достъп до хардуера, като това му позволява извличане или записване на информацията. „User space“ кодът няма достъп до толкова ценни и опасни нива, затова той може само и единствено да провокира такъв процес чрез дефиниран лист от системни повиквания.
* „m\_filename“ – константен текст, помнещ пътеката на файла, който се обработва от обекта.
* „m\_lock“ – „pthread mutex“, заключващ писането в този файл временно. Причината за използването му е, че ако две нишки записват едновременно, информацията ще стане хаотична и безполезна. Избран е мютекс от библиотеката „posix threads“, защото тя е поддържана на много платформи като Уиндоус, Линукс, Андроид и други. „Posix“ нишките са по-стабилни от   
  „std::thread“ нишките, защото те стоят в основата на стандартната имплементация.

Класът има конструктор и деструктор. Конструкторът инициализира мютекса, а деструкторът го унищожава. „write“ и „read“ се грижат за отварянето и затварянето на поток от данни и редуването при работата с него.

#### **Клас „PN532“**

„PN532“ е клас опаковка (wrapper class) опростяващ работата с „nfc-tools/libnfc“ (https://github.com/nfc-tools/libnfc). „Libnfc“ е безплатна библиотека с отворен код, разработена от „nfc-tools“ на езика C, предоставяща пълната функционалност на елемента PN532 чрез опростен програмен интерфейс. Кодът в нея отново използва файловата система на Линукс, за да изпрати информация през UART, без значение дали хардуера е вграден или добавен. Тази информация се записва в определените регистри на сензора и задейства вътрешните му механизми. Библиотеката трябва да бъде инсталирана на машината и програмата да бъде свързана с нея (за GNU Compiler Collection това става с флаг „-lnfc“ при компилация).

Класът обгръща работата с библиотеката. Една инстанция обслужва един сензор. Конструкторът и деструкторът инициализират и почистват след библиотеката. Методът „poll“ включва сензора в режим на блокиращо изчакване. Програмата спира докато сензорът изпрати обратно информация за засечено устройство. „poll“ се нуждае от алокиран в паметта обект, върху който да запише извлечената информация и вид на модулацията, използвана от устройството.

#### **Клас „ClientSocket“**

За връзката през интернет мрежата е отреден класът „ClientSocket“. Той използва Линукс библиотеките „socket“, „types“, „netinet/in“, „netdb“ и „unistd“, за да осъществи клиент-сървърна връзка чрез ниво 4 от „OSI“ модела, а именно транспортният слой. „OSI“ е седемпластна структура от хардуер и софтуер, отговорна за преноса на данни през мрежа. По-горните пластове използват и надграждат осигуреното от по-долните. Транспортното ниво е отговорно за връзка между два отделни IP адреса, чрез TCP, UDP и други. Разглежданият клас използва Transmission Control Protocol и Internet Protocol(TCP/IP).

Конструкторът на класа получава параметри и инициализира основните атрибути. Деструкторът осигурява прекъсването на връзката. Методът „connectToServer“ е отговорен за свързването към сървър. В него се създава сокет от адресно семейство „AF\_INET“, през който ще се предават данни, след което изтегля IP адреса на сървъра от DNS\*. Адреса, порта и сокета са подадени като аргументи на функцията connect. Ако връзката е успешна, методът връща резултат 0, а ако не – 1. „disconnectFromServer“ прекъсва връзката. Методите „send“ и „receive“ използват функциите „write“ и „read“, за да взаимодействат с операционната система и библиотеките ѝ, а „isConnected“ дава информация за състоянието на връзката.

\* „Domain Name System“ (DNS) е система с данни, съдържаща адресите на всеки сървър, достъпен чрез „Uniform Resource Locator“ (URL).

#### **Клас „GPIO“**

GPIO изходите на микроконтролера се контролират чрез записване на стойности във фиксирани регистри. Отново операционната система е ниво на абстракция, което улеснява работата значително, но на цената на по-ниска производителност. Контролът от програмата е имплементиран чрез абстрактния клас „GPIO“. Той е абстрактен, защото всички изходни сигнали ще бъдат специализирани, но ще имат препокриващи се общи характеристики. Отново конструкторът инициализира, а деструкторът деинициализира изхода на платката. Това става чрез частните методи „export“ и „unexport“. Те подават команди на ОС да захрани или изключи съответния хардуер. Чрез метода „setDirection“ може да се избере посока на крака. Двете възможности са вход и изход. Методът „setValue“ е валиден само в режим на изход и през него се избира стойността на изходния сигнал. Възможностите са логическа 0 или 1. „getNumber“, „getDirection“ и „getValue“ са съответно методите, които извличат номера, посоката и стойността на даден GPIO пин. В основата на „setValue“ и „getValue“ стоят два други частни метода: „writeValue“ и „readValue“. Чрез тях се достъпва файловата система и се записват съответните стойности. При стартиране всеки използван пин е инициализиран като вход, защото когато е в режим вход, той има много високо входно съпротивление и това би спасило веригата от евентуално късо съединение при грешно свързване.

#### **Клас „LED“**

За управление на ЛЕД диодите е създаден „LED“ клас. Той наследява „GPIO“, като дава достъп до прост интерфейс, състоящ се от 2 метода: „settled“ и „getValue“. Първият е отговорен за захранване на диода, а вторият за извличане на информацията относно неговото състояние. Чрез конструктора се контролира възможността за инвертиране на диода, тоест, ако хардуерът е логически обратно свързан, софтуерът решава този проблем.

#### **Клас „Lock“**

За управление на ключалката е използван „Lоck“ класът. Той наследява „GPIO“, като опростява работата с ключалката. Съдържа два метода: „setLock“ и „getValue“. Първият контролира захранването на ключалката, а вторият е наследен от класа родител и дава информация за състоянието на хардуера. След инициализация ключалката е в затворено положение.

### **Софтуерна архитектура на сървърна апликация**

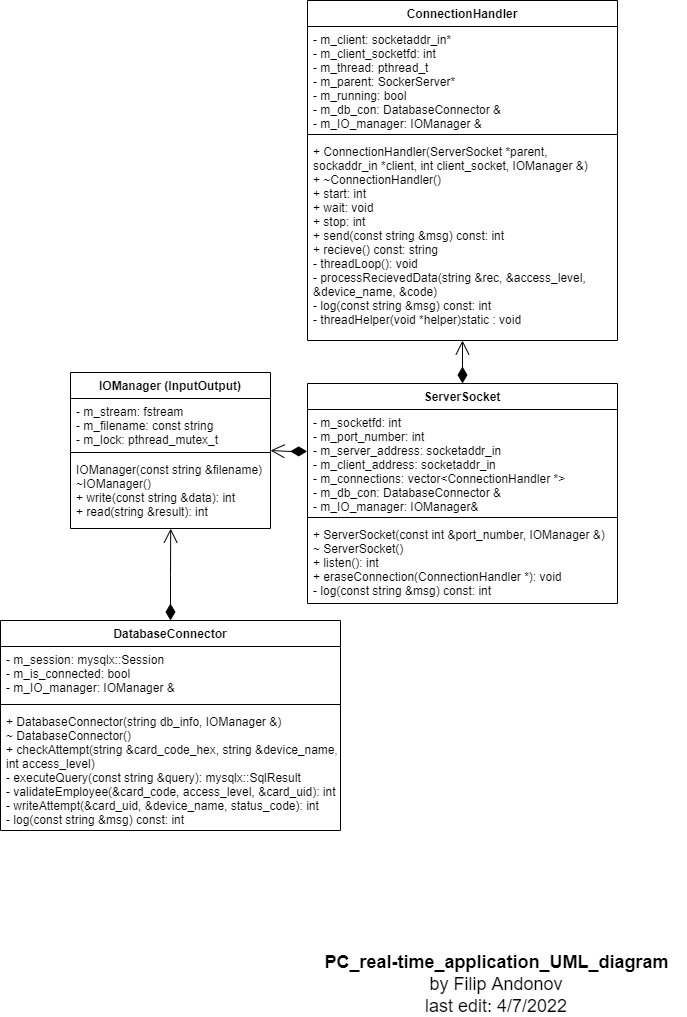
Сървърният модул се състои от група взаимодействащи помежду си класове. Написан е на C++, като е комбиниран с библиотеки предоставени от „MySQL“ и Линукс. Програмата е планирана за дълготрайно използване, затова е създаден режим на работа, в който тя не отпечатва информация с цел спестяване на памет и режим, в който отпечатва случващото се за да спомогне поддръжката в бъдеще. Оформянето на класовете е същото като в софтуера, предназначен за микроконтролера. Те са разделени на заглавен файл (header file) и ресурсен файл(source file), като всеки съдържа инстанция на класа „IOManager“.

#### **Конвенции за именуване**

В C и C++ кода е използвана „lower camel case“ конвенция за наименуване. Името на всяка променлива и метод започва с малка буква, като всяка следваща дума е залепена и започва с главна буква. Полетата (fields) на клас започват с префикс „m\_“.

### **Софтуерни класове за сървърната апликация**

Разпределението и съдържанието на класовете е описано в UML диаграмата на фиг. 5, отговаряща за тази част от системата. Те са „IOManager“, „DatabaseConnector“, „ServerSocket“ и „ConnectionHandler“.

Фигура 5. UML диаграма на сървърния модул

#### **Клас „ServerSocket“**

„ServerSocket“ класът играе ролята на разпределител, който обработва новите свързвания към сървъра. Всяка връзка се приема, запазва се като показател и се отделя в нова нишка. Конструкторът и деструкторът са отговорни за менажирането на паметта. За включване на сървъра се използва методът „listen“. Той стартира поток към сокет от адресно семейство „AF\_INET“. Семейството е избрано така, че да е съвместимо с клиентските заявки, изпратени от порталния модул. След успешно създаване, сокетът се инициализира и стартира. Програмата влиза в безкраен цикъл, чакащ постъпването на нова връзка. Щом получи заявка през мрежата, тя бива приета и се свързва с новосъздаден обект от вид „ConnectionHandler“, запазва се показател към него във вектор и се пуска изпълнението му в самостоятелна нишка, след което отново се влиза в режим на изчакване за нова връзка. За прекъсване е отговорен методът „eraseConnection“, който чрез итератор, намира показателя и го премахва, като извиква деструктора му.

#### **Клас „ConnectionHandler“**

Всяка отделна връзка се обработва от модула „ConnectionHandler“. Конструкторът инициализира полетата. Деструкторът почиства паметта. Методът „start“ създава нова нишка изпълняваща „threadLoop“ посредством „threadHelper“, „stop“ задейства безопасно спиране, а „wait“ терминира нишката след успешно завършване. „send“ и „receive“ са методи, използвани за изпращане и получаване на данни през мрежата. „threadLoop“ методът започва изпълнението си с получаване на данни за вход от портален модул, които се обработват чрез спомагателния метод „processRecievedData“, извличайки ниво на достъп, име на устройството и код на картата. Обработената информация се проверява в базата данни чрез обект от вид „DatabaseConnector“ и спрямо резултата се изпраща обратно съответното съобщение. Вариантите отговор през мрежата са два: позволен и забранен достъп, като лесно могат да бъдат добавени и допълнителни възможности.

#### **Връзка между „MySQL“ и сървърен модул**

За осъществяване на връзка между сървърния модул и „MySQL“ се използва конектора „MySQL Connector/C++ 8.0“. Той поддържа три програмни интерфейса: „XDevAPI“, „XDevAPI for C“ и „JDBC API“. За C++ апликации е препоръчително да се използва „XDevAPI“, защото е оптимизиран за езика и е актуален. Възможно е да се изтегли под формата на бинарен файл и на програмен код. Наръчник за инсталация и използване на интерфейса е предоставен от производителя [10].

#### **Клас „DatabaseConnector“**

„DatabaseConnector“ е клас обвивка, капсулиращ библиотеката „MySQL XDevApi“ с цел улесняване на работата. Чрез полето „m\_session“ се отваря нова сесия към базата данни. Това става в конструктора. Деструкторът затваря сесията. Заявките към сървъра се изпълняват чрез „executeQuery“. „checkAttempt“ служи като публичен метод – буфер, синхронизиращ въвеждането на опит за вход в системата чрез последователно използване на частни методи. Той е извикван от „ConnectionHandler“ в „threadLoop“ и изцяло отделя интерфейса на библиотеката от останалия код. „validateEmployee“ изпраща заявка за определен потребител, проверява дали той има достъп да премине през портала и връща резултат. „writeAttempt“ записва опита в базата данни.

### **Потребителски интерфейс / потребителско преживяване**

За създаването на потребителски интерфейс / потребителско преживяване (UI/UX) модел на уеб модула е използван AdobeXD(10)(фиг. 6). Това е безплатен софтуер, позволяващ изграждането на интерактивни модели. Приложението е подходящо, защото предоставя инструменти за създаване и групиране на графични елементи и анимации.

Graphical user interface, application

Description automatically generatedФигура 6. Екранна снимка от Access\_Control\_System\_UI\_Model.xd, създаден чрез AdobeXD

Всеки прозорец е съставен от различни елементи: правоъгълници, черти, текст, икони и др. На снимката може да се забележи главния прозорец „Web 1920 – Main“. Той ще съдържа таблица с всички служители, въведени в системата. Всеки ред в таблицата предоставя кратка информация за служител. Чрез бутона „remove“ желаният служител бива изтрити. Бутонът add води до друга страница – „Web 1920 – Add“. Тази страница съдържа форма, чрез която да се добави ново досие на служител. От навигационния списък (синьото поле с къщичката в него) можем да навигираме отново към главната страница. От там при клик на бутона „details“, програмата ще ни отведе към нов прозорец. Той се казва „Web 1920 – Details“ и съдържа подробна информация за маркирания служител. От там можем да разберем имена, телефонен номер, ниво на достъп, код на устройството, което използва, работното му време в определен период и да изведем кога и на кой портал е минал.

### **Софтуерна архитектура – уеб приложение**

Потребителски приветлив достъп до информацията, натрупана от системата, осигурява третият модул – уеб приложение.

За осъществяване на уеб модула е използвана рамката (framework) Джанго (Django). Тя е безплатна и базирана на езика питон, като следва архитектурния модел модел-изгледи-шаблон (model-views-template). Използвана е за да спести усилия и време при създаването на функционален уебсайт, служещ за показване и редактиране на информацията в базата данни. Алтернативни рамки са Express, Rails, Spring, Vue и Flask. Предимството на Django пред останалите е, че е написан на програмния език питон, което го прави лесен и интуитивен за използване, поддържа връзка с бази данни и предоставя всичката, нужна за целта на проекта, функционалност. Добрата документация [11] и многото ресурси по темата значително подобряват кривата на обучението и времето за решаване на проблеми.

#### **Джанго архитектурата**

Архитектурният модел на Джанго (model-views-template) работи на следния принцип:

Моделът (model) представлява сбор от класове, описващи данните, които се използват и съхраняват от бекенд частта на приложението. Тези данни и заявките от потребител биват подготвени от функции или класове, наречени изгледи (views). Визуализацията на крайния резултат се случва чрез шаблоните (templates). Шаблонът представлява съвкупност от JavaScript, HTML, CSS и Django template language. Чрез HTML се описва съдържанието на страницата. CSS служи за описване на визията и разположението на отделните елементи. Чрез JavaScript могат да бъдат добавени анимации и код, който да се изпълнява от потребителското устройство. Django template language е език за програмиране, служещ за свързване на изгледа чрез надграждане на HTML кода. Основните оператори в него са:

* {{ име }} – Променлива или обект, чиято стойност бива замествана в HTML кода, спрямо състоянието на програмата. Стойността се подава чрез речник с името „context“ от изгледа.
* {% block име %} HTML код {% endblock име %} – Блок от код, който може да бъде поставян многократно. Това позволява преизползване на кода, като промените направени в неговата дефиниция ще бъдат приложени върху всяка инстанция.
* {% if условие %} HTML код 1{% else %} HTML код 2{% endif %} – Условен оператор „Ако – то“. Ако резултат от условието е логическа единица, тогава ще се изпълни HTML код 1, а ако не – HTML код 2
* {% for променлива in колекция от данни %} HTML код {% endfor %} – HTML кода ще се изпълни за всеки елемент от колекцията, като стойността е достъпна чрез името на променливата, заградена с два чифта къдрави скоби.
* {% csrf\_token %}– Автоматично интегриране на CSRF токен, предотвратяващо „Cross-site request forgery“ атаки от злонамерени лице.
* {% load static %} – Зарежда съдържание на папката „static“ (например CSS файловете).
* {% extends “аплкиация/файл.html” %} – Информира Джанго, че тази страница използва код, взет от споменатия файл, като основа.

#### **Инсталация**

Инсталацията на рамката става чрез системния софтуер Пип (pip). Той е написан на питон и служи за инсталация на софтуери и поддържането на всички техни зависимости, като се свързва с интернет хранилища, от които изтегля нужните данни.

Инсталация на Джанго става чрез въвеждане на следната команда в терминала:

pip install django

За проверка на инсталацията можем да въведем

python -m Django --version

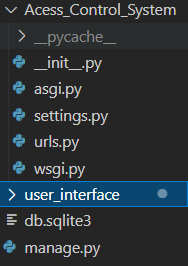
Резултат от втората команда трябва да бъде номера на инсталираната версия.

#### **Създаване и управление на Джанго проект**

Джанго проектът се състои от общи настройки и отделни приложения. Целта на приложенията е да се постигне абстракция, позволяваща преизползване на код. За създаване на проект се въвежда следната команда в терминала, отворен в желаната директория:

django-admin startproject name

Като „name“ се замества с името на проекта (Access\_Control\_System) написано на латиница. В него не се допускат наклонени черти и тирета. След успешно създаване, можем да видим нова директорията и файл с името „manage.py“ (фиг. 7).

Фигура 7. Файлова структура на Джанго проект

„manage.py“ – Служи за контролиране на рамката, включване и изключване на уеб сървъра, миграции, създаване на профили и др.

В папката има няколко файла:

* „\_\_init\_\_.py“ – Служи за идентификация, че този проект е написан за питон. Файлът стои празен.
* „setting.py“ – Съдържа настройки за проекта и уеб сървъра. Тук се намират конфигурациите за базите данни, шаблоните, приложенията, дебъга, валидацията на паролите, езика, времевата зона и др.
* „urls.py“ – Съдържа настройки за конфигурацията на различните адресни пътища (URL configuration) в домейна на всички приложения, изградени в проекта.
* „wsgi.py“ – Конфигурация на връзката между уеб апликацията и сървъра.
* „asgi.py“ – Конфигурация за асинхронна връзка между апликацията и сървъра.

Включването на сървъра става чрез командата:

python manage.py runserver

Спирането се случва чрез клавишната комбинация за прекъсване „CTRL + C“.

С въвеждането на командата:

python manage.py startapp name

създаваме нова апликация, като „name“ заместваме с името на апликацията (user\_interface). Резултат от изпълнението е нова папка със същото име. Съдържа файлове със същите имена и цел като главния проект. Допълнително са създадени директориите „migrations“, „static/user\_interface“ и „templates/user\_interface“.

Миграцията е процес, в който Джанго обновява модела на базата данни полуавтоматично. Това интегриране води до възможност за бъдещо надграждане. В директорията „migrations“ се съдържа информация за миграциите.

В „static/user\_interface“ се намират всички CSS файлове, използвани от шаблоните, а в „template/user\_interface“ шаблоните, написани на Django template language и HTML.

#### **Адресни пътища**

Маршрутизацията във вътрешната структура на уеб приложението се задава чрез файловете „urls.py“, като съществува един общ в директорията на проекта и по един специфичен за всяка създадена апликация. Отделните адреси са от вид „URLResolver“ и се декларират в лист с име „urlpatterns“. Обектите се инициализират посредством функцията „path“, която е претоварена (overload) с няколко аргумент листа. Първият аргумент във всички листове приема низ от символи, показващ адреса на който отговаря съответния обект, а втория приема или изглед, който да покаже на съответния адрес, или тюпъл от обекти, показващ по-следваща поредица от адреси(най-често подаван от апликация).

В проекта са създадени два „urls.py файла“ – един в главната директория и един в „user\_interface“ апликацията. Кодът в главната папка имплементира един път към линка „/admin/“, който показва администраторския панел, и един път към „/“, служещ за навързване на адресите, подадени от „user\_interface“.

Адресите в „user\_interface/urls.py“ са:

* „“ – Главната страница, показваща изгледа на всички служители
* „employee/new/“ – Извежда изгледа, отговорен за добавянето на нов служител
* „employee/<int:pk>/“ – Показва изглед към по-детайлна информация за служител, като го селектира чрез интегрално число(int), служещо за идентификационен номер, подадено в адресното пространство на мястото на <int:pk>
* „employee/<int:pk>/update/“– Изглед към форма, позволяваща модифицирането на вече съществуващ конкретен служител
* „employee/<int:pk>/remove/“ – Извежда изглед за изтриване на избран служител от базата данни

#### **Изгледи**

Джанго изгледите представляват код, който получава уеб запитвания и връща уеб отговор. Резултатът може да бъде под формата на уеб код, препращане към друга страница, съдържанието на такава или друго. Могат да съдържат бизнес логика за обработване на данни. Извикването им е отговорност на маршрутизатора, обяснен в предната точка. Съдържат се във файлове с името „views.py“.

Разделят се на два вида: базирани на клас и базирани на функция.

Базираните на функция изгледи, представляват обикновена функция, която връща уеб резултат. Предимствата им са, че са лесни за имплементация, прочит и логиката на кода е лесно проследима. Недостатъците им са, че трудно се преизползват и извикват външни функции за уеб отговорите.

Изгледите, базирани на класове, представляват класове, наследяващи от структурата на Джанго. Тяхната цел е да предоставят алтернативен начин за имплементация, подходящ за по-добра интеграция, преизползване и „чистота“ на кода. Те се намират в „django.views“. Основният базов клас се нарича „View“. Той е наследен и надграден от няколко вградени класа. При разработка на нов изглед се наследява някой от предоставените класове, след което всичко нужно се пренаписва(override) и надгражда, за да отговаря на изискванията. Недостатъците на този подход е, че може да бъде объркващ при четене и голяма част от имплементацията е скрита от разработчика.

Всеки един от изгледите свързва вградения модел с класа „Card“, изтегля данните от базата данни, посочва шаблона, който ще се използва и го зарежда в браузъра на потребителя. Списък с изгледите, използвани в проекта:

* „EmployeeListView“ – Наследява „ListView“ и сортира служителите възходящо по име.
* „EmployeeDetailView“ – Наследява „DetailView“ и „FormMixin“, извежда подробна информация за конкретен служител, включително и опитите за преминаване през системата, филтрирани по време.
* „EmployeeCreateView“ – Наследява „CreateView“, добавя полета за въвеждане на данни за нов служител, пренаписва метода за проверката на полетата и извежда еднократно съобщение за резултата от заявката.
* „EmployeeUpdateView“ – Наследява „UpdateView“, добавя полета за въвеждане на данни за вече съществуващ служител, пренаписва метода за проверка на полетата, като добавя еднократно съобщение за резултата на заявката.
* „EmployeeDeleteView“ – Наследява „DeleteView“, добавя форма за потвърждение и пренасочва към листа с всички служители при успешна заявка.

#### **Модели**

Моделът е единственият окончателен източник и носител на информация в приложението. Репрезентиран е от клас с полетата и поведенията на складираните данни. Класовете се описват във файловете „models.py“ и наследяват от основния клас „models.Model“. Класът родител предоставя готови методи, включително изтегляне на данните от интегрираната база данни.

В приложението са използвани два модела със съответните полета:

* „Card“ – Репрезентира данните на всеки служител. Съдържа полетата (аналог на базата данни):
  + „card\_uid“ – Идентификационен код на картата в базите данни, използван за основен ключ
  + „associated\_name“ – Име на служител с максимален размер от 40 символа
  + „associated\_phone\_number“ – Телефонен номер на служител с максимален размер от 20 символа
  + „granted\_access\_level“ – Ниво на достъп на служител
  + „card\_code“ – Уникален код на картата с максимален размер от 40 символа
* „Check\_attempt“ – Репрезентира данните за преминаванията през системата. Съдържа полетата (аналог на базата данни):
  + „attempt\_id“ – Идентификационен код на опита в базата данни, използван за основен ключ
  + „card“ – Вторичен ключ към устройството, направило опита
  + „device\_name“ – Име на портала, на който е направен опита, с максимален размер от 40 символа.
  + „status\_code“ – Код, репрезентиращ резултата от опита за преминаване през системата
  + „time“ – Точно време, когато е регистриран опита

## **Заключение**

Описаната по-горе теоретична част от дипломната работа на тема: „Система за контрол на достъпа на служители“ е текст с богато научно съдържание, успешно отразяващ анализа, проектирането, организирането и разработването на сложна многопластова системна архитектура, включваща разнообразни теми от сферата на информационните технологии, с цел решаване на социален казус от реалния живот.

Теоретичната част е разделена на глави, а те на теми, с цел структурирано представяне на информацията. Главите се фокусират върху различни аспекти на системата. Всяка тема разглежда отделните проблеми и взетите решения, като описва причината за избора. Използвани са технически термини, графики и примери с цел по-лесно усвояване на материала.

Освен практическата си приложимост, проектът е стойностен за читателя, защото описва основни концепции в областта на електрониката, системното програмиране, мрежовата комуникация и работата с операционна система, също така го запознава с модерни технологии за изграждане на графичен интерфейс чрез уеб апликация.

Представената информация в теоретичната част е достатъчна за разработването на системата. Всички основни похвати са пояснени подробно и надградени чрез списъка с използвана литература.

Порталният модул може да се предпази като бъде вграден в защитна кутия. Това ще наложи конфигурация в операционната система, която ще задейства и заключва софтуера при включване в мрежата за енергоснабдяване. Възможно е предвидено надграждане в посока елиминиране на втория щепсел с цел опростяване на крайния потребител. За организирането на разработката на допълнителна функционалност можем да използваме съответната колона с бележки от вид „Feature“ в „Trello“

# **Практическа част**

# **Списък на използвана литература**

Списък на използваната литература, отбелязвана с квадратни скоби:

1. Брус, Е. Thinking in C++ Vol 1, 2010.
2. Молой, Д. Exploring Raspberry Pi: interfacing to the real world with embedded Linux. Джон Уилей & Сонс, 2016.
3. Строустръп, Б. The C++ programming language. Pearson Education India, 2000.
4. Уейсфийлд, М. The object-oriented thought process. Pearson Education, 2008.
5. Git документация – <https://git-scm.com/doc>
6. Raspberry Pi документация – [https://www.raspberrypi.com/documentation/computers/ raspberry-pi.html#voltage-specifications](https://www.raspberrypi.com/documentation/computers/%20raspberry-pi.html#voltage-specifications)
7. Solenoid Lock документация – <https://www.amazon.com/Atoplee-Electric-Assembly-Solenoid-27X29X18mm/dp/B0125VGLT0>
8. PN532 документация – <https://www.nxp.com/docs/en/nxp/data-sheets/PN532_C1.pdf>
9. MySQL документация – <https://dev.mysql.com/doc/>
10. MySQL CPP Connector документация – [https://dev.mysql.com/doc/connector-cpp/8.0/en/ connector-cpp-installation-source.html](https://dev.mysql.com/doc/connector-cpp/8.0/en/%20connector-cpp-installation-source.html)
11. Django документация – <https://docs.djangoproject.com/en/4.0/>

## **Други полезни линкове**

Други полезни линкове, отбелязвани с кръгли скоби:

1. Проучване на Националния статистически институт – [https://nsi.bg/sites/default/files/ files/pressreleases/Population2020\_IVGTQG5.pdf](https://nsi.bg/sites/default/files/%20files/pressreleases/Population2020_IVGTQG5.pdf)
2. Linux – <https://www.linux.org/>
3. Trello дъска – <https://trello.com/b/c4JleDja/dr>
4. TortoiseGit – <https://tortoisegit.org/>
5. Github – <https://github.com/>
6. Github хранилище – <https://github.com/callmeFilip/Access_Control_System.git>
7. EasyEDA – <https://easyeda.com/>
8. Draw – <https://draw.io/>
9. Visual Studio Code – <https://code.visualstudio.com/>
10. Adobe XD – <https://www.adobe.com/products/xd.html>