**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

Факултет по телекомуникации

Катедра „Технологии и мениджмънт на комуникационни системи”

Специалност: Телекомуникации

**ДИПЛОМНА РАБОТА**

*Тема: Проектиране на учебна електронна платформа, базирана на GNU/Linux*

Студент: Васил Ивайлов Букоев

Фак. No: 111220084

Научен ръководител: гл. ас. д-р Милен Тодоров

София, 2024

**Съдържание**

[Увод 3](#_Toc167906771)

[Глава 1 - Литературен обзор 5](#_Toc167906772)

[1.1. Информационни системи 5](#_Toc167906773)

[1.2. Система за подпомагане на решения (DSS) 8](#_Toc167906774)

[1.3. Системи за управление на процесите 8](#_Toc167906777)

[1.4. Системи за мениджмънт на училище/университет 9](#_Toc167906778)

[1.5. Въведение в програмните езици 11](#_Toc167906782)

[1.6. Миграции с Visual Studio и записът на данни в MSSQL Studio 13](#_Toc167906783)

[Глава 2 – Теоретична част 18](#_Toc167906784)

[2.1. IDE: Интегрирана среда за разработка – Visual Studio 18](#_Toc167906785)

[2.2. SQL Server Management Studio – SSMS 21](#_Toc167906786)

[2.3. GNU/Linux – Ubuntu / Apache Server 26](#_Toc167906787)

[Глава 3 – Инженерно решение на поставената задача 31](#_Toc167906792)

[Администратор 36](#_Toc167906793)

[*преглед добавяне на клас и добавяне на годишна такса* 38](#_Toc167906794)

[*преглед и добавяне на учебен предмет* 40](#_Toc167906795)

[*преглед и добавяне на учител* 42](#_Toc167906796)

[*добавяне на клас и учебен предмет към определен учител* 43](#_Toc167906797)

[*добавяне на такса към определена лекция на учител* 44](#_Toc167906798)

[*следене на таксата към определена лекция на учител(детайли)* 46](#_Toc167906799)

[*добавяне на ученик* 47](#_Toc167906801)

[*преглед на присъствие на ученик* 48](#_Toc167906802)

[*добавяне на оценка* 49](#_Toc167906803)

[*детайли за оценка* 51](#_Toc167906804)

[*добавяне на присъствие на учител към определена дата и време* 51](#_Toc167906805)

[*преглед на детайли за присъствие на учител* 53](#_Toc167906806)

[Учител 54](#_Toc167906807)

[*преглед и добавяне на присъствие на ученик* 55](#_Toc167906808)

[*проверка на присъствие на студент* 57](#_Toc167906809)

[*проверка(детайли) на оценката на студент* 58](#_Toc167906810)

[Глава 4 – Анализ на получените резултати, приложимост и изводи 59](#_Toc167906811)

[Използвана литература 61](#_Toc167906812)

# **Увод**

С разрастването на цифровата ера и нарастващата зависимост от информационните технологии, използването на отворени и стабилни операционни системи става ключов фактор за развитието на иновативни софтуерни платформи. В този контекст, GNU/Linux изпъква като мощен фундамент за създаване на електронни системи, които не само предоставят функционалност, но и поддържат принципите на свободния софтуер. Настоящата дипломна работа се насочва към проектиране на електронна платформа, базирана на GNU/Linux, с цел подобряване на функционалността й и интеграцията във високотехнологични информационни системи.

С бързия темп на технологични промени и нарастващата сложност на изискванията в бизнес средата, компютърните системи трябва да бъдат не само издръжливи, но и гъвкави и лесни за разширение. GNU/Linux, като свободен и отворен операционен софтуер, предоставя отлична основа за разработване на такива системи, които могат да се адаптират към променящите се нужди и изисквания на потребителите. В този контекст, настоящата дипломна работа има за цел да изследва и разработи електронна платформа, използваща GNU/Linux, като ключов компонент за подпомагане на иновациите и устойчивото развитие в областта на информационните технологии.

Проектирането на електронна платформа със специфична ориентация към GNU/Linux не само ще допринесе за разширяване на практическите възможности на софтуерните решения, но и ще насърчи прилагането на принципите на свободния софтуер в корпоративната среда. Това може да създаде почва за по-широко приемане на отворените технологии и да улесни разработката на интелигентни и лесни за поддръжка информационни системи. В следващите глави ще бъдат разгледани изискванията към платформата, проектирането на архитектурата, изборът на технологии, разработването и тестването, както и възможностите за интеграция със съвременни информационни системи. Като цяло, работата ще насочи вниманието си към създаването на устойчива и иновативна електронна платформа, способна да удовлетвори разнообразните нужди на бизнеса в цифрова епоха.

В процеса на разработка и проектиране на електронната платформа ще бъде използвана съвременна методология, включваща анализ на изискванията, системно проектиране и интегриране на най-новите технологии. Съчетаването на добре установени практики с иновационен подход ще допринесе за създаването на решение, което не само ще отговаря на текущите предизвикателства, но и ще бъде готово за бъдещите изисквания на бизнеса и обществото.

Предназначението на разработваната електронна платформа е да бъде приложена в широк спектър от сфери, включително корпоративни, образователни и обществени институции. Със своята универсалност и гъвкавост, тази платформа цели да обслужва разнообразни потребителски групи и да допринесе за оптимизацията на бизнес процесите и управлението на информацията.

Целта на дипломната работа „Проектиране на електронна платформа, базирана на GNU/Linux“ е да проучи и анализира проблематичните аспекти, свързани със създаването и управлението на успешна онлайн платформа за управление на училищна система. Главната цел е да се представи обща представа за основните предизвикателства и важни аспекти, които трябва да се разгледат и се прилагат при разработването на онлайн платформа за „мениджмънт“ на училищна система.

Задачи на дипломната работа:

1. Да се проучат възможностите на операционна система, базирана на GNU/Linux за реализация на уеб сървър.
2. Да се предложи подход за реализиране на уеб сървър, осигуряващ хостинг на разработваната платформа чрез използване на операционна система, базирана на GNU/Linux.
3. Да се реализира сървър по предложения подход, като се осигури платформена независимост на потребителя.
4. Да се осигури контрол на достъпа до разработваната електронна платформа и защита на информацията.

Задачите на дипломната работа са насочени към изследване, анализ, проектиране и предложение на практически решения за успешното разработване и управление на онлайн система за управление, с фокус върху проблематичните аспекти, които възникват в процеса.

# **Глава 1 - Литературен обзор**

Информационните системи играят ключова роля в съвременния свят, като предоставят основен механизъм за обработка, съхранение и предаване на информация. Те се използват във всички сфери на обществото и бизнеса, подпомагайки ефективността, управлението и вземането на решения. В този контекст, разбирането на същността и функциите на информационните системи става от критично значение. Информационните системи се състоят от хардуерни и софтуерни компоненти, които взаимодействат, за да обработват и поддържат информация. Хардуерът включва компютърни устройства, мрежови съединения и периферни устройства, докато софтуерът обхваща програми и операционни системи. Съществен елемент в този контекст е и човешкият ресурс, който участва в управлението, проектирането и използването на информационните системи.

Въпреки че информационните системи са изключително мощни инструменти, те са свързани и с предизвикателства. Сигурността на данните, управлението на информацията и интеграцията между различни системи са често срещани теми, които изискват специфични решения. Възможностите за анализ и извличане на стойност от данните, съхранявани в информационните системи, представляват също важен фактор за успех във всяка област.

## **1.1. Информационни системи**

Информационната система е комбинацията от информационни технологии и действията на хората, които ги прилагат за управлението на процеси, вземане на решения и други с помощта на компютърни системи. Системата е предназначена за използване от организация или физическо лице и дава възможност за съхранение на бази данни, управление и обработка на цялата информация или на част от нея. Съществуват различни информационни системи: финансови, промишлени, географски, и други подобни. Като цяло информационните системи могат да бъдат разделени на такива, предназначени да извършват определена операция (обработка на транзакции, Transaction Processing) и на такива, които са предназначени за събиране на данни, необходими при вземането на решения (Decision Support). Според това на какво ниво в организацията се ползват, информационните системи могат да бъдат разделени на четири нива, които от долу нагоре са:

* Системи за обработка на транзакциите – ползвани основно от оперативните работници. Това са всички системи, които се използват в ежедневната работа на дадена компания, било то географска информационна система, системи за CAD, графично оформление и онлайн публикуване, подпомагане на телефонни услуги или други.
* Управленски (или мениджърски) информационни системи – ползвани от оперативния мениджмънт. Примери са системите за управление на взаимоотношенията с клиентите (CRM), за планиране на ресурсите на предприятието (ERP) системите и за управление на съдържанието.
* Системи за подпомагане на решения – системи за подпомагане на колективното управление и взимане на решения.
* Директорски информационни системи – предоставят редовни отчети и информация за прогреса по работата в организацията.

Според „Оксфорд“: „Данните са отделни части информация, обикновено форматирани по специфичен начин“. Данните [1] се измерват, събират, докладват и анализират, след което често се визуализират с помощта на графики, изображения или други инструменти за анализ. Суровите данни ("непреработени данни") могат да бъдат колекция от числа или знаци преди да бъдат "почистени" и коригирани от изследователи. Те трябва да бъдат коригирани, така че да можем да премахнем инструментални грешки или грешки при въвеждане на данни. Обработката на данни обикновено се извършва по етапи, и поради това "обработените данни" от един етап могат също да бъдат разглеждани като "сурови данни" за следващия етап. Полевите данни са данни, събрани в неконтролирана среда. Експерименталните данни са данни, които се генерират при наблюдение на научни изследвания.

Данните могат да бъдат генерирани от – хора, машини или при комбинирането между двете (от хора и машини заедно). Те често могат да бъдат генерирани навсякъде, където се създава и съхранява информация във структурирани или неструктурирани формати.

Данните са важни, защото помагат: за вземане на по-добри решения, при решаването на проблеми, като се намира причината за непроизводителност, да се оцени производителността, за подобряване на процесите, за разбиране на потребителите и пазара.

Обикновено данните могат да се класифицират в две части – категорийни и числови.

Категорийните данни са такива данни, които имат дефинирана категория, например семейно положение, политическа партия, цвят на очите.

Числовите данни могат допълнително да се класифицират в две категории: дискретни данни и непрекъснати данни. Дискретните данни съдържат данни с дискретни числени стойности, например брой на децата, дефекти за час и др. Непрекъснатите данни съдържат данни с непрекъснати числени стойности, например тегло, напрежение и др.

На напреднало ниво могат да се класифицират данните в четири части:

* Номинална скала- Номиналната скала класифицира данните в няколко отделни категории, при които не се предполага рангов критерий. Например пол, семейно положение.
* Ординална скала - класифицира данните в отделни категории, където се предполага рангов критерий. Например: степен на учебно-научна длъжност - професор, доцент, асистент; оценки на студенти - А, B, C, D, E, F.
* Интервална скала - представлява подредена скала, при която разликата между измерванията е смислено количество, но измерванията нямат истинска нулева точка. Например температура по Фаренхайт и Целзий, години.
* Относителна скала - представлява подредена скала, при която разликата между измерванията е смислено количество, и измерванията имат истинска нулева точка. Следователно можем да извършваме аритметични операции върху данните. Например тегло, възраст, заплата и др.

Съществуват няколко форми за представяне на данни.

*Символна форма*

Символ – служи за условно обозначаване на някакво понятие, явление или процес. Символите могат да бъдат алфавит, знаци, управляващи, графични, специални и т.н. Символите се кодират с кодови таблици, като за всеки символ се определя точно определена двоична комбинация, в зависимост от избрания метод за кодиране:

* ASCII – американски стандартен код за информационен обмен;
* EBCDIC – разширен двоично-десетичен код;
* Unicode – универсален код.

*Текст*

Форма за представяне на данните във вид на последователност от символи, на избран език, съдържателно разглеждана като едно цяло. В текста основно се използва естествен език и неговата азбука. При писане на програми се използват езици за програмиране. Видове текстово представяне:

* Текст – обикновен текст, чрез който се съставят документи.
* Хипертекст – свързан чрез ключови думи (хипервръзки) текст, чрез който се създават верижно свързани документи. На тази база се изгражда Web-пространството в Интернет.
* Метатекст – текст, който в себе си съдържа данни за собствената структура и съдържание. Той е основа за бъдещето развитие на Интернет като семантична мрежа.

## **1.2. Система за подпомагане на решения (DSS)**

Система за подпомагане на решения (DSS) [2] е компютърна програма, използвана за подкрепа за вземане на решени и действия в организация или бизнес. Система за подпомагане на решения преценява и анализира голямо количество данни, събирайки обширна информация, която може да се използва за решаване на проблеми и в процеса на вземане на решения.

Типичната информация, използвана от системата за подпомагане на решения, включва целеви или прогнозирани приходи, цифри за продажби или минали от различни периоди, както и други данни, свързани с инвентара или операциите. Системата за подпомагане на решения събира и анализира данни, синтезирайки ги, за да произведе обширни информационни отчети. По този начин, като информационно приложение, система за подпомагане на решения се различава от обикновено приложение за операции, чиято функция е просто да събира данни. Тази система може да бъде използвана от управлението на операции и други планови отдели в организацията, за да събере информация и данни и да ги синтезира в действащ разумен интелект. Фактически тези системи се използват предимно от средно и горно управление.

## **1.3. Системи за управление на процесите**

Системи за управление на процесите [3], понякога наречени и системи за промишлено управление, действат като части от оборудването по производствени линии по време на производството, извършвайки тестове на процеса по различни начини и предоставяйки данни за наблюдение и отстраняване на проблеми. Съществуват много видове системи за управление на процесите, включително надзор и събиране на данни (SCADA), програмируеми логически контролери (ПЛК) или системи за разпределено управление, които работят за събиране и предаване на данни, получени по време на производствения процес. Системата за управление на процесите може да бъде относително проста сензорна система, която получава вход, заедно с контролер, който обработва входа, и приемник, който обработва изхода. По-сложните устройства за управление на процесите са роботизирани и изпълняват много задачи. Тези устройства могат да предават своите данни на компютърната система за управление на предприятието на компанията чрез софтуер на средно ниво, наречен система за изпълнение на производството.

## **1.4. Системи за мениджмънт на училище/университет**

Системите за управление на училища или университети [4] (School/University Management Systems) са комплексни софтуерни приложения, предназначени да подпомагат и оптимизират административните и образователни процеси в образователните институции. Тези системи предоставят интегрирани решения за управление на учебния процес, административни задачи и комуникация с ученици, родители, преподаватели и персонал.

Основни функционалности на системите за управление на училище/университет:

1. **Регистрация и административни задачи – електронни регистрационни формуляри. Те дават в**ъзможност за електронна регистрация и попълване на формуляри за записване, както и у**правление на ученически данни. Използват за с**ъхранение и управление на данни за учениците, включително лични данни, класове, успехи и др.
2. **Управление на учебния процес – състои се от разписание на учебните занятия**, създаване на разписания за класове, преподаватели и ученици, о**ценяване и следене на успеха,** записване и проследяване на оценките, автоматично изчисляване на средни успехи.
3. **Финансово управление - такси и финансови транзакции,** обработка на такси, таксуване, възможност за плащане онлайн, у**правление на бюджета, което включва с**ледене на финансовите средства и разпределянето на бюджета.
4. **Комуникация и сътрудничество - електронна поща и съобщения. Вътре в системата се внедрява електронна поща за по-лесна комуникация, както и се добавя портал за родители и ученици, които да имат д**остъп до информация за успехи, разписания и обявления.
5. **Служба за поддръжка и обслужване на клиенти - състои се от техническа поддръжка, която цели да** осигури помощ и техническа поддръжка за потребителите, както и о**братна връзка и отчети, което пък от своя страна всъщност се използва за с**ъздаване на отчети и предоставяне на обратна връзка.
6. **Сигурност и достъп - контрол върху достъп -** регулиране на достъпа до чувствителна информация, както и сигурност на данните, като се цели защита на личните и учебни данни на учениците и учителите.
7. **Интеграция с други системи: интеграция с ERP системи, които са „системи за управлението на предприятието“** и се използват за пълната автоматизация.
8. **Специфични функционалности за университети - електронно обучение и управление на курсове, което в себе си съдържа о**рганизация и управление на онлайн курсове, н**аучни проекти и изследвания.**

Предимства на системите за управление на училище/университет:

1. **Ефективно управление на процесите – п**о-ефективно и оптимизирано управление на учебните и административни процеси.
2. **По-добра комуникация - п**одобрена комуникация между училище, ученици, родители и преподаватели.
3. **Лесен достъп до информация - б**ърз и лесен достъп до важна информация за успехи, разписания и финанси.
4. **Автоматизация на задачи -** автоматизация на рутинните административни задачи, което освобождава време за преподаватели и администратори.
5. **По-голяма прозрачност - п**о-голяма прозрачност на учебните и финансови процеси.

Недостатъци на системите за управление на училище/университет:

1. **Необходимост от oбучение - н**якои потребители може да изпитват затруднения в усвояването на сложните функционалности, което изисква допълнително обучение.
2. **Инвестиции в инфраструктура - в**недряването и поддръжката на такава система изисква значителни финансови и технически ресурси.
3. **Сигурност -** необходимост от високи стандарти за сигурност, защото системите съдържат чувствителна лична информация.
4. **Интеграция -** интеграцията с други системи може да предизвика трудности при съвместимостта.

## **1.5. Въведение в програмните езици**

Програмният език [5] е набор от инструкции и синтаксис, които се съчетават и се използват за създаване на софтуерни програми. Някои от ключовите характеристики на програмните езици включват:

1. Синтаксис: Конкретните правила и структура, използвани за писане на код в програмния език. Наборът от правила, които управляват структурата и формата на програмния език.
2. Типове Данни: Видовете стойности, които могат да бъдат съхранени в програма, като числа, низове и булеви стойности. Класификация, която указва какъв вид данни може да съдържа променливата, като например цяло число, низ или булева (логическа) стойност.
3. Променливи: Именувани места в паметта, които могат да съхраняват стойности.
4. Оператори: Символи, използвани за извършване на операции върху стойности, като събиране, изваждане и сравнение. Ключова дума, представляваща действие или операция, извършвана върху една или повече стойности или променливи, като + (събиране), - (изваждане), \* (умножение) и / (деление).
5. Контролни Структури: използват се за контролиране на потока на програма, като if-else проверки, цикли и извиквания на функции.
6. Библиотеки и Рамки: Колекции от предварително написан код, които могат да се използват за изпълнение на общи задачи и ускоряване на разработката.
7. Парадигми: Стилът или философията за програмиране, използвана в езика, като процедурна, обектно-ориентирана или функционална.

Програмният език е формален език, който уточнява набор от инструкции за компютър, за да изпълни конкретни задачи. Той се използва за писане на софтуерни програми и приложения, както и за управление и манипулация на компютърни системи. Има много различни програмни езици, всеки със своя синтаксис, структура и набор от команди. Някои от най-често използваните са Java, Python, C++, JavaScript и C#. Изборът на програмен език зависи от конкретните изисквания на проекта, включително използваната платформа, предназначената аудитория и желания резултат. Езиците за програмиране продължават да се развиват и променят с течение на времето, като се създават нови езици и се актуализират по-старите, за да отговорят на променящите се нужди.

Основните термини в програмирането са:

1. Алгоритъм: Стъпкова процедура за решаване на проблем или изпълнение на задача.
2. Функция: Самостоятелен блок код, който изпълнява конкретна задача и може да бъде извикван от други части на програмата.
3. Управление на потока: Редът, в който се изпълняват изявления в програмата, включително цикли и условни изявления.
4. Коментар: Част от текст в програма, която се пренебрегва от компилатора или интерпретатора и се използва за добавяне на бележки или обяснения към кода.
5. Дебъгване (Коригиране/Отстраняване на грешки): Процесът на намиране и отстраняване на грешки или бъгове в програма.
6. IDE: Интегрирана среда за разработка - софтуерно приложение, предоставящо обширна среда за разработка за писане, дебъгване и тестване на код.

Предимства и недостатъци на програмните езици:

Предимства:

* Повишена продуктивност: Програмните езици предоставят набор от абстракции, които позволяват на разработчиците да пишат код по-бързо и ефективно.
* Портативност (Преносимост/Мобилност): Програми, написани на програмния език с високо ниво на абстракция, могат да се изпълняват на много различни операционни системи и платформи.
* Четливост: Добре проектираните програмните езици могат да направят кода по-четим и по-лесен за разбиране както за оригиналния автор, така и за други разработчици.
* Голяма Общност: Много програмни езици имат големи общности от потребители и разработчици, които могат да предоставят поддръжка, библиотеки и инструменти.

Недостатъци:

* Сложност: Някои програмни езици могат да бъдат сложни и трудни за усвояване, особено за начинаещи.
* Производителност: Програми, написани на програмни езици с високо ниво на абстракция, могат да се изпълняват по-бавно от програми, написани на езици с по-ниско ниво на абстракция.
* Ограничена Функционалност: Някои програмни езици може да нямат вградена поддръжка за определени видове задачи или да изискват допълнителни библиотеки, за да изпълнят определени функции.
* Фрагментация: Има много различни програмни езици, което може да доведе до фрагментация и да затрудни споделянето на код и сътрудничеството с други разработчици.

## **1.6. Миграции с Visual Studio и записът на данни в MSSQL Studio**

Всяка нова версия на Visual Studio поддържа повечето видове проекти, файлове и други ресурси. Работата с тях винаги е по-добра, когато се използват новите функционалности. Обратната съвместимост с предишни версии като Visual Studio 2019, Visual Studio 2017, Visual Studio 2015, Visual Studio 2013 и Visual Studio 2012 е почти запазена напълно, но въпреки това, поддръжката за някои видове проекти се променя с времето. По-нова версия на Visual Studio може да не поддържа определени проекти изобщо или може да изисква да актуализирането на проекта така, че той да не е вече обратно съвместим.

Съществуват няколко вида проекти, които са описани във Visual Studio 2022 или пък са създадени в по-ранни версии на програмата. Тези проекти са:

* .NET Core проекти (xproj)
* ASP.NET Core Web App и ASP.NET Core Web App с включено в него Application Insights
* C#/Visual Basic Webform или Windows Form
* Кодиран UI Test
* Database Unit Test Projects (csproj, .vbproj)
* F#
* InstallShield MSI setup
* Microsoft Azure Tools за Visual Studio
* Model-View-Controller framework (ASP.NET MVC)
* Modeling
* SQL Server Integration Services (SSIS)
* Web Deployment (wdproj)

Въпреки стремежа да бъдат поддържани съвместимостта и приложимостта на тези проекти с предишни версии на програмата, могат да се появят промени, които не са съвместими с тези проекти и тези промени биха се отразили на самият проект по негативен начин. Когато това се случи, по-нова версия на Visual Studio няма да зареди проекта или да предложи път за миграция [6]. Възможно е да трябва да поддръжка на този проект в предишна версия на Visual Studio. Понякога по-нова версия на Visual Studio може да отвори проект, но трябва да го актуализира или мигрира по начин, който може да го направи несъвместим с предишни версии. Visual Studio използва следните критерии, за да определи дали такава миграция е необходима:

* Съвместимост с целевите версии на платформите, назад до Visual Studio 2013 RTM.
* Съвместимост на активите по време на проектиране с предишни версии на Visual Studio. Visual Studio 2022 цели да се справи с увековечените активи по време на проектиране без да ги повреди, така че предишните версии да могат все още да отворят проекта.
* Дали новите активи по време на проектиране биха нарушили съвместимостта с предишни версии, назад до Visual Studio 2013 RTM & Update 5.

Понякога съвместимостта не е възможна. Тогава Visual Studio отваря „ръководството за обновление“, за да направи необходимите еднопосочни промени. Тези еднопосочни промени могат да включват промяна на свойството ToolsVersion във файла на проекта, което точно показва коя версия на MSBuild може да превърне изходния код на проекта в изпълним и разпространим. Това, което прави проекта несъвместим с предишни версии на Visual Studio, не е версията на Visual Studio, а версията на MSBuild, определена от ToolsVersion. Ако версията на Visual Studio съдържа инструменталната верига на MSBuild, която съответства на ToolsVersion в проекта, тогава Visual Studio може да извика тази инструментална верига, за да изгради проекта.

За поддържане на съвместимост с проекти, създадени в предишни версии, Visual Studio 2022 включва необходимите инструментални вериги на MSBuild, за да поддържа ToolsVersion 15, 14, 12 и 4. Проекти, които използват някоя от тези стойности на ToolsVersion, трябва да доведат до успешно изграждане. (Подлежи отново на това, дали Visual Studio 2022 поддържа типа проект, както е описано на страницата за Насочване към платформата и съвместимост.)

Освен миграции на проект, съществуват и миграции, които се извършват от проекта към базата данни, която се ползва, за да се пазят в нея, моделите на проекта. За да се направи реално приложение достъпно за използване от други хора през Интернет, трябва то да бъде съхранено в облачната платформа Azure. Миграциите са нещо, което е доста полезно и се пише в доста от случаите, за да може приложението да изпрати моделите си към базата от данни без да се губи информация. Всичко зависи от това как са зададени параметрите на моделите, дали е направен добре т.нар. „Connection string“ или „връзката към базата от данни“, която в повече от случаите е добре да съдържа името на сървърът, към който ще се пращат моделите от данни. Друго нещо, което е задължително да съдържа „нишката за връзка“ е името на базата от данни, която ще се създаде в приложението SQL Server Management Studio – SSMS. Там ще бъдат създадени всички таблици от приложението, диаграмите, които ще представляват т.нар. връзки между таблиците, Views(изгледи на базата), външни ресурси, синоними, хранилище, което ще съхранява информацията вътре в него, което може да се счита дори и за копие на тази информация. Също така базата от данни ще съдържа и някакъв вид сигурност – Security. Тази сигурност ще запази в себе си кои потребители са влизали в нея, ролите на тези потребители, схеми, ключове, сертификати, „криптирани“ ключове и полици за сигурност. Always Encrypted използва два типа криптографски ключове, за да защити данните - един ключ за криптиране на данните и друг ключ за криптиране на ключа, който криптира тези данни. Ключът за криптиране на колона криптира данните, а ключът за криптиране на майстор-колоната криптира ключа за криптиране на колоната. Друго нещо, което съдържа т.нар. „Connection string“ е опцията Trusted\_Connection или т.нар. потвърдена връзка. Доверената връзка е същото като използването на Windows Authentication (Удостоверяване чрез Windows) в SQL Server. Удостоверяването се извършва от домейна, а разрешението се обработва от SQL Server. SQL Server също така може да използва свои собствени входове, като например потребителят sa. Те се удостоверяват и авторизират от SQL Server. Те са възможни само ако SQL Server се изпълнява в режим на смесено удостоверяване (Mixed Authentication Mode). Другата опция, която е добре да бъде включена е опцията, която осигурява сертификацията на сървъра или когато „Trust\_Server\_Certificate“ опцията е настроена на true, транспортният слой ще използва SSL за криптиране на канала и ще пропусне проверката на веригата от сертификати за удостоверяване на доверието. Ако „Trust\_Server\_Certificate“ е настроено на true и е включено криптирането, нивото на криптиране, зададено на сървъра, ще се използва, дори ако Encrypt е настроено на false. В противен случай връзката ще се провали.

Освен „нишката за връзка“ между средата за разработка и приложението, в което се съдържа базата от данни, преди да се изпратят миграции към базата за данни е необходимо да се създадат т.нар. конфигурации на данните. Те ще опишат на базата точно са връзките на моделите от отделните таблици. Съществуват два вида ключови стойности: външен ключ (foreign key) и първичен ключ (primary key) [7]. Първичните ключове и външните ключове са два вида ограничения, които могат да се използват за гарантиране на цялост на данните в таблиците на SQL Server. Тези са важни обекти в базата данни.

* Таблица може да съдържа само едно ограничение за първичен ключ.
* Първичен ключ не може да включва повече от 16 колони и обща дължина на ключа от 900 байта.
* Индексът, генериран от ограничението за първичен ключ, не може да причини броя на индексите върху таблицата да надвишава 999 „неклъстерирани“ (некласифицирани) индекса и 1 „клъстериран“ (класифициран) индекс.
* Ако не е указано „клъстерирано“ или „неклъстерирано“ за ограничение за първичен ключ, се използва „клъстерирано“, ако няма „клъстериран“ индекс върху таблицата.
* Всички колони, дефинирани в рамките на ограничение за първичен ключ, трябва да бъдат дефинирани като “not null”. Ако не е посочена възможност за “nullability” (нулева стойност), всички колони, участващи в ограничение за първичен ключ, имат nullability настроено на not null.
* Ако първичен ключ е дефиниран върху колона от тип CLR потребителски дефиниран тип, реализацията на типа трябва да поддържа бинарно наредба.

Външен ключ (FK) е колона или комбинация от колони, която се използва за създаване и утвърждаване на връзка между данните в две таблиците, с цел контролиране на данните, които могат да бъдат съхранявани в таблицата с външен ключ. При връзка с външен ключ се създава връзка между две таблиците, когато колоната или колоните, които съдържат стойността на първичния ключ за една таблица, бъдат използвани от колоната или колоните в друга таблица. Тази колона става външен ключ във втората таблица. Една таблица може да има връзка с максимум 253 други таблици и колони като външни ключове (изходящи връзки). SQL Server 2016 (13.x) увеличава лимита за броя на другите таблиците и колоните, които могат да се отнасят към колони в една таблица (входящи връзки), от 253 на 10,000. (Изисква поне ниво на съвместимост 130.) Увеличението има следните ограничения:

* Повече от 253 връзки като външни ключове се поддържат само за операции по изтриване (DELETE DML операции). Операциите по актуализиране (UPDATE) и сливане (MERGE) не са поддържани.
* Таблица с външен ключ, който се отнася към себе си, остава ограничена до 253 външни ключове.
* Повече от 253 външни ключове в момента не са налични за columnstore (сървър с колонен склад) индекси, таблиците с оптимизирана памет, Stretch Database или разделени таблиците с външни ключове.

След създаването на конфигурации на данните е възможно преминаването към описването на методите, които ще направят т.нар. „посяване“ на данни в базата данни. То не е задължително да се прави преди първата миграция, която ще инициализира базата от данни, но е препоръчително, защото по този начин в базата ще се създадат модели с параметрите, които по-рано са били зададени. Първата миграция обичайно се води – „инициализация на базата“ (Initialize Database) миграция. Това „посяване“ или „засяване“ на данни се прави, като се създаде един обект от тези модели, които преди това са били създадени и този модел ползва данните на моделите. Пример: ако е създаден модел на ученик, който да има идентификатор (Id), име (First Name), фамилия (Last Name), възраст (age), клас (class), ще се използват тези данни и ще се създаде обект. Тъй като в моделите от базата се прави най-обикновена инициализация на моделите, тоест не се задават никакви параметри. Вече при “seed” или „посяване“ в базата от данни се задават и тези параметри като приемат реални стойности.

# **Глава 2 – Теоретична част**

За да бъде създадена платформата е използвана интегрирана среда за разработка “Visual Studio Enterprise 2022”.

## **2.1. IDE: Интегрирана среда за разработка – Visual Studio**

Visual Studio [8] е интегрирана среда за разработка (IDE), разработена от Microsoft за създаване на приложения за настолни компютри, графичен потребителски интерфейс (GUI), конзолни, уеб приложения, мобилни приложения, облачни и уеб услуги и др. С помощта на тази интегрирана среда за разработка е създаден управляван код, както и бинарна информация, която компилирана да се стартира на процесор. Средата за разработка (IDE) използва различни платформи на софтуерното развитие на Microsoft, като Windows Store, Microsoft Silverlight и Windows API и др. Visual Studio не е ограничена до конкретен език, тъй като тази среда може да бъде използвана за писане на код на C#, C++, VB (Visual Basic), Python, JavaScript и много други езици. Тя предоставя поддръжка за 36 различни програмни езика и е налична за Windows и macOS. Еволюцията на Visual Studio започва с Visual Studio 97 включва различни инструменти за софтуерно развитие. Най-новата версия е Visual Studio 2022, която отговаря на версия 17.8. Издадена на 8 януари 2024 г., внася доста подобрения и нови функции. Тя поддържа .Net Framework версии от 3.5 до 8.

Интегрираната среда за разработка има 3 издания:

* Community
* Professional
* Enterprise

Както вече беше споменато, за разработката на приложението е използвана изданието – Enterprise. То предоставя голям брой функционалности. Това е интегрирано, завършено решение за екипи от всякакъв размер с изисквания за високо качество и мащабируемост (производителност на мащаба). Предоставя се 90-дневна безплатна проба на това издание и след изтичане на пробния период потребителят трябва да заплати, за да продължи да го използва. Основната му предимство е, че е високо мащабируемо и доставя софтуер с високо качество.

Предимства на използването на средата за разработка Visual Studio (IDE):

1. Пълнофункционална платформа за програмиране:

Visual Studio IDE представлява пълнофункционална платформа за програмиране, поддържаща няколко операционни системи, уеб и облачни приложения. Потребителите могат лесно да преглеждат потребителския интерфейс, така че да могат бързо и точно да пишат своя код.

1. Мощен инструмент за отстраняване на грешки (Debugging):

Visual Studio предлага силен инструмент за отстраняване на грешки, който помага на разработчиците бързо да откриват потенциални грешки в кода.

1. Увереност във възможностите за хостинг на сървъра:

Разработчиците могат да поддържат (host) своето приложение на сървъра с увереност.

1. Поддръжка на жив код и автокомпилиране:

Независимо от програмния език, който разработчиците използват, Visual Studio предлага поддръжка на жив код. За по-бързо развитие, платформата предоставя опция за автокомпилиране. Вградената интелигентна система предоставя описания и съвети за API-та.

1. Лесно сътрудничество:

Чрез Visual Studio IDE разработчиците могат лесно да сътрудничат с колегите си в един и същ проект. Тази среда помага на разработчиците да споделят, изтеглят и качват своя код с колегите си.

1. Персонализация и разширяемост:

Всеки потребител на Visual Studio има възможността да я персонализира. Всеки може да добави функции в зависимост от своите нужди, като например да изтеглят добавки и инсталират разширения в своята IDE.

Разширения, които могат да се добавят към средата за разработка. Това са добавки, които позволяват персонализирането и подобряването на Visual Studio, като се добавят функционалности или пък се интегрират вече съществуващи инструменти. Разширението може да бъде от всякакъв характер на сложност, но основната му цел е да увеличи продуктивността и да отговаря на работния процес. Повечето от информацията за всяко едно приложение (разширение) може да бъде намерена в документацията на Microsoft и в GitHub. Най-често използваните разширения са: GitHub Copilot и ReSharper (JetBrains). GitHub Copilot предоставя предложения в стил довършване от изкуствен интелигентен програмист, докато програмирате. Могат да се получат предложения от GitHub Copilot при започване на писането на кода, който се използва, или чрез написване на коментар с естествен език, описващ какво точно трябва да направи кода. С това разширение се постига много лесен достъп до писането на тестове към самото приложение. Това е едно от трудните действия при направата на уебсайт. Действието, което извършва GitHub Copilot e да изпрати запитване към уебсайта, от другата страна се изпраща информация за елемент и GitHub Copilot генерира тестове към тази програма. Друго полезно действие, което може да бъде извършено от това разширение е да направи връзката между база от данни и уебсайт. Също така то може да направи и заявка към базата от данни(в. т. ч създаване на таблица, избор на определена информация от вече създадена таблица и др.). Друго подобно разширение към интегрираната среда за разработка е ReSharper. Това е популярно разширение за повишаване на продуктивността на разработчика за Microsoft Visual Studio. То автоматизира почти всичко, което може да се автоматизира. Открива грешки при компилация, грешки при изпълнение, „излишен“ код директно по време на писане, предлагайки интелигентни корекции за тях. ReSharper помага да се изследва кода, визуализирайки структурата на файловете, йерархията на типове и стил, веригите от извиквания и стойности, зависимостите между проекти. То позволява моментално да се прегледа цялото решение и насочва директно към точния файл и ред, декомпилирайки библиотечен код при необходимост. Поддръжката на разширението към средата за разработка е за C#, VB.NET, XAML, XML, HTML, CSS, ASP.NET, ASP.NET MVC, ASP.NET Core, NAnt и MSBuild скриптове, включително обширна функционалност за работа с различни езици.

Анализът на зависимости в проект (Project dependency analysis) е процес на преглед на взаимоотношенията и зависимостите между различни компоненти или модули в софтуерен проект. Тази зависимост може да бъде подготвена като структура от Resharper. Анализа помага за разбирането на различни части от софтуерен проект от разработчици и управител. ReSharper може да изгражда йерархии на проекти и да визуализира графика на зависимости между проектите, която се да запазва и чрез нея се сравнява с нови състояния на решението на различни етапи, без да има нужда от компилиране – директно от йерархията на проекта или визуалния граф на зависимости, може да се извиква навигационни действия и рефакториране на код (рефакторинг). Наборът от рефакторинги на Resharper подобрява основните възможности, предоставени в Visual Studio, по отношение на брой, удобство и обхват на приложение - преобразуване на метод в свойство, извличане на стил/ресурс, преместване в друг файл, безопасно изтриване, преобразуване на параметри. Всеки от рефакторингите анализира целия обхват на кода, върху който се прилага (който може да бъде толкова широк, колкото е голям кода), включително код на различни езици, и използва този анализ, за да актуализира структурата на кода по най-интелигентен начин. Всички рефакторинги на ReSharper работят в C#, и голямата част от тях са налични и в VB.NET, някои в XAML, HTML, ASP.NET, Razor, CSS, Resx и сценарии за създаване (MSBuild и NAnt). Рефакторингите се допълват от стотици контекстни действия, които автоматизират преструктурирането и пренаписването на код по различни начини.

## **2.2. SQL Server Management Studio – SSMS**

За база данни в приложението се използва програмата - Microsoft SQL Server Management Studio (SSMS) [9], което е софтуерно приложение, разработено от Microsoft. Използва се за конфигуриране, управление и администриране на всички компоненти в Microsoft SQL Server. Първоначално представено с Microsoft SQL Server 2005, това е наследникът на Enterprise Manager в SQL 2000 или преди това. Инструментът включва както текстови редактори, така и графични инструменти, които работят с обекти и функционалности на сървъра.

Централна част на SSMS е Object Explorer, който позволява на потребителя да преглежда, избира и действа върху всички обекти в сървъра. Също така се предоставя и отделна експрес версия, която може да бъде свободно изтеглена; обаче последните версии на SSMS са напълно способни да се свързват и управляват всяка SQL Server Express инстанция. Microsoft също включи обратна съвместимост с по-стари версии на SQL Server, позволявайки по-нова версия на SSMS да се свързва към по-стари инстанции на SQL Server. Също така идва с Microsoft SQL Server Express 2012, или потребителите могат да го изтеглят отделно.

Започвайки от версия 11, приложението е базирано на Visual Studio 2010 shell, използвайки WPF за потребителския интерфейс. Версиите 18 и следващи са базирани на Visual Studio 2017 Isolated Shell. През юни 2015 г., Microsoft обявява намерението си да пуска бъдещи версии на SSMS независимо от изданията на SQL Server database engine.

SQL Server Management Studio (SSMS) е интегрирана среда за управление на всяка SQL инфраструктура. SQL Server Management Studio се използва, за да се достъпват, конфигурират, управляват, администрират и разработват всички компоненти на SQL Server, Azure SQL Database, Azure SQL Managed Instance, SQL Server on Azure VM и Azure Synapse Analytics. SSMS предоставя единен комплексен инструмент, който комбинира разнообразие от графични инструменти с множество богати текстови редактори, осигурявайки достъп до SQL Server за разработчици и администратори на бази данни с различни нива на умения. Този инструмент се използва за бизнес интелигенция – за да се достъпват, управляват и администрират услуги като анализ, интеграционни услуги, както и услуги, чрез които се осигуряват отчети - "Report service" (система за отчети) обикновено се отнася до софтуерна услуга или инструмент, който предоставя възможност за създаване, конфигуриране и управление на различни видове отчети и анализи. Тези услуги често се използват в областта на базите данни и информационните технологии. В контекста на SQL Server Management Studio (SSMS), това може да включва вградените средства за създаване и управление на отчети върху данни, съхранявани в SQL Server или други свързани услуги като Azure SQL Database или Azure Synapse Analytics. Такива "report service" обикновено предоставя възможности за форматиране на отчетите, визуализация на данни и споделяне на генерирани отчети с други потребители. Въпреки че всички три технологии за бизнес интелигентност ползват от SQL Server Management Studio (SSMS), административните задачи, свързани с всяка от тези технологии, са леко различни.

SSMS позволява да бъдат управлявани обекти в Analysis Services, като се запазват резервни копия и се обработват обекти. Management Studio предоставя проект за скриптове за Analysis Services, в който може да бъдат разработвани и запазвани скриптове, написани на Multidimensional Expressions (MDX), Data Mining Extensions (DMX) и XML for Analysis (XMLA). Използвайки проекти за скриптове за Analysis Services, за да се извършват управленски задачи или да се създават отново обекти като бази данни и кубове на инстанциите на Analysis Services. Например: може да се разработят XMLA скрипт в проект за скриптове за Analysis Services, който създава нови обекти директно върху съществуваща инстанция на Analysis Services. Проектите за скриптове за Analysis Services могат да бъдат запазвани като част от решение и интегрирани с контрол на изходен код.

XML (Extensible Markup Language) е универсален стандартен език за маркиране, използван за представяне и обмен на структурирани данни между компютърни системи. Той предоставя гъвкав начин за организиране на информация и структуриране на данни, като използва тагове (маркери) за определяне на различни елементи в документа.

Основни характеристики на XML включват:

1. Разширяемост (Extensibility): Позволява на потребителите да дефинират свои собствени тагове и атрибути, които отговарят на конкретните им нужди и структури на данни.
2. Четимост от човек (Human-readable): XML документите са лесни за четене и разбиране от хора, тъй като текстът е лесно разпознаваем и структуриран.
3. Иерархичност: Данните се организират в дървовидна структура с родителски и децентни елементи, което улеснява представянето на взаимосвързана информация.
4. Независимост от платформа: XML може да бъде използван на различни операционни системи и архитектури.

XML се използва в различни области, като уеб разработка, обмен на данни между приложения, съхранение на конфигурационни файлове и други сценарии, където структурирането на данни е от съществено значение.

XMLA (XML for Analysis) е стандарт за обмен на данни и изпълнение на команди в областта на анализа на данни и бизнес интелигентност. Този стандарт се използва основно в контекста на управление и манипулиране на данни в сървъри за анализ, като например Microsoft SQL Server Analysis Services (SSAS).

XMLA скриптовете са XML-базирани скриптове, които се използват за автоматизиране на управленски задачи и операции в средата на Analysis Services. Тези скриптове могат да бъдат използвани за създаване, модификация и управление на обекти като кубове, измерения, виртуални кубове и други елементи на анализа.

Multidimensional Expressions (MDX) [10] е заявков език, който се използва за работа с многомерни данни и тяхното извличане в Microsoft SQL Server Analysis Services. MDX се базира на спецификацията XML for Analysis (XMLA), като има специфични разширения за SQL Server Analysis Services. В MDX се използват изрази, съставени от идентификатори, стойности, оператори, функции и операции, които SQL Server Analysis Services може да оцени, за да извлече обект (например множество или член) или скаларна стойност (например низ или число).

MDX заявки и изрази в SQL Server Analysis Services се използват за следните цели:

* Връщане на данни към клиентско приложение от куб в SQL Server Analysis Services.
* Форматиране на резултатите от заявките.
* Извършване на задачи по дизайн на кубове, включително дефиниране на изчислени членове, наименовани множества, определени обхвати и ключови показатели на производителност (KPI).
* Извършване на административни задачи, включително управление на сигурността на измерения и клетки.

MDX е повърхностно подобен в много отношения на SQL синтаксиса, който обикновено се използва с релационни бази данни. Въпреки това, MDX не е разширение на SQL езика и се различава от него по много начини. За да се създават MDX изрази за дизайн или сигурност на кубове, или да се създават MDX заявки за извличане и форматиране на многомерни данни, е необходимо да се оценяват и разбират основни концепции в MDX и многомерното моделиране, синтаксисни елементи на MDX, оператори, заявки и функции на MDX.

Data Mining Extensions (DMX) [11] е език, който се използва за създаване и работа с модели за обработка на данни в Microsoft SQL Server Analysis Services. Възможно е да се използва DMX за създаване на структурата на нови модели за обработка на данни, за обучение на тези модели, както и за преглеждане, управление и да се предскаже срещу тях. DMX се състои от заявления на езика за дефиниране на данни (DDL), заявления на езика за манипулиране на данни (DML), и от функции и оператори.

Спецификацията Microsoft OLE DB за обработка на данни в областта на дата-майнинг - възможностите за обработка на данни в Analysis Services са изградени в съответствие със спецификацията Microsoft OLE DB за обработка на данни в областта на дата-майнинг. Спецификацията Microsoft OLE DB за обработка на данни в областта на дата-майнинг дефинира, ч че това е структура, която съдържа информацията, определяща дата-майнинг модел и в същото време е език за създаване и работа с дата-майнинг модели. Спецификацията определя основата на дата-майнинга като виртуален обект за модела на обработка на данни. Обектът за модел на обработка на данни капсулира всичко, което е известно за конкретен модел на дата-майнинг. Обектът за модел на обработка на данни е структуриран като SQL таблица, с колони, данни и мета-информация, описваща модела. Тази структура позволява да се използва езика DMX, който е разширение на SQL, за създаване и работа с модели. Има възможност да се използват DMX заявки, за да се създават, обработват, изтриват, копират, преглеждат и правят прогнози срещу модели за обработка на данни. Има два типа заявки в DMX: заявки за дефиниране на данни и заявки за манипулиране на данни. Всеки от типовете заявки се използва за изпълнение на различни видове задачи. Следните раздели предоставят повече информация за работата с DMX заявки:

* Заявки за дефиниране на данни
* Заявки за манипулиране на данни
* Основи на заявките

Управление на решенията за Integration Services с помощта на SQL Server Management Studio - SSMS позволява да бъде използвана услугата за Integration Services за управление на пакети и наблюдение на работещи пакети. Също така може да бъде ползван Management Studio, за да се организират пакетите в папки, за стартиране на пакети, внасяне и изнасяне на пакети, мигриране [12] на пакети от Data Transformation Services (DTS) и обновяване на пакети за Integration Services. Пакетът за Data Transformation Services (DTS) е основен логически компонент на DTS; всеки обект в DTS е наследник на компонента "пакет". Пакетите се използват всеки път, когато се модифицират данни с помощта на DTS. Всички метаданни за преобразуването на данните са включени в пакета. Пакетите могат да бъдат запаметени директно в SQL Server или могат да бъдат запазени в Microsoft Repository или в COM файлове. SQL Server 2000 също позволява да се запазват пакетите във Visual Basic или в файл на друг език, когато се запазват във VB файл. Изпълнява се VB скрипт за динамичното създаване на обекти в пакета и неговите компоненти.

Един пакет може да съдържа произволен брой обекти за връзка, но не е задължително да съдържа такива. Те позволяват на пакета да чете данни от всеки източник на данни, съвместим с OLE DB, и могат да бъдат разширени, за да обработват и други видове данни. Функционалността на пакета се организира в задачи и стъпки. DTS Task е отделен набор от функционалности, изпълняван като една стъпка в DTS пакет. Всяка задача определя работна задача, която трябва да бъде изпълнена като част от процеса на преместване и трансформация на данни или като работа, която трябва да бъде изпълнена.

Data Transformation Services предоставя редица задачи, които са част от обекта на модела на DTS и които могат да бъдат достъпвани графично чрез DTS Designer или програмно. Тези задачи, които могат да бъдат конфигурирани индивидуално, покриват разнообразни сценарии за копиране на данни, трансформация на данни и известия. Например, следните видове задачи представляват някои действия, които може да бъдат извършвани с DTS: изпълнение на едно SQL изявление, изпращане на имейл и трансфер на файл с FTP.

Стъпка в рамките на DTS пакет описва реда, в който се изпълняват задачите и предварителните конструкции, които описват какво да се прави в случай на повреда или неуспех. Тези стъпки могат да бъдат изпълнявани последователно или паралелно. Пакетите могат също така да съдържат глобални променливи, които могат да бъдат използвани в целия пакет. SQL Server 2000 позволява входни и изходни параметри за задачите, което значително разширява полезността на глобалните променливи. DTS пакетите могат да бъдат редактирани, защитавани с парола, планирани за изпълнение и възстановявани по версия.

## **2.3. GNU/Linux – Ubuntu / Apache Server**

***GNU*** [13] е създаден с цел да стане преработка на цялата операционна система UNIX, която да може да се разпространява свободно.

Въпреки че пренаписването на милиони редове код може да изглежда страшно за един или двама души, разпределените усилия на десетки или дори стотици програмисти направиха проекта възможен. Трябва да се знае, че UNIX е проектиран да се изгражда от отделни части, които могат да бъдат свързани помежду си. Понеже те възпроизвеждаха команди и инструменти с добре известни, публикувани интерфейси, това усилие можеше лесно да се раздели между много разработчици.

Не само се постигат същите резултати с новия код, но в някои случаи този код е по-добър от оригиналните версии на UNIX. Понеже всеки може да вижда кода, който се произвежда за проекта, лошо написаният код може бързо да бъде коригиран или заменен с времето.

GNU обществената лицензия включва следните основни функции:

1. Правата на автора: Оригиналният автор запазва правата върху софтуера си.
2. Свободно разпространение: Хората могат да използват GNU софтуера в своя собствен софтуер, да го променят и разпространяват както желаят. Те обаче трябва да включат изходния код в разпространението си (или да го направят лесно достъпен).
3. Запазване на авторското право: Дори ако се препродаде софтуера, трябва да се запази оригиналното GNU споразумение със софтуера, което означава, че всички бъдещи получатели на софтуера ще имат възможност да променят изходния код.

Не се предоставя гаранция за GNU софтуера. Ако нещо се обърка, оригиналният разработчик на софтуера няма задължение да оправя проблема. Въпреки това много организации, големи и малки, предлагат платена поддръжка (често във форма на абонамент) за софтуера, когато той е включен в техните Linux [14] или други разпространения на отворен код.

Лицензите за отворен код, използвани от проекти за разработка на софтуер, трябва да отговарят на тези критерии, за да бъдат приети като софтуер с отворен код от OSI. Около 70 различни лиценза са приети от OSI да бъдат използвани за етикетиране на софтуер като "OSI Certified Open Source Software". Освен GPL, други популярни лицензи, одобрени от OSI, включват следните:

1. LGPL: Малката обществена лицензия на GNU (LGPL) често се използва за разпространение на библиотеки, от които зависят други приложения.
2. BSD: Лицензът за разпространение на софтуер на Университета на Бъркли позволява разпространението на изходен код, като се изисква изходният код да запази уведомлението за авторското право на BSD и да не използва имената на сътрудниците за одобрение или реклама на производен софтуер без писмено разрешение. Голямо разлика спрямо GPL е, че BSD не изисква хората, които променят кода, да предават тези промени на общността. Вследствие на това фирми със собствен софтуер като Apple и Microsoft са използвали BSD код в собствените си операционни системи.
3. MIT: Лицензът MIT е като BSD лиценза, с изключение, че не включва изискването за одобрение и реклама.
4. Mozilla: Лицензът на Mozilla покрива използването и разпространението на изходен код, свързан с уеб браузъра Firefox и друг софтуер, свързан с проекта на Mozilla. Той е много по-дълъг от останалите споменати, защото съдържа повече дефиниции за това как би трябвало да се държат сътрудниците и тези, които използват отново изходния код. Това включва представяне на файл с промените при представянето на модификации и че тези, които правят свои собствени добавки към кода за разпространение, трябва да бъдат осведомени за патентни проблеми или други ограничения, свързани с техния код.

Една дистрибуция на Linux се състои от компонентите, нужни за създаването на работеща Linux система, и процедурите, нужни за инсталирането и стартирането на тези компоненти. Технически, Linux е всъщност само ядрото (kernel). Преди ядрото да бъде полезно, трябва да е в наличност друг софтуер, като основни команди (GNU инструменти), услуги, които се предлагат (като отдалечен вход или уеб сървъри), и може би интерфейс за работен плот и графични приложения. След това трябва да се съберат и инсталират на твърдия диск на компютъра.

С изключение на някои забележими изключения (като Gentoo, Slackware и още няколко), мнозинството дистрибуции на Linux използват един от двата формата за пакетиране — DEB и RPM:

* DEB (.deb) пакетиране: Проектът Debian GNU/Linux създава .deb формата за пакетиране, която се използва от Debian и други дистрибуции базирани на Debian (като Ubuntu, Linux Mint, KNOPPIX и други). С помощта на инструменти като apt-get, apt и dpkg, дистрибуциите на Linux могат да инсталират, управляват, обновяват и премахват софтуер.
* RPM (.rpm) пакетиране: Първоначално наречен Red Hat Package Manager, но по-късно рекурсивно преименуван на RPM Package Manager, RPM е предпочитаният формат за пакетиране за SUSE, Red Hat дистрибуциите (RHEL и Fedora) и тези, базирани на Red Hat дистрибуциите (като CentOS, Oracle Linux и други). Командата rpm е първият инструмент за управление на RPM пакети. По-късно, е добавен yum за подобряване на възможностите на RPM, а сега dnf е по пътя да замести yum в бъдеще.

Debian GNU/Linux е една от най-популярните дистрибуции за Linux сървъри поради предлаганата стабилност. Дистрибуцията е резултат от доброволен труд за създаване на свободна и висококачествена дистрибуция с набор от приложения.

Debian e първата Linux дистрибуция, която включва система за управление на пакети, наречена dpkg (Debian Package), за лесна инсталация, актуализация и премахване на софтуер. Също така е първата дистрибуция, която може да бъде обновявана без да се изисква повторна инсталация.

***Ubuntu*** е най-популярната дистрибуция за настолни компютри. Тя използва същия пакетен мениджър (dpkg) като Debian. Ubuntu предлага по-бърз жизнен цикъл на изданията в сравнение с Debian, като предоставя най-новите версии на софтуера. Компанията Canonical, издател на Ubuntu, предлага комерсиална поддръжка.

Ubuntu предлага различни варианти за различни използвания:

* Ubuntu Desktop: Този вариант е най-популярният за настолни компютри и лаптопи. Предлага гарантирана поддръжка за пет години с безплатни актуализации за сигурност и поддръжка.
* Ubuntu Pro: Този вариант е оптимизиран и сертифициран за облачни доставчици като Amazon AWS, Microsoft Azure, Google Cloud Platform, IBM Cloud и Oracle.
* Ubuntu Core: Тази опция е оптимизирана за Интернет на нещата (IoT) и Edgе устройства.
* Ubuntu Server: Предлага се поддръжка, както беше описано по-рано, за 10 години. Използва се за инсталиране в частни или обществени данни центрове и поддържа множество архитектури.

***Apache***

Когато уеб браузър поиска файл, уеб сървърът ще обработи заявката, ще потърси местоположението на искания файл и след това ще отговори с намереното. На пръв поглед това е доста просто. Разбира се, зад кулисите се случват много невидими действия.

Всички тези решения са направени за по-добра производителност на уеб сървъри с висока посещаемост, където милисекундите се натрупват бързо при голям брой връзки. За създаване на развойна среда обаче, Apache [15]сам по себе си е повече от достатъчен. Никога няма да има проблеми относно производителността на сървъра или високите натоварвания поради много едновременни посетители на сайта в средата.

***Създаване на нова директория***

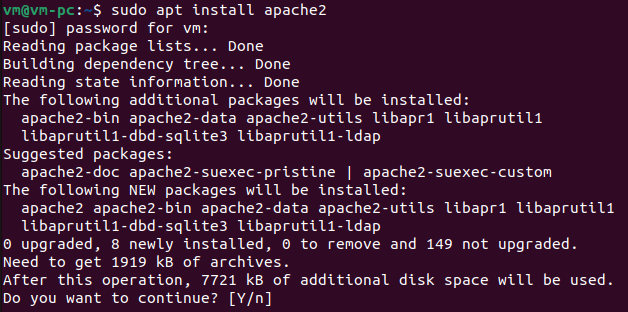
След отваряне на терминала и въвеждането на следната команда в macOS или Linux: **cd ~/**, ще се отиде до домашната директория на системата. В Windows трябва да се стигне до корена на диск C. Вместо това, може да се въведе: **cd \**

След като вече е на правилното място може да се създаде директория за сайтове, въвеждайки следната команда: **mkdir sites**, следвана от тази команда за преминаване към новата директория: **cd sites**. Накрая, трябва да се създаде директория за основния сайт. Предишните две стъпки (**mkdir** и **cd**), са за създаване на директория, наречена **primary**.

Едно от предимствата на DNS е, че няма строга 1:1 съответствие на домейн име към IP адрес. Множество домейн имена могат да бъдат присвоени на един IP адрес, което позволява един сървър да обработва множество уебсайтове. Тъй като всяко домейн име споделя един хост, те се наричат виртуални хостове. Това позволява на една инстанция на Apache да обработва множество уебсайтове едновременно, без да се налага настройване на отделни сървъри.

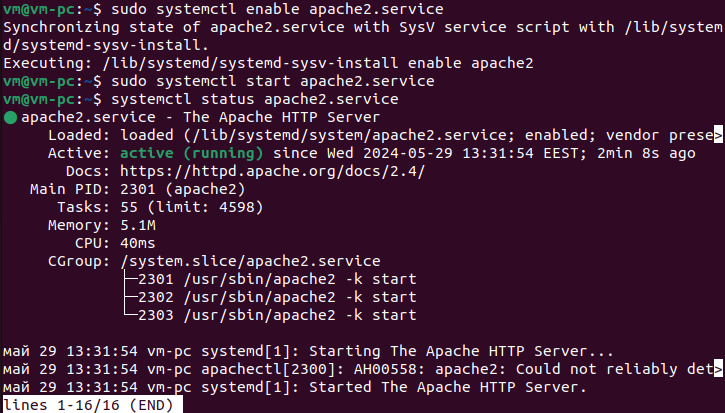
# **Глава 3 – Инженерно решение на поставената задача**

На фиг.3.1 е показана инсталацията на Apache сървър на конфигурация, използваща GNU/Linux.



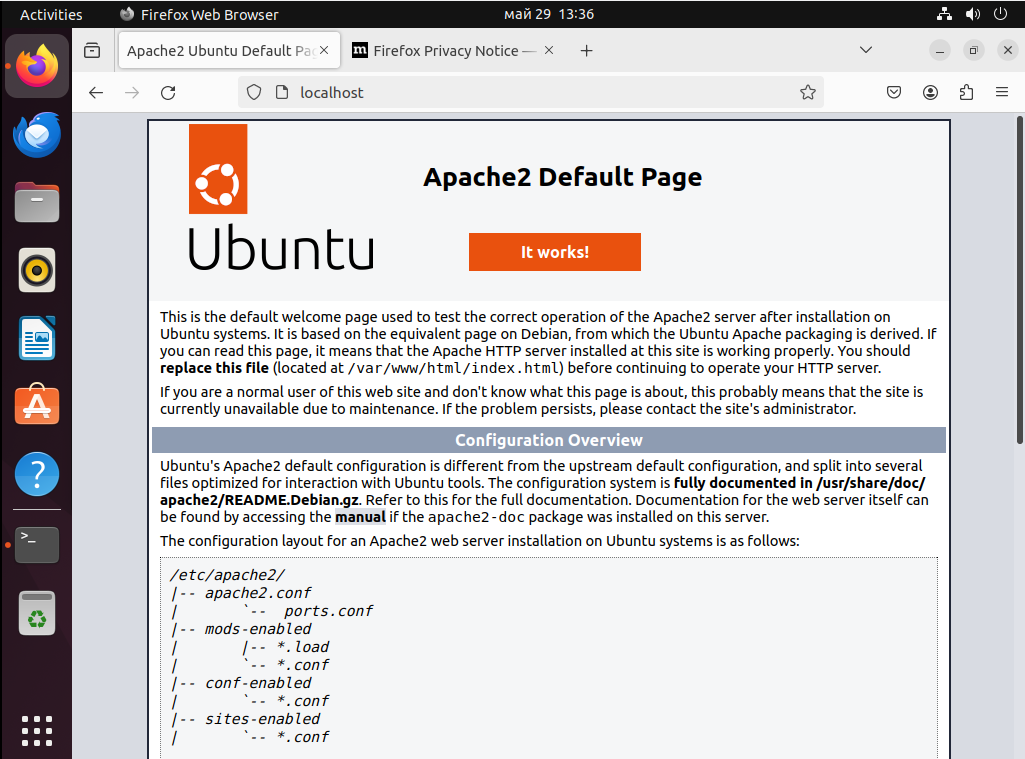
*Фиг.3.1. Инсталиране на Apache уеб сървър*

Фигура 3.2 показва осигуряването на автоматично стартиране на уеб сървъра като фонова услуга и проверка на текущия ѝ статус.



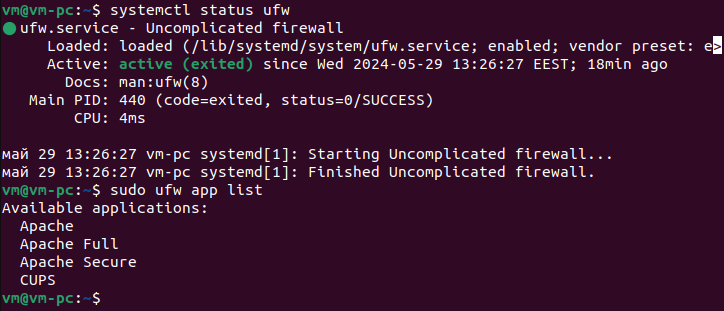
*Фиг.3.2. Осигуряване на автоматично стартиране и статус на Apache сървъра*

Фигура 3.3 показва страницата по подразбиране, която се стартира при отваряне на уеб браузър.



*Фиг.3.3. Стартиране на страница по подразбиране на Apache сървъра*

Осигурена е защита на сървъра, който ще хоства електронната платформа чрез използване на защитна стена, което е показано на фиг.3.4.



*Фиг.3.4. Проверка на статус на защитната стена*

Приложението **School Management System** е изградено по максимално улеснен и разбираем начин. Има няколко функционалности, които са напълно достатъчни, за да покрият всичко необходимо за осъществяване на идеята на приложението, а именно приложението да предостави възможност на потребителите да лесно и достъпно да управляват оценките и да следят отсъствията си.

За проектирането на уеб приложението **School Management System**, което е предназначено за употреба при търсене на книги онлайн, е използвано Microsoft Visual Studio, ASP.NET Core. Версията на .NET, с която е реализиран проектът, е .NET 6.0. Версията е сравнително нова – появява се през ноември 2021г. и се очаква да бъде заменена с .NET 7.0 през ноември 2022г.

ASP.NET Core е безплатна рамка за уеб приложения с отворен код, разработена от компанията Microsoft. Рамката ASP.NET Core поддържа успоредно управление на версиите, така че различните приложения, които се разработват на една машина, могат да се насочат към различни версии на ASP.NET Core. Това не е възможно с предишни версии на ASP.NET.

ASP.NET Core позволява използване на езика за програмиране С# в пълния му потенциал. В основата на ASP.NET Core стои концепцията на "модел-изглед-контролер" (MVC). MVC е архитектурен шаблон за уеб приложения, разработен отново от Microsoft, който позволява разделение на начините на представяне на информацията. В случая на Web API (Application Programming Interface), няма изрично използване на изгледи (views), тъй като API-то предоставя данни и резултати вместо представления на информация пред потребителя.

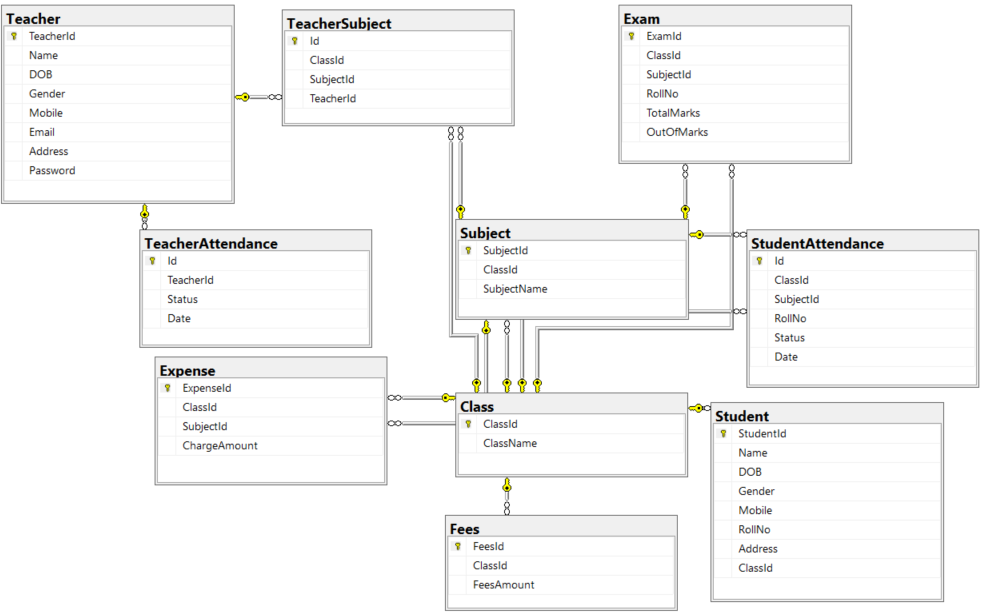
В ASP.NET Core Web API:

* Класовете модели представляват структури от данни, които се използват за представяне на информацията, която API-то обработва или връща. Моделите управляват данните, логиката и правилата на приложението.
* Контролерите са класове, които приемат HTTP заявките от клиента и ги обработват. Те съдържат методи, които се изпълняват при получаване на определени HTTP заявки и извършват нужната обработка върху моделите и другите компоненти на приложението.

ASP.NET Core Web API се фокусира върху предоставянето на данни и функционалност през уеб интерфейса, вместо на изгледи (views). Това означава, че API-то осигурява достъп до ресурсите и операциите, които могат да бъдат извършвани върху тях, като например четене, създаване, актуализиране и изтриване. Резултатите от изпълнението на заявките обикновено се връщат в желания формат, като JSON или XML, в зависимост от конфигурацията и предпочитанията на приложението.

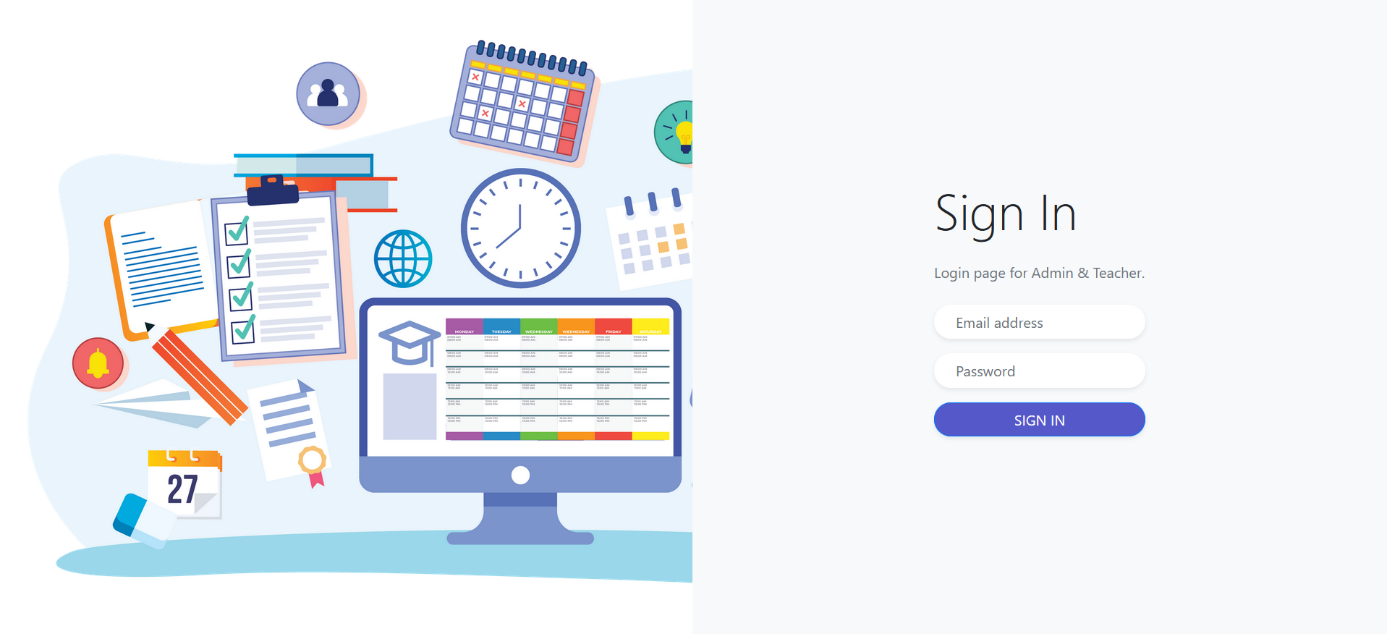
Освен Web API, за разработката на уеб приложението е използвано Microsoft SQL Server(MSSQL) за база данни, където да се съхраняват всички записи. За връзка с базата данни помага Connection String, а за работа с базата – Entity Framework Core. Използван е подхода на създаване първо на таблиците в самата база данни и след това изграждането на обектите като е възможен и обратния подход – първо да се започне с дефинирането на обектите във Visual Studio, а след това чрез миграции да се създаде базата, което още се нарича Code First. За целта на приложението обаче е ползван първият описан подход.

За приложението School Management System e избрана релационна база данни – Microsoft SQL Server. То е релационна база данни, разработена от Microsoft. Тя предоставя сигурно и надеждно съхранение и управление на структурирани данни. MSSQL се използва широко за разработка на бизнес приложения, уеб сайтове и системи за управление на данни.

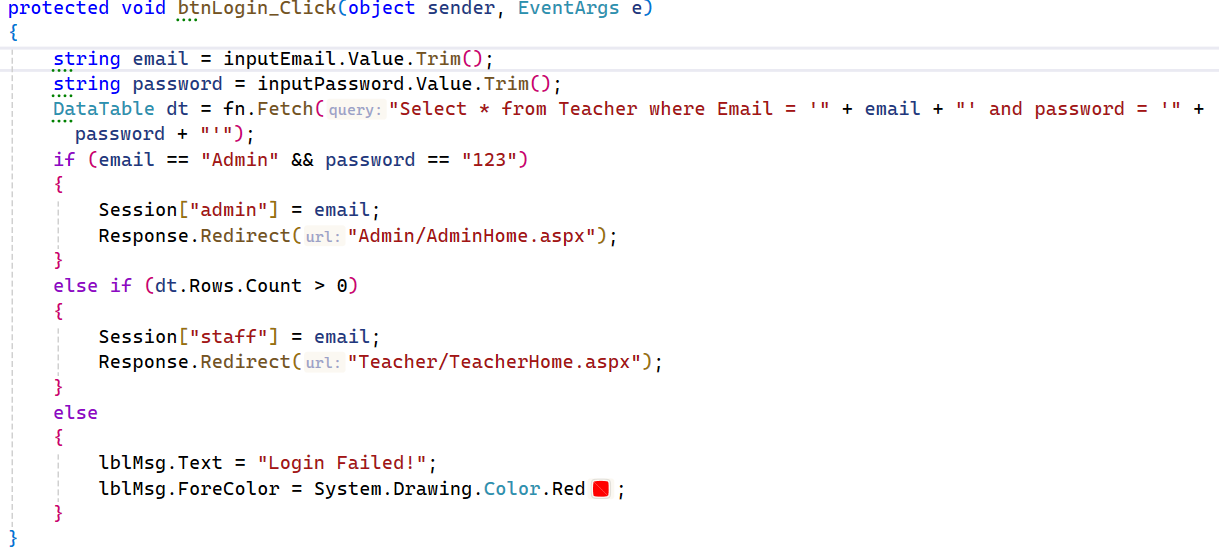


*Фиг.3.5. Блокова диаграма, съдържаща таблиците и връзките между тях*

Влизането в приложението се извършва отново в ASP.NET Core като там има 2 опции – да се използва вградената опция за влизане, въз основа на която потребител може да влезе в приложението чрез своя имейл адрес и парола и след това да достъпва своя акаунт чрез „логване” с имейл адрес и парола.

*Фиг.3.6. Началната страница, позволяваща влизане в системата*

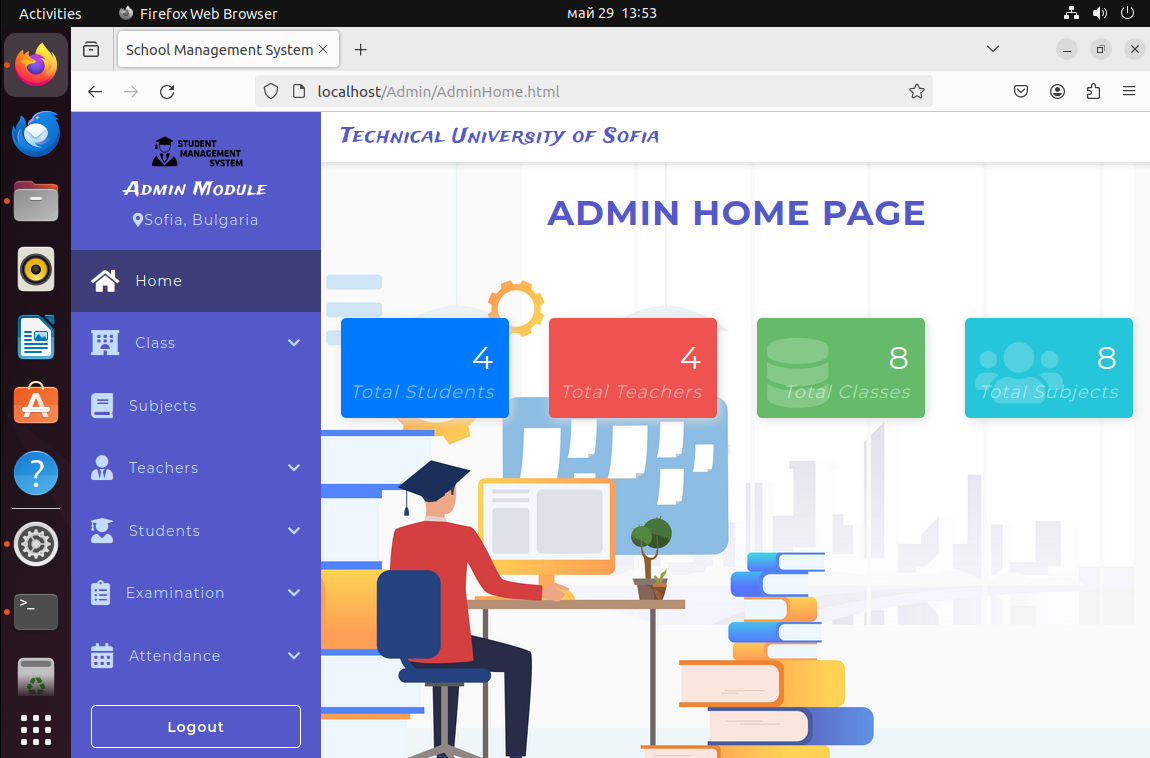
В момента, в който потребителят въведе своята парола, тя се хешира, а при влизане в системата се сравнява и съответно, ако се въведе грешна парола излиза съобщение на екрана. Опцията, която е използвана за приложението е да се ползва собственоръчно направена форма за влизане в системата, която използва функция в JavaScript – btnLogin\_Click, която „слуша“ при натискането на бутона Login, при въведени email и парола се праща Fetch заявка към базата данни и се селектира от таблицата “Teacher”, съответния email и парола, които са въведени. Ако email е Admin и паролата е “123”, значи ролята, която ще бъде назначена на съответния потребител ще бъде ролята на администратор и като отговор на заявката ще бъде върната началната страница на администратора. Другите два варианта са ролята да бъде назначена на потребителя да влезе като “staff” и като отговор на заявката ще бъде върната началната страница на учителя. Третият вариант е да не бъде открит учител или администратор регистриран през базата и съответно влизането в системата да не е успешно. Следващата фигура е пример на кода на приложението за влизане в системата.



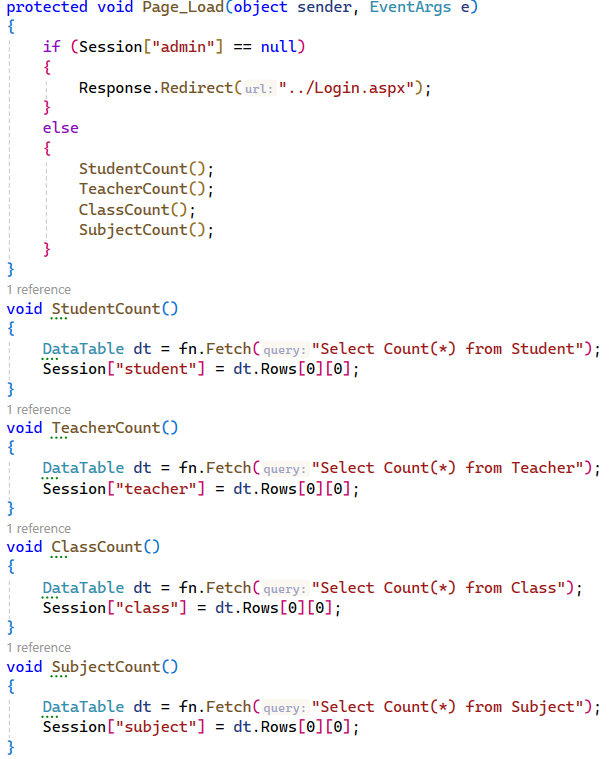
*Фиг.3.7. Изходен код за началната страница, позволяваща влизане в системата*

Правата и достъп в приложението School Management System са два типа. Първият от тях е показан на фиг.3.4. На фигурата е показана платформена независимост чрез достъпване на платформата от компютърна конфигурация, използваща Ubuntu и браузър Mozilla Firefox.

## **Администратор**

****

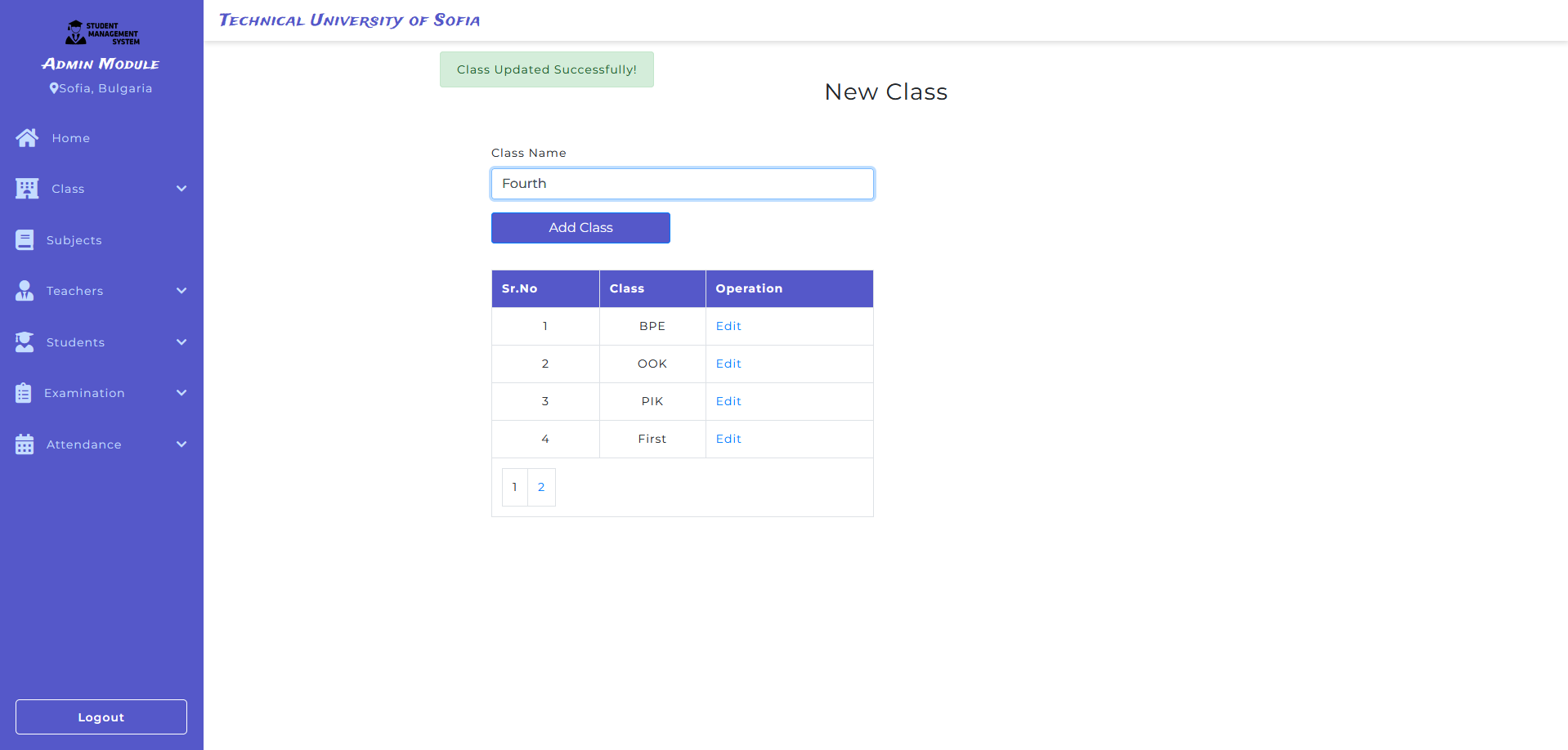
*Фиг.3.8. Начална страница на администратор*

****

*Фиг. 3.9. Изходен код за начална страница на администратор*

В случай, че потребител е с роля на администратор, на екрана на потребителя ще се появи началната страница Admin. Там могат да се видят всички права и функционалности на администратора, които са:

## *преглед добавяне на клас и добавяне на годишна такса*

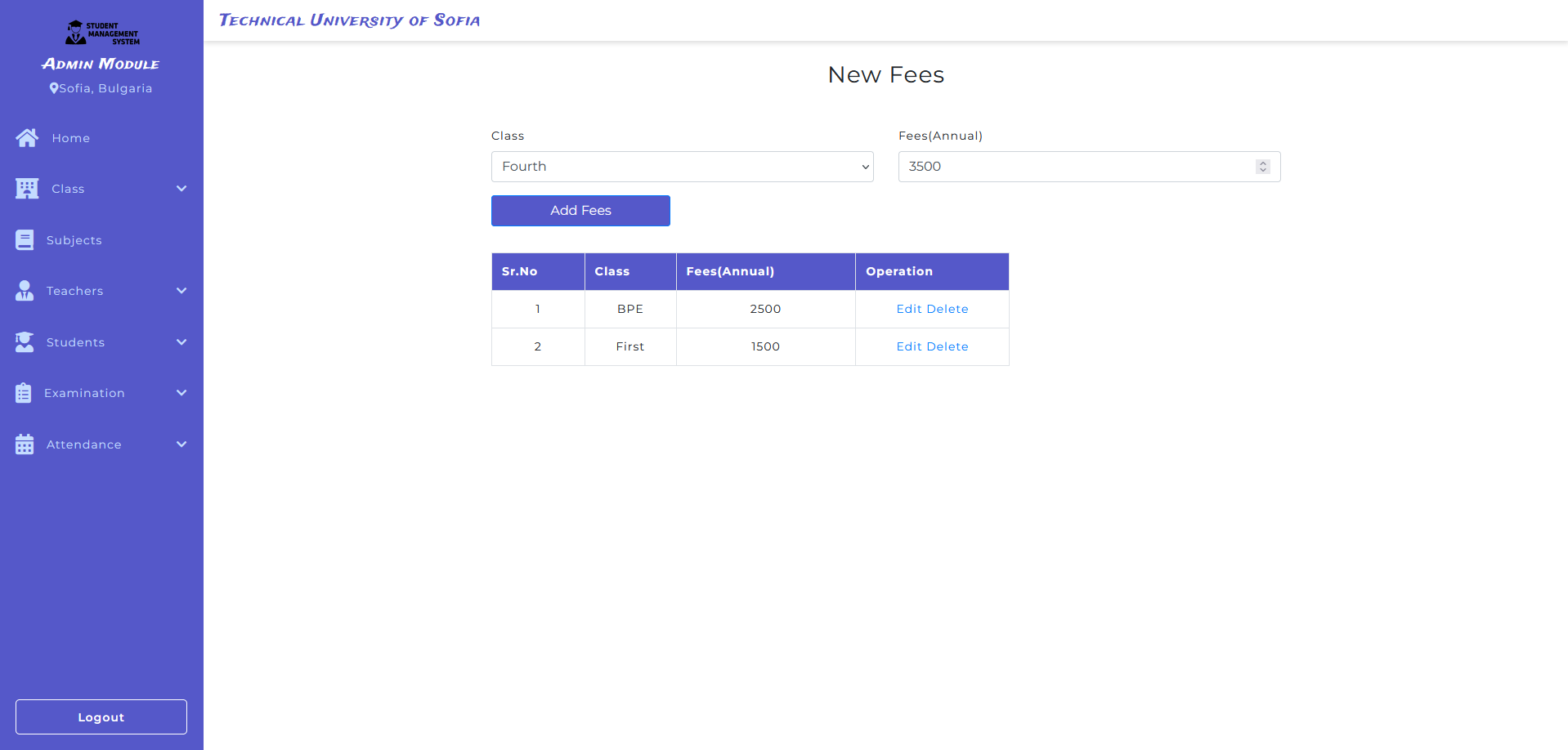


*Фиг.3.10. Страница за добавяне на нов клас*

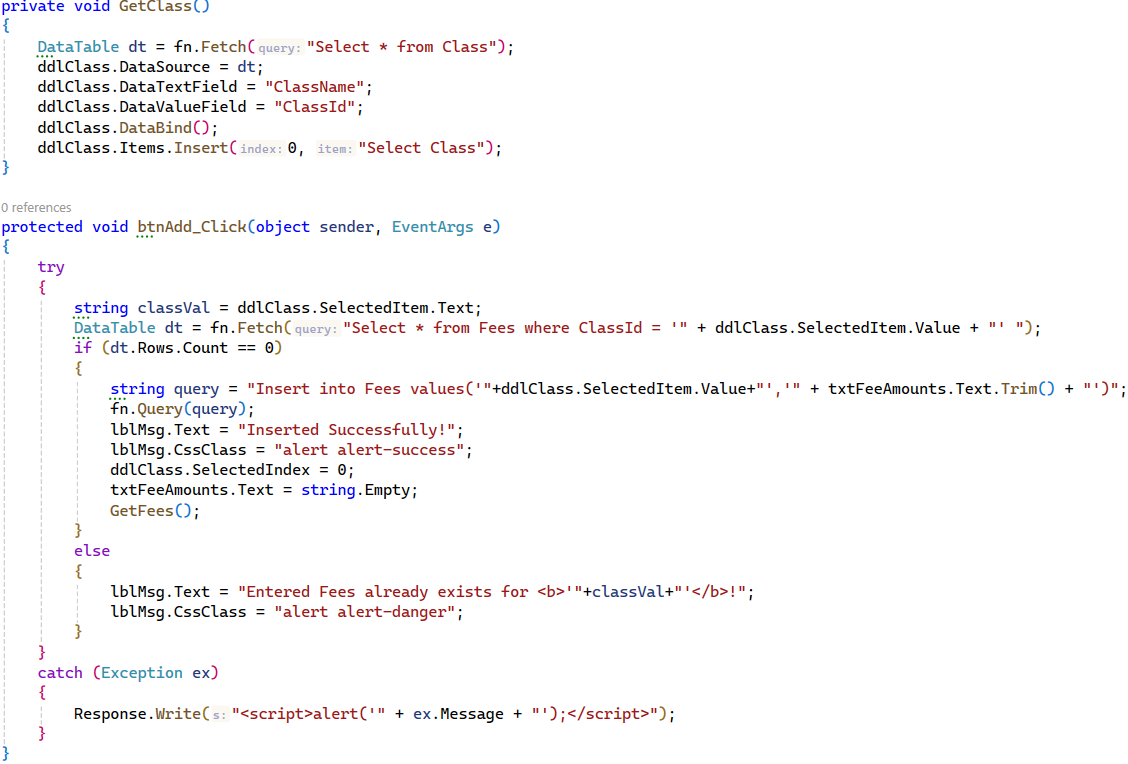


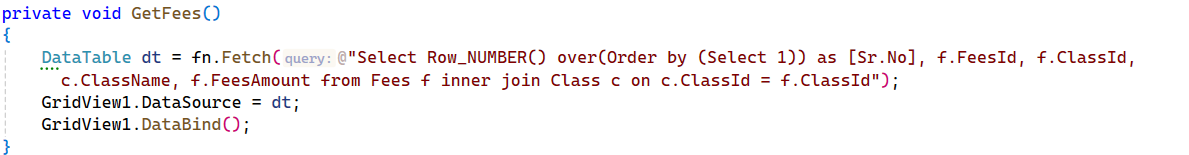
Фиг. 3.11. *Изходен код на страницата за добавяне на нов клас*

Това действие се извършва благодарение на добавянето на името на класа към таблицата с класове. След натискането на бутон “Add class” в базата данни, в таблицата с класове се добавя съответното име на класа, като при необходимост може да бъде редактирано това име чрез натискането на бутон “Edit” в таблицата в колона “Operation”. Добавянето става като чрез try-catch се прави заявка към базата, прави се проверка дали вече името не съществува и ако то съществува не се разрешава повторното му създаване, като на екрана чрез Javascript функция се изписва съобщение за грешка.



*Фиг. 3.12. Страница за добавяне на годишна такса*

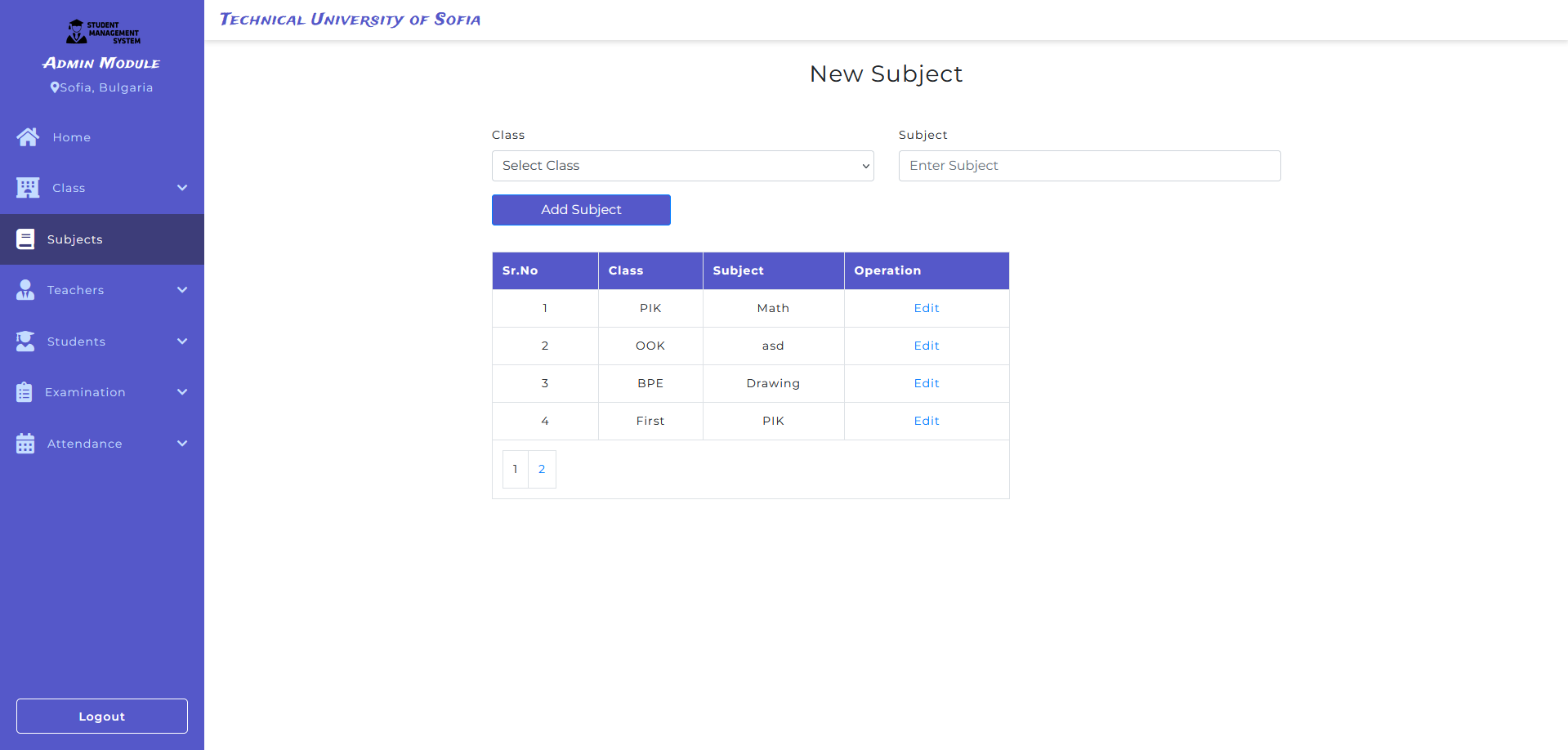




*Фиг.3.13. Изходен код на страница за добавяне на годишна такса*

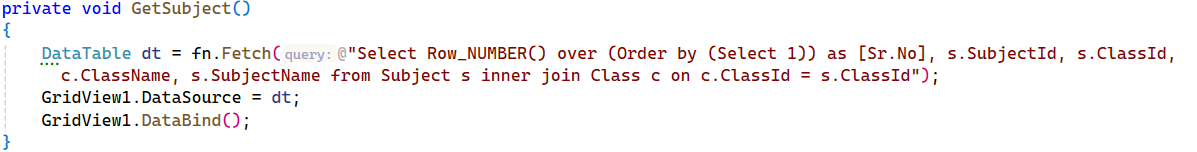
Добавянето на годишна такса се извършва чрез функция “btnAdd\_Click”, която прави заявка към базата, като чрез Fetch заявка се опитва да вземе таксите към съответния идентификатор на класа, към който се опитва да се добави таксата. Прави се проверка дали вече към този клас не е добавена стойност, която да бъде неговата такса. Ако тази проверка мине успешно и няма вече добавена такса към този клас, това означава, че успешно ще бъде добавена стойност на класа. Ако проверката не е успешна на екрана, чрез Javascript функция ще се визуализира съобщение, което ще покаже, че не е било успешно добавянето на такса към клас. При таксите е възможно освен да се редактира стойността на таксата, също така и да се изтрие самата такса, като тогава директно се изтрива стойността от базата данни и има възможност отново да се добави. Таблицата показва номера на класа, името на класа, неговата годишна такса и операцията, която може да се извърши (редакция и изтриване).

## *преглед и добавяне на учебен предмет*



*Фиг.3.14. Страница за добавяне на учебен предмет*

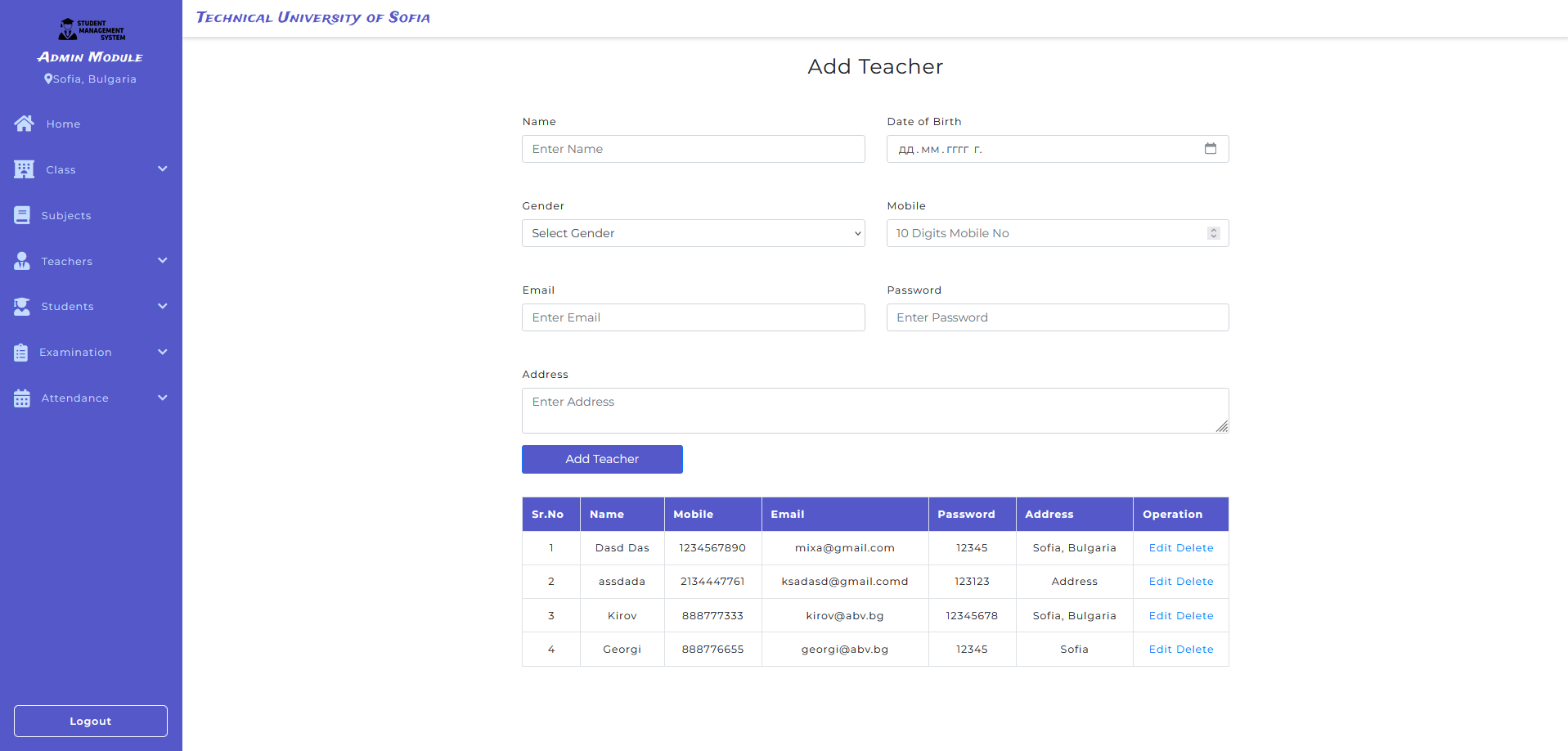




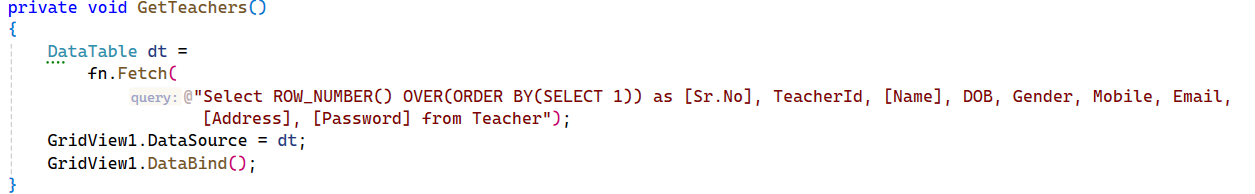
*Фиг.3.15. Изходен код на страница за добавяне на учебен предмет*

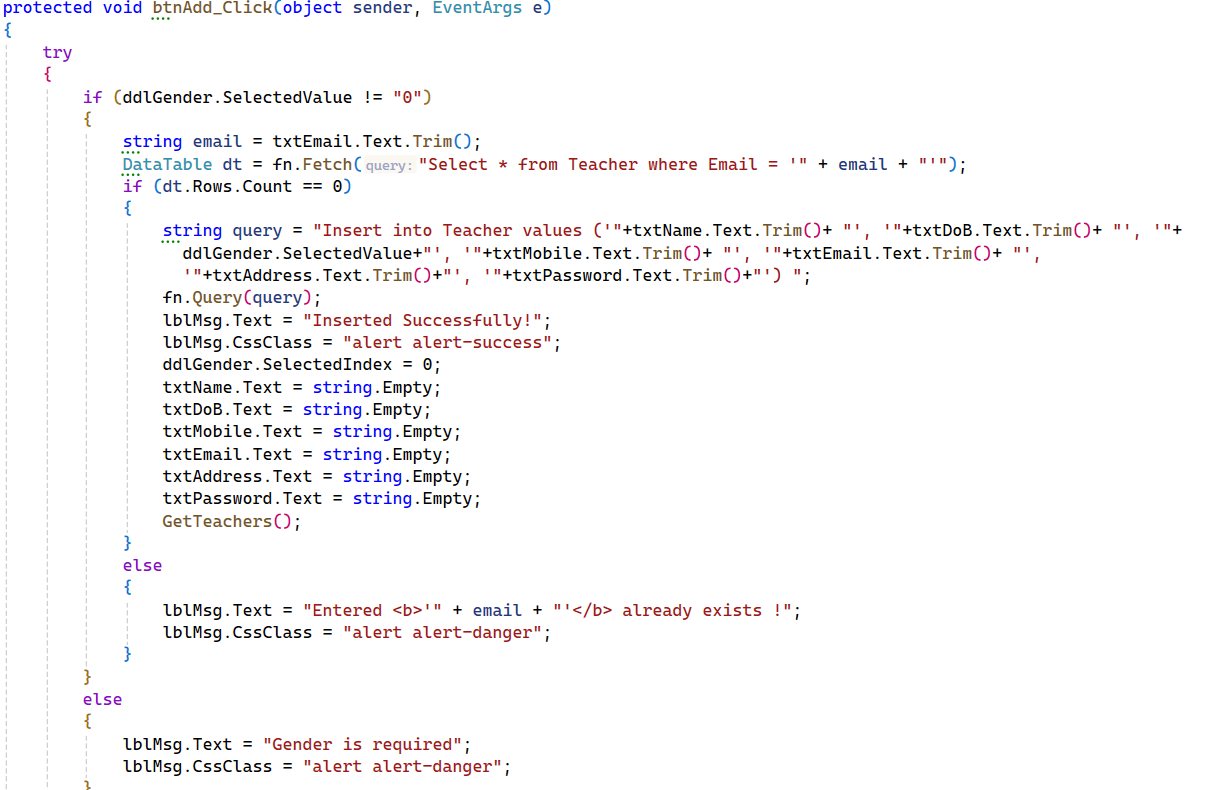
Добавянето на учебен предмет се извършва като се селектира клас, в който да се добави съответния учебен предмет и името, което трябва да бъде въведено в поле учебен предмет – “Subject”. Прави се Fetch заявка към базата данни, като се селектира от нея съответния идентификатор на клас и се добави името на учебен предмет. За да се визуализира таблицата, която е на фигурата, се използва метода “GetSubject()”, който с Fetch заявка извлича цялата информация от таблицата за учебен предмет. В таблицата се визуализират 4 колони, като те са съответно за номер на клас, името на класа, името на съответния учебен предмет, който е към този клас и операцията, която може да се извърши върху самия учебен предмет, а тя е именно редакция на името на предмета.

## *преглед и добавяне на учител*



*Фиг.3.16. Страница за добавяне на учител*

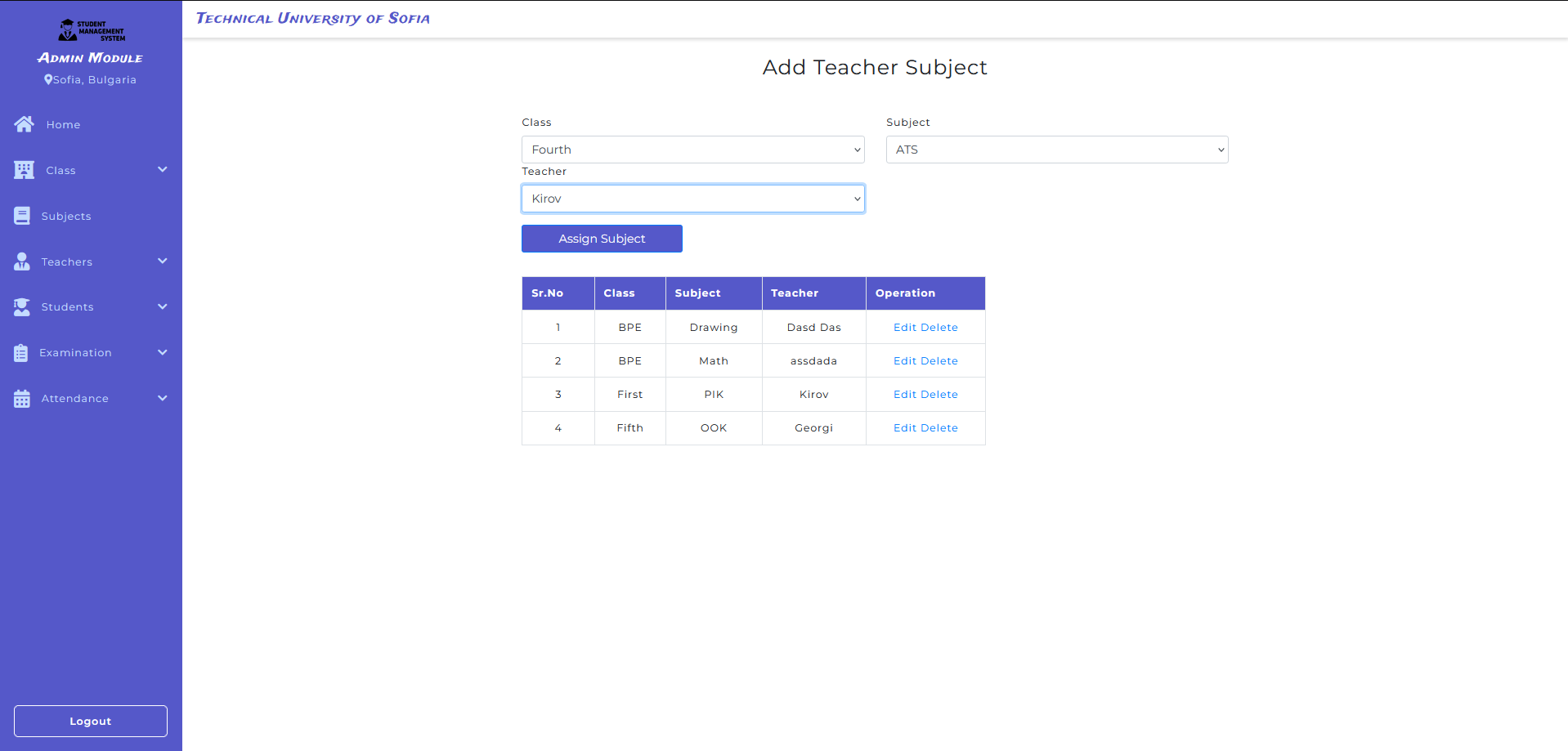




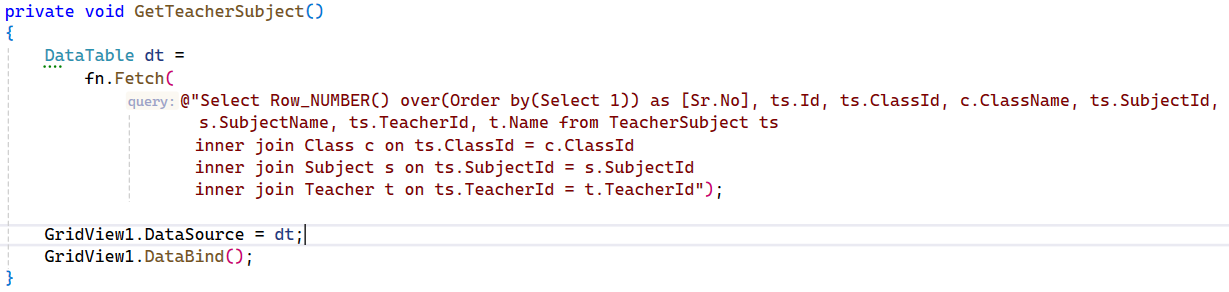
*Фиг.3.17. Изходен код на страница за добавяне на учител*

За да се извърши операцията добавяне на учител е необходимо да се напише неговото име, датата му на раждане, полът му, телефонният му номер, имейл, както и паролата, с която той ще влиза в платформата. Също така се изисква и да се запише неговият адрес към базата данни. За да се визуализира таблицата, която е на фигурата е нужно да се използва метод “GetTeachers()”, който метод извлича от базата данни информация чрез Fetch заявка, която информация съдържа името на учителя, телефонният му номер, имейл, както и адрес. В последната колона се съдържа операцията, която може да се извърши върху определения ред от таблицата. Операцията може да бъде редакция на реда или неговото изтриване.

## *добавяне на клас и учебен предмет към определен учител*



*Фиг.3.18. Страница за добавяне на учебен предмет към учител*



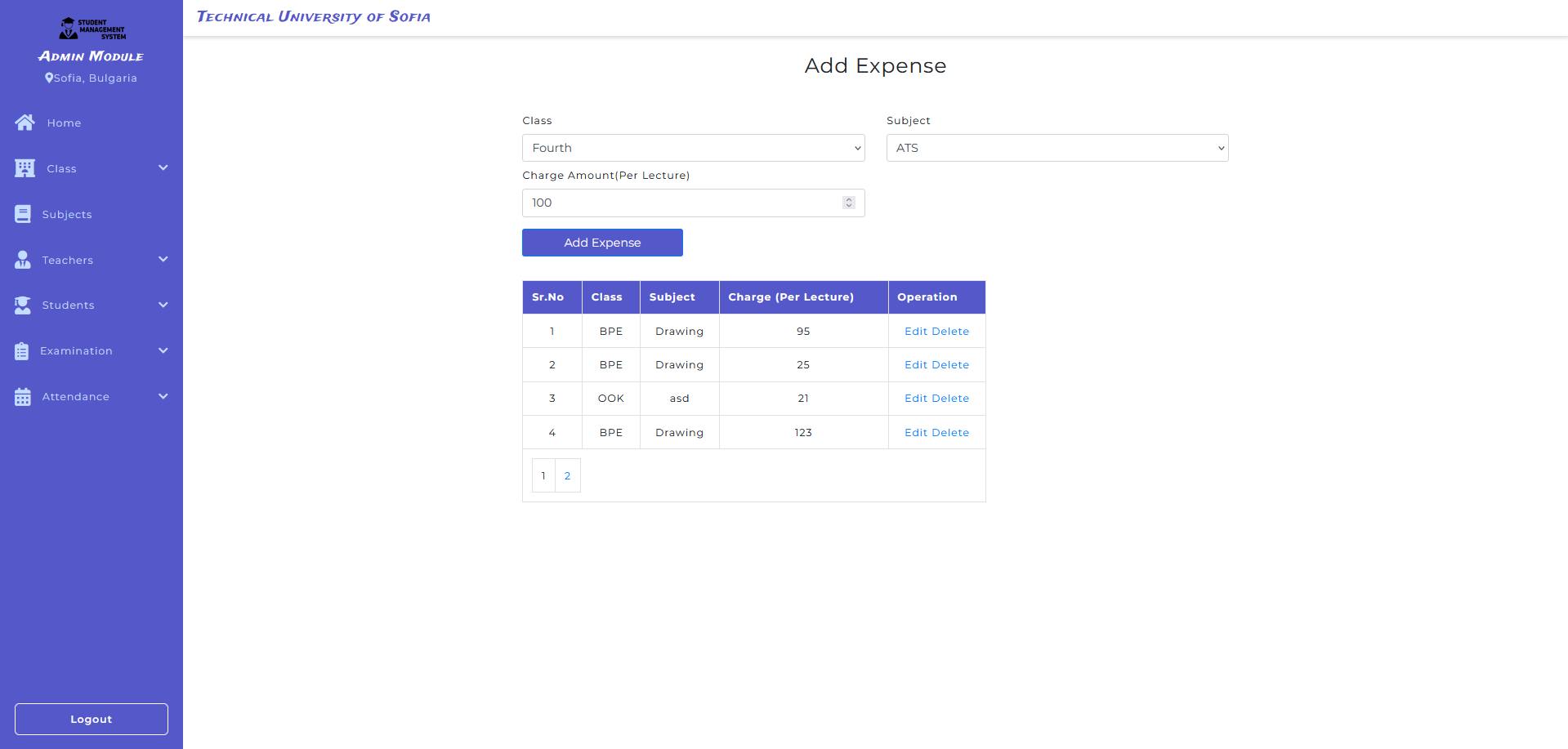
*Фиг.3.19. Изходен код на страница за добавяне на учебен предмет към учител*



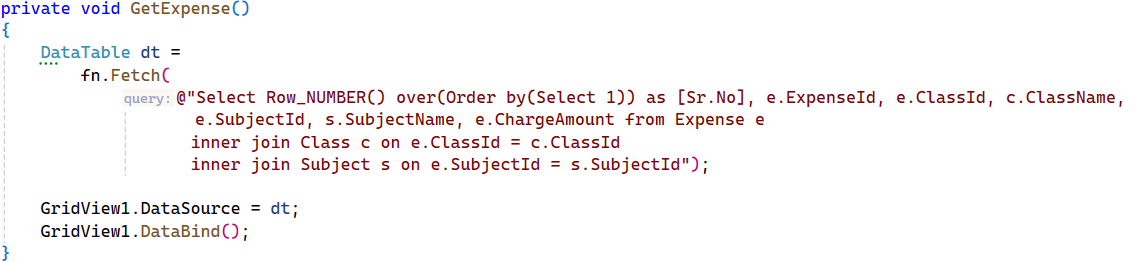
*Фиг.3.20. Изходен код на функция за добавяне на учител към предмет*

За да бъде извършена операцията добавяне на учител, който да води определен предмет се изисква да се вземе информацията за всички учители от базата данни, като се вземе името и идентификатора на учителя, след което да се вземе съответния учебен предмет отново от базата данни. Прави се заявка, която да вземе името и идентификатора на вече добавения предмет от базата данни. Също така трябва от базата да се достъпи и класа, към който ще се добавя предмета и учителя.

## *добавяне на такса към определена лекция на учител*



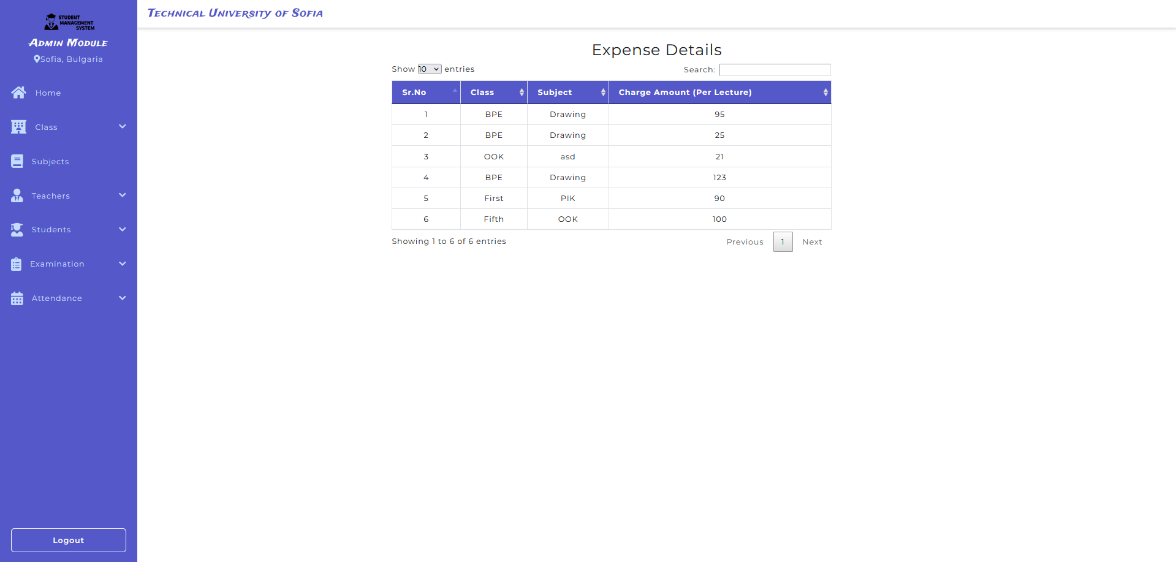
*Фиг.3.21. Страница за добавяне на такса към лекция на учител*

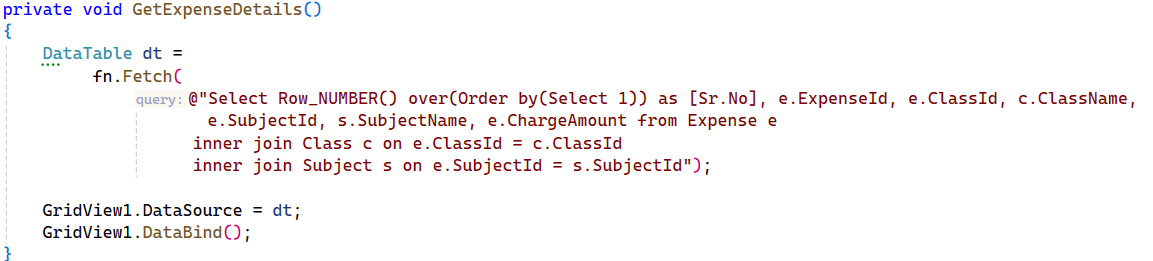


*Фиг.3.22. Изходен код на страница за добавяне на такса към лекция на учител*

За всеки предмет, който се добави в платформата се добавя и учител, който да бъде негов ръководител. След това, в платформата се налага да се добави и такса към определен предмет, която се води „такса към лекция“. Тази такса е по-малка като размер от таксата, която е годишната за класовете. Добавянето на тази такса се извършва като от базата данни се вземе, чрез Fetch заявка се извиква „такса-разход“, която да се добави към определен клас идентификатор. Прави се проверка (if-else), дали може да се добави тази такса към този клас. Ако вече има добавената такса, на екрана се показва съобщение-грешка, което да покаже, че операцията не може да бъде извършена. За визуализацията на таблицата, която е на фигурата се използват от базата данни следните данни: име на клас, учебен предмет, такса към определена лекция. В последната колона е полето, което дава възможност за операция редакция или изтриване.

## *следене на таксата към определена лекция на учител(детайли)*

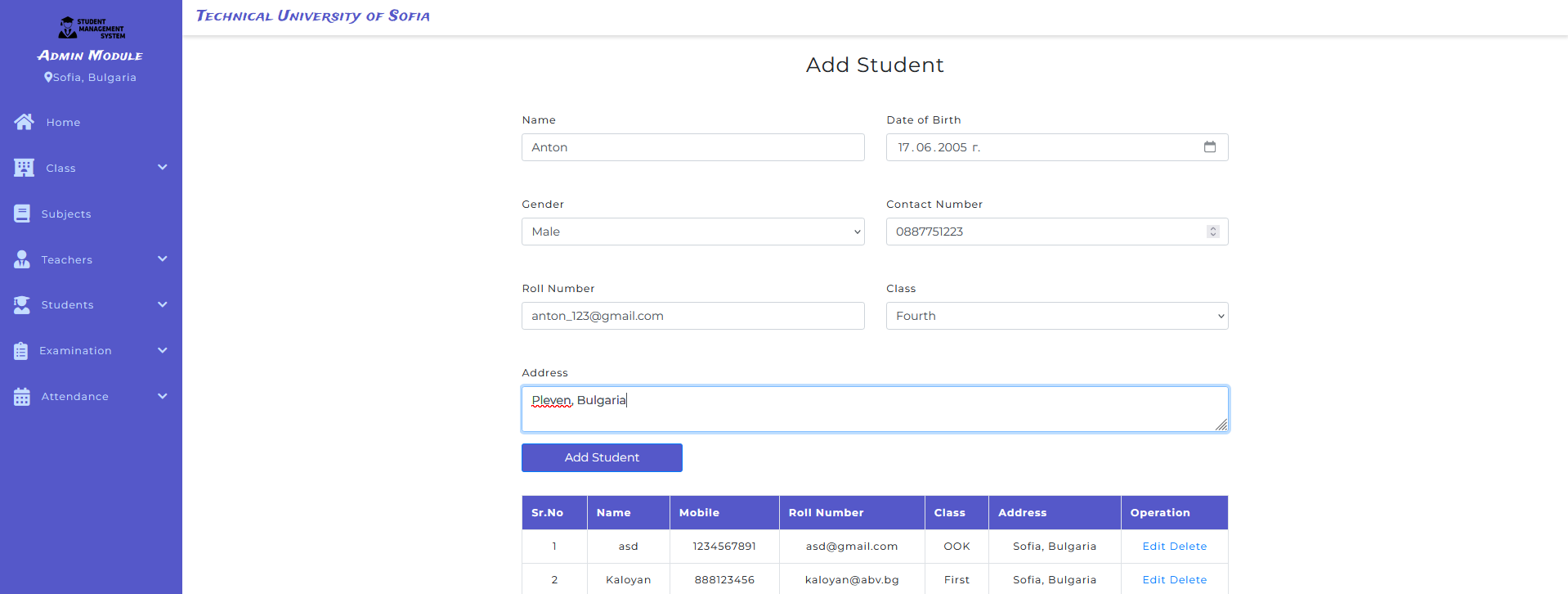
*Фиг.3.23. Страница за детайли към такса на лекция*



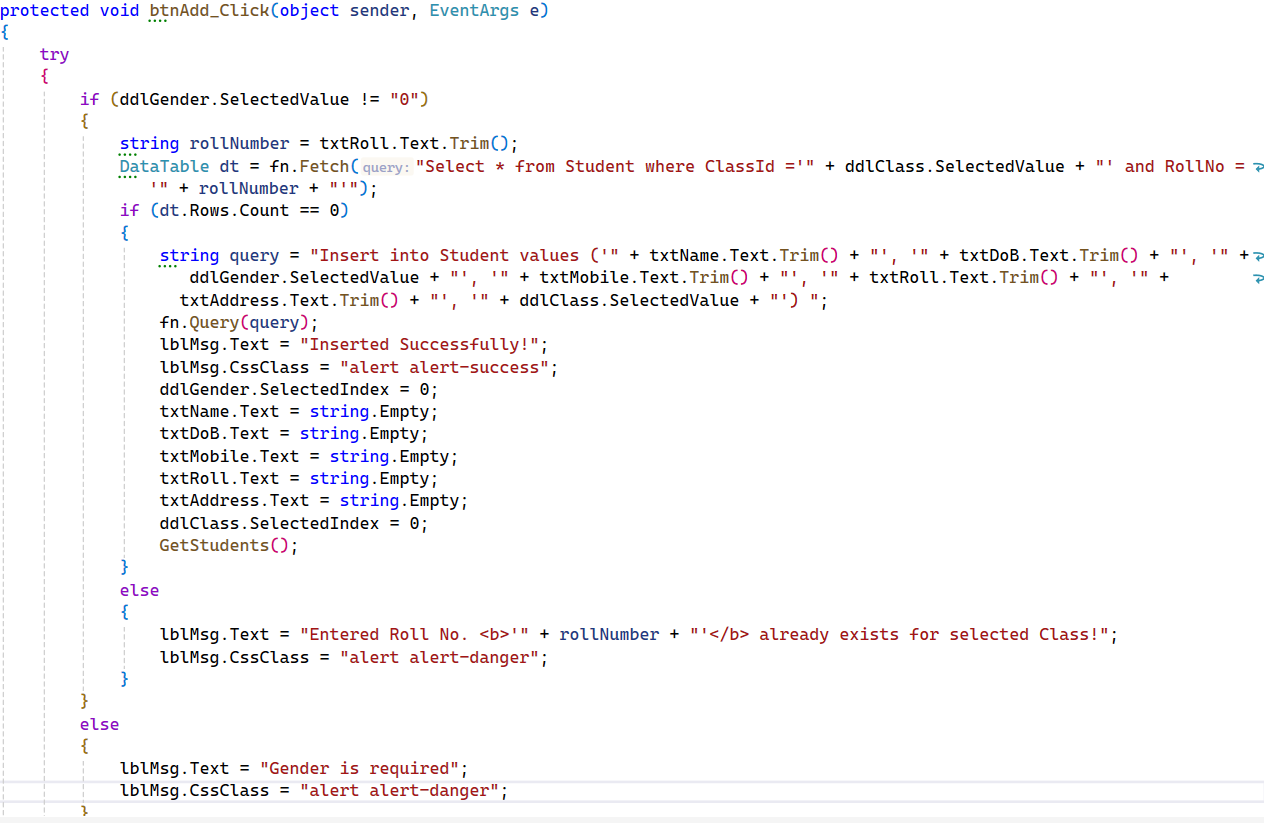
*Фиг.3.24. Изходен код на страница за детайли към такса на лекция*

“Expense Details” (в превод-детайли за такси към лекция) показва детайли към горе показаната функция за добавяне на такса. Това действие се извършва благодарение на извикването на метод GetExpenseDetails(), като от базата данни се вземат класът, към който е прикрепен учебния предмет. Към този предмет има създадена такса за лекция. Наблюдава се статистика, която е визуализирана под таблицата. Тази статистика показва, колко такси са показани в таблицата. От дясно на нея се визуализира разделянето на таблицата на различни таблици и също така възможността за сменянето на страниците на таблицата.

## *добавяне на ученик*



*Фиг.3.25. Страница за добавяне на ученик(студент)*

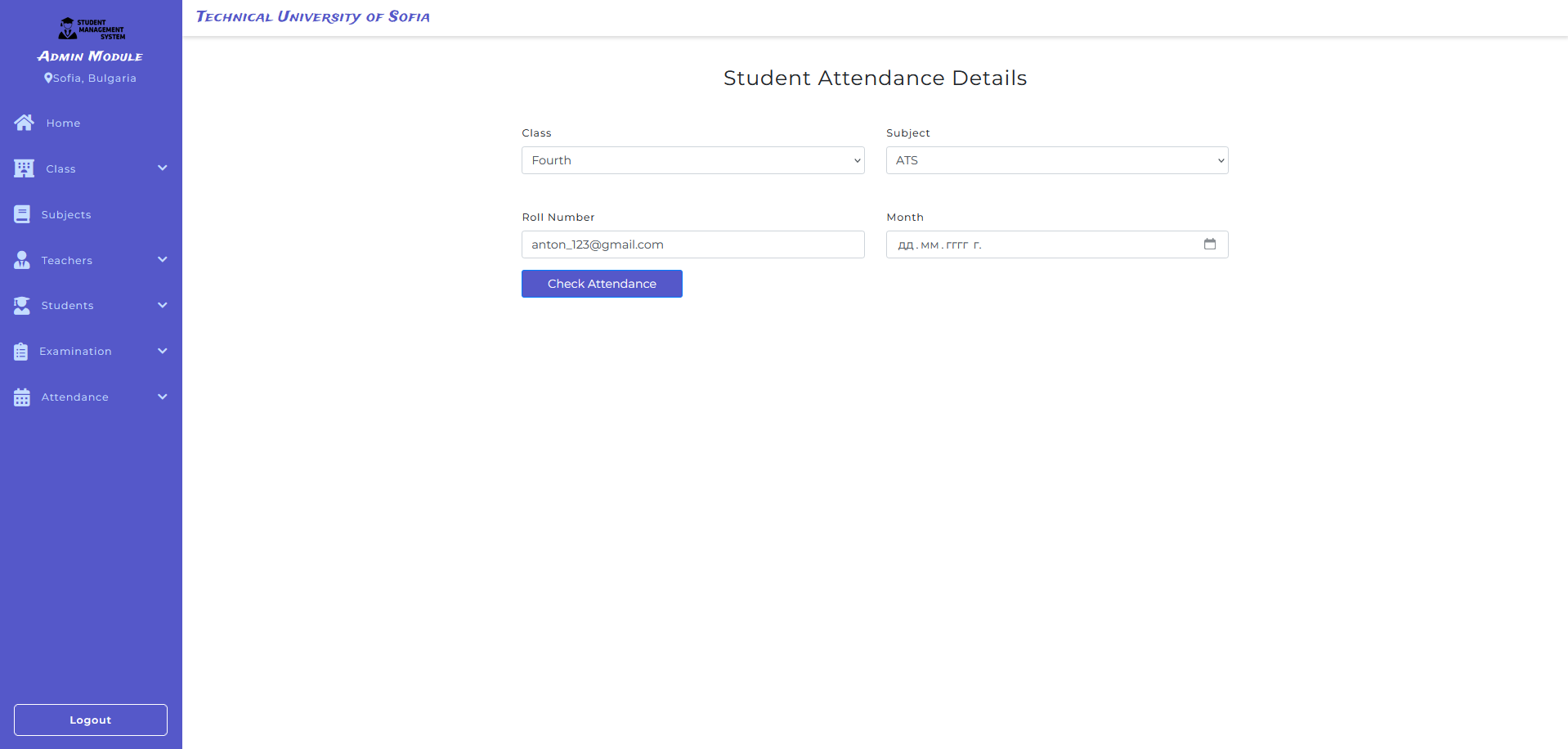




*Фиг.3.26. Изходен код на страница за добавяне на ученик(студент)*

Добавянето на ученик се извършва след като в приложението се добавят името на ученика, датата на раждане (има опция да се избере директно от календар датата на раждане), телефонен номер, имейл адрес, класът, към който да се добави студента и адресът. След натискането на бутон “Add Student” се извършва добавянето на ученикът, като за това се използва “Insert into” заявката към базата данни, след което студента се добавя към таблицата Student, като се запълват всичките изброени по-горе колони. Проверката за вече добавен студент се извършва по неговият имейл или Roll No. Ако в базата вече има добавен студент с такъв Roll No, това означава, че няма да се допусне повторното му добавяне и цялата операция ще бъде прекратена. Друга проверка, която се прави е за задължителното поле “Gender“ - пол. Ако няма избрана опция за полът на студента, няма да се допусне неговото добавяне, като на екрана ще се появи грешка. Това поле се води задължително. За визуализация на таблицата, която показва студентите, които са добавени в платформата, се използва метода GetStudents(). Чрез него се прави заявка към базата данни, за да се извлече нужната информация – име, телефонен номер, имейл (Roll No), класът, в който е записан студента, адресът му. В последната колона са операциите, които може да се извършват – редакция и триене на студент.

## *преглед на присъствие на ученик*



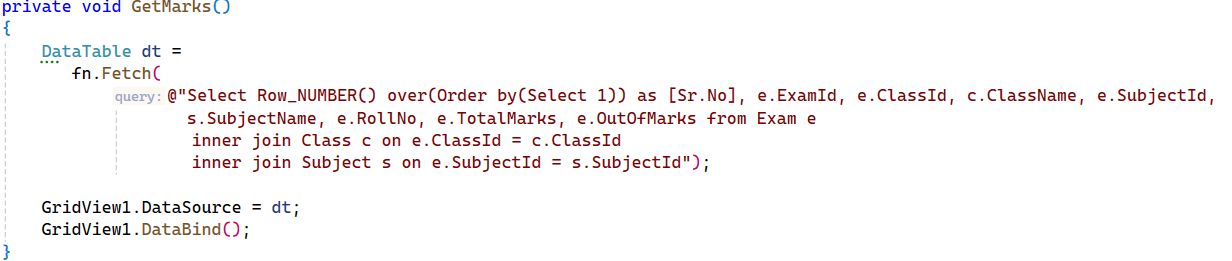
*Фиг.3.27. Страница за детайли на присъствието за даден студент*

Детайли за присъствие на ученик се вземат като администратора въведе класа, учебния предмет, Roll No (имейл), и месеца, в който иска да види присъствието. Чрез натискането на бутон “Check Attendance” се прави FETCH заявка към базата данни, чрез която се проверява дали за този месец има въведено присъствие на студента. Ако има такова ще се визуализира неговото присъствие под бутона. Ако обаче такова не съществува, под бутонът ще излезе съобщението „No record to display“. То показва, че в базата данни не е записано присъствие на студента през този месец и в зададеният предмет.

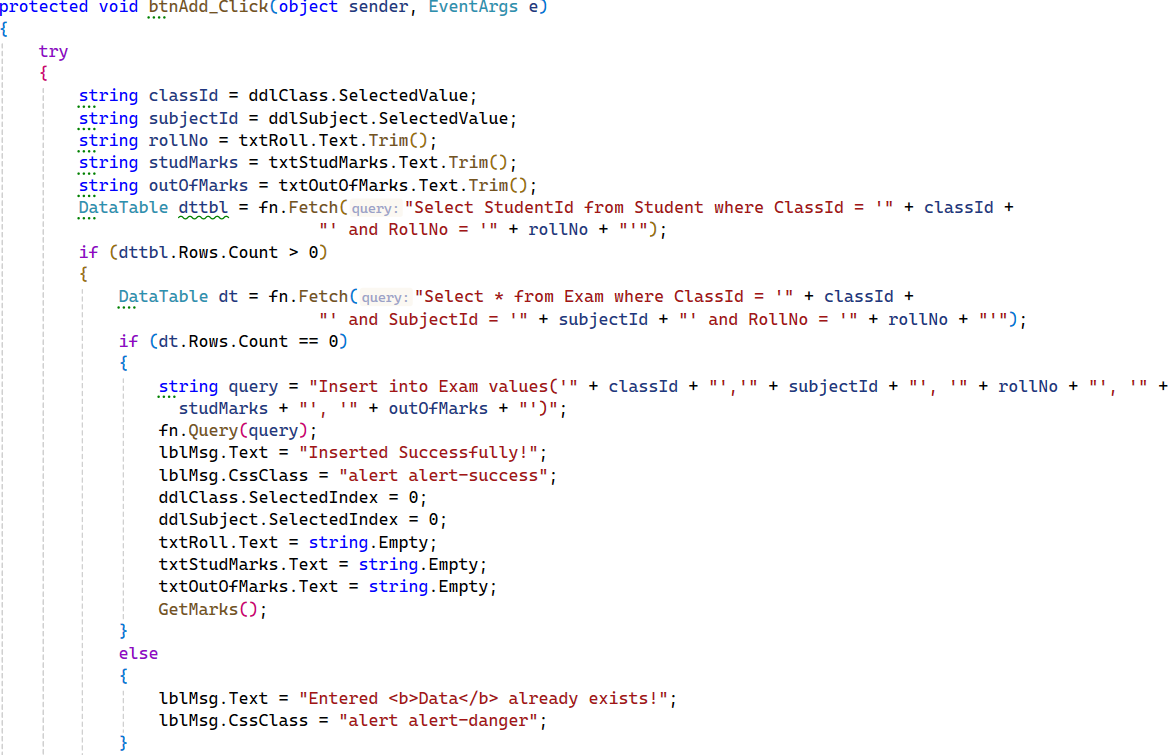
## *добавяне на оценка*



*Фиг.3.28. Страница за добавяне на оценки*



*Фиг.2.29. Изходен код на функция за извеждането на оценки от таблица*



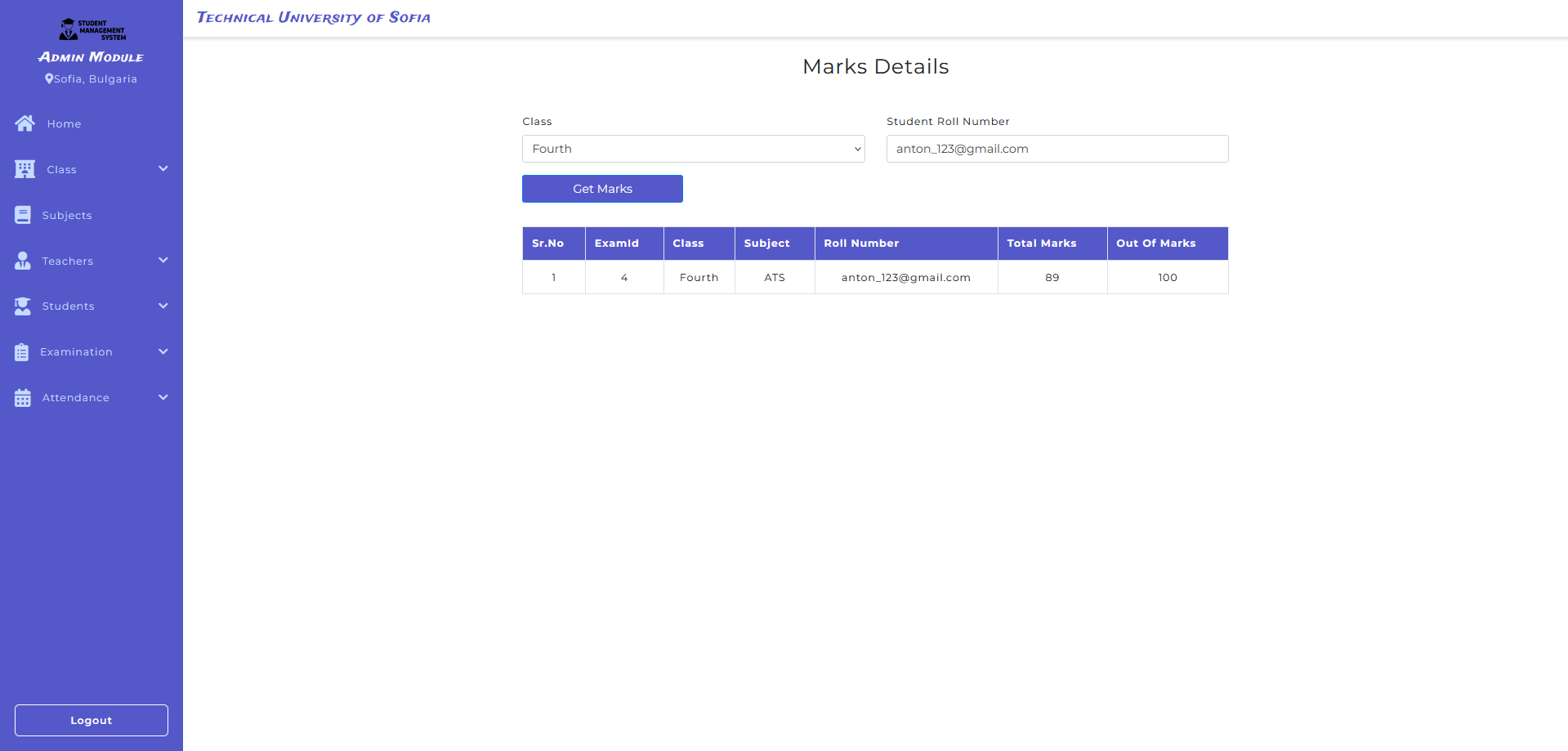


*Фиг.3.30. Изходен код на страница за добавяне на оценки*

За да бъде добавена оценка към определен предмет и студент от навигацията се избира опция “Examination” – изпитване. След като се селектира тази опция се натиска бутон “Add Marks”. Изгледът, който се показва на екрана показва, че за да се добави оценка е нужно да се селектира клас, към който клас да принадлежи определен учебен предмет. След това се изисква да се въведе Roll Number – имейл, който да покаже точно на кой студент ще се добави оценката. Последните две опции показват всъщност и самата оценка каква ще е – в първата част на двете опции се въвеждат точките, които е изкарал съответния студент, а във втората се добавят общият брой точки, които принадлежат към съответното изпитване.

За да се извърши успешно процесът по добавяне на оценка са необходими няколко метода – GetClass(), GetMarks() и btnAdd\_Click(). Първите два метода, чрез Fetch заявки към базата данни селектират нужните колони – клас, предмет, Student Roll Number – имейл на студент. Метода GetMarks() извежда двете колони за попълване на оценка. btnAdd\_Click() добавя в базата данни оценката, като записва информацията в колоните от таблицата Marks.

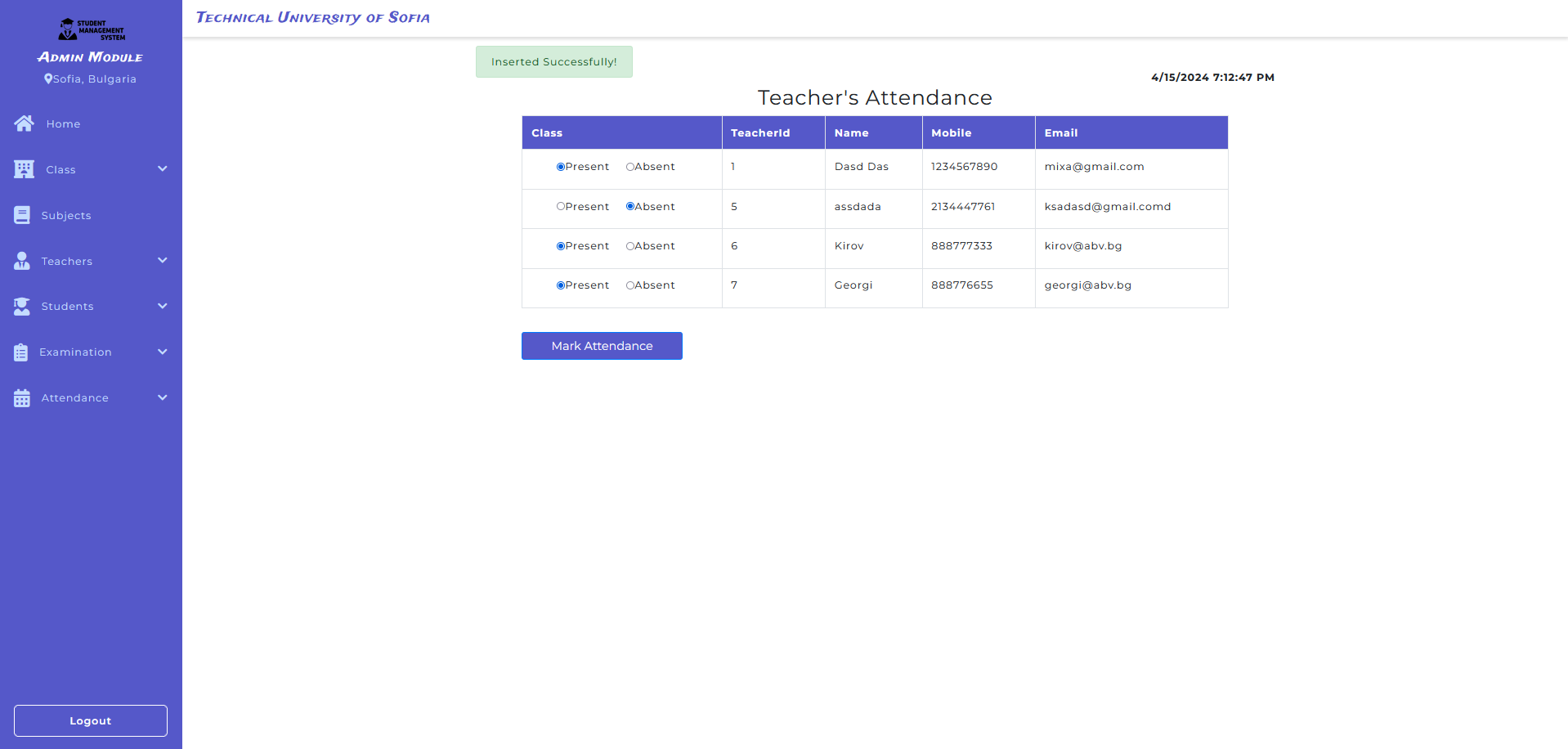
## *детайли за оценка*



*Фиг.3.31. Страница за преглед(детайли) за оценки*

Детайли за оценката на ученик се вземат като администратора въведе класа и Roll No (имейл) на студента. Чрез натискането на бутон “Get Marks” се прави FETCH заявка към базата данни, чрез която се проверяват оценките на студента и дали има въведени такива. Ако има ще се визуализират само неговите оценки под бутона. Ако обаче не съществуват, под бутонът ще излезе съобщението „No record to display“. То показва, че в базата данни не са записани оценки на студента, който е от зададения клас и със зададения Student Roll Number – студентски имейл.

## *добавяне на присъствие на учител към определена дата и време*



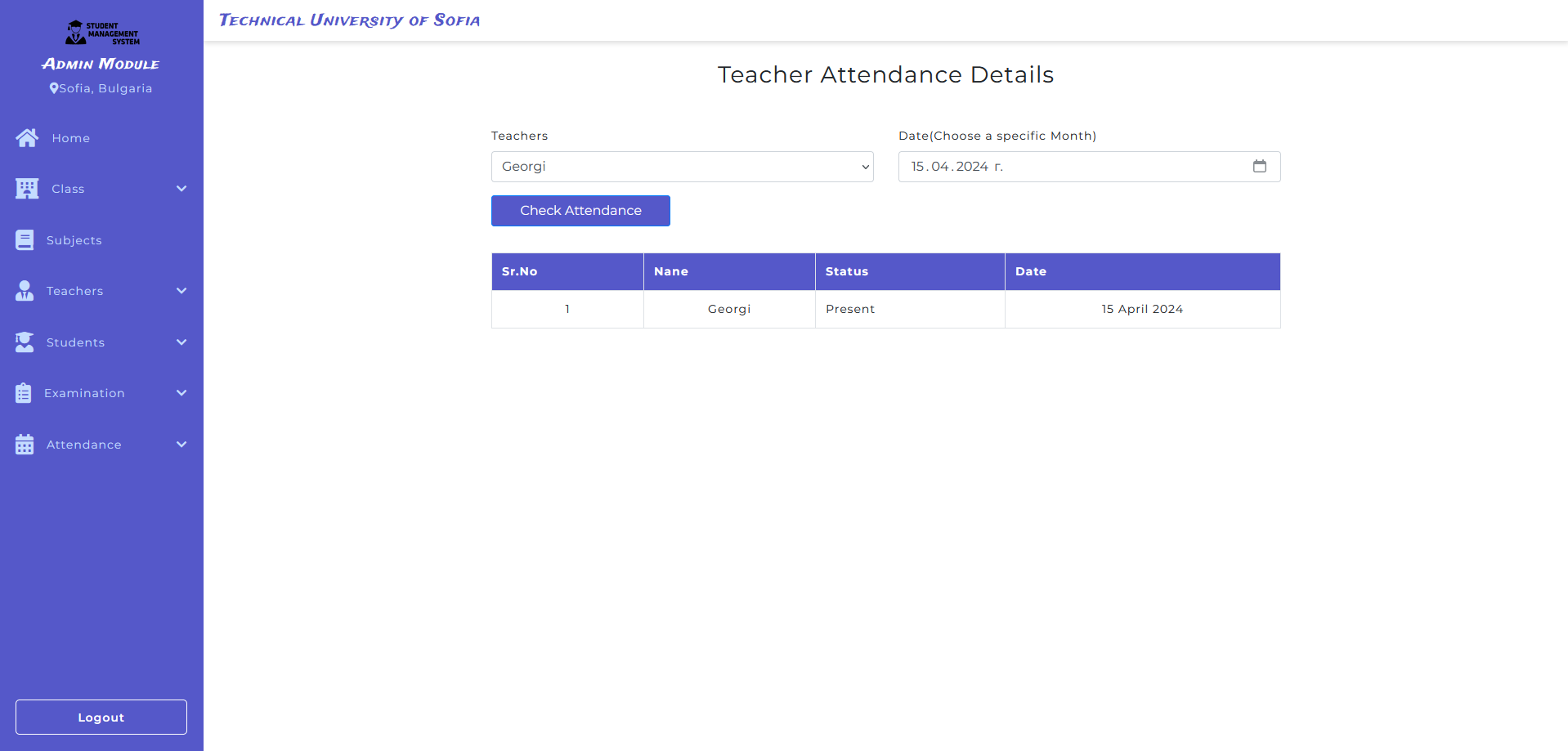
*Фиг.3.32. Страница за добавяне на присъствие на учител*



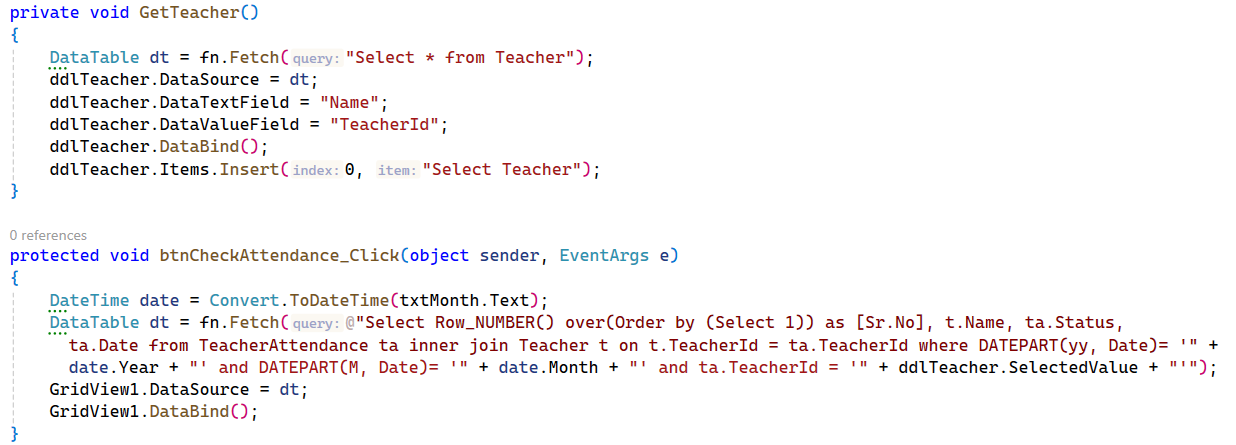
*Фиг.3.33. Изходен код на страница за добавяне на присъствие на учител*

Присъствието на учител се прави само и единствено от администратор. Само той има достъп до тази страница. За да се извърши това действие се прави FETCH заявка към базата данни, която да селектира Id на учителя, името му, мобилен номер, имейла му. Това се случва благодарение на метода – Attendance(). След като се случи това се създават два „радио бутона“ или “RadioButton”. Те служат за по-лесното отбелязване на присъствието на даден учител от администратор. След това се прави проверка кой от двата бутона е натиснат. Ако статуса на радио бутон 1 е “Checked“ (отбелязан), тогава променливата status приема стойност 1. Прави се и втора проверка, за да се изключи опцията и двата бутона да бъдат натиснати по едно и също време. Ако радио бутон 2 е “Checked“ (отбелязан), тогава променливата “status” става равна на 0. При натискането на бутон “Mark Attendance”, към базата данни се изпраща информация, която съдържа колоните с идентификатора на учителя, неговият статус и датата, когато той е бил на работа. За да бъде ориентиран администратора се добавя и таймер, който показва текущото време в горният десен ъгъл.

## *преглед на детайли за присъствие на учител*



*Фиг.3.34. Страница за детайли на присъствие на учител*

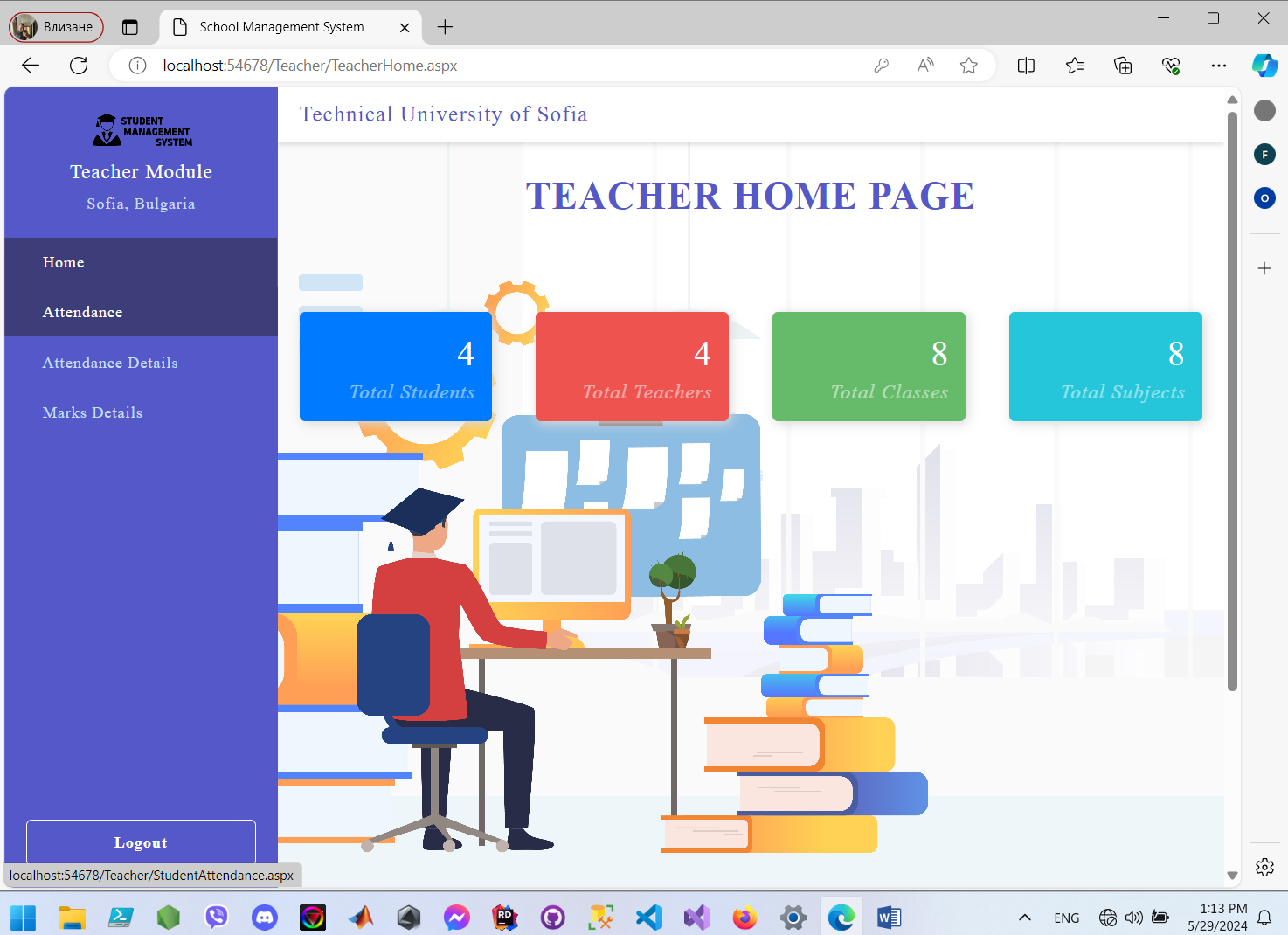


*Фиг.3.35. Изходен код на страница за детайли на присъствие на учител*

Детайли за присъствие на учителя се вземат като администратора въведе кой от всички учители иска да провери, както и датата(месеца), в който иска да види присъствието. Чрез натискането на бутон “Check Attendance” се прави FETCH заявка към базата данни, чрез която се проверява дали за този месец има въведено присъствие на учителя. Ако има такова ще се визуализира в колоната “Status” ще се появи, че той присъства или стойността ще е “Present”. Ако обаче такова не съществува, в таблицата, в колона “Status” ще се появи, че той отсъства или стойността ще е “Absent”. То показва, че в базата данни не е записано присъствие на студента през този месец и в зададеният предмет.

На фиг. 3.36 е показана началната страница на модул „Учител“, като е демонстрирана и платформената независимост както по отношение на използвания хардуер (лаптоп), така и по отношение на използваната операционна система (Microsoft Windows 11) и използвания браузър (Microsoft Edge).

## **Учител**



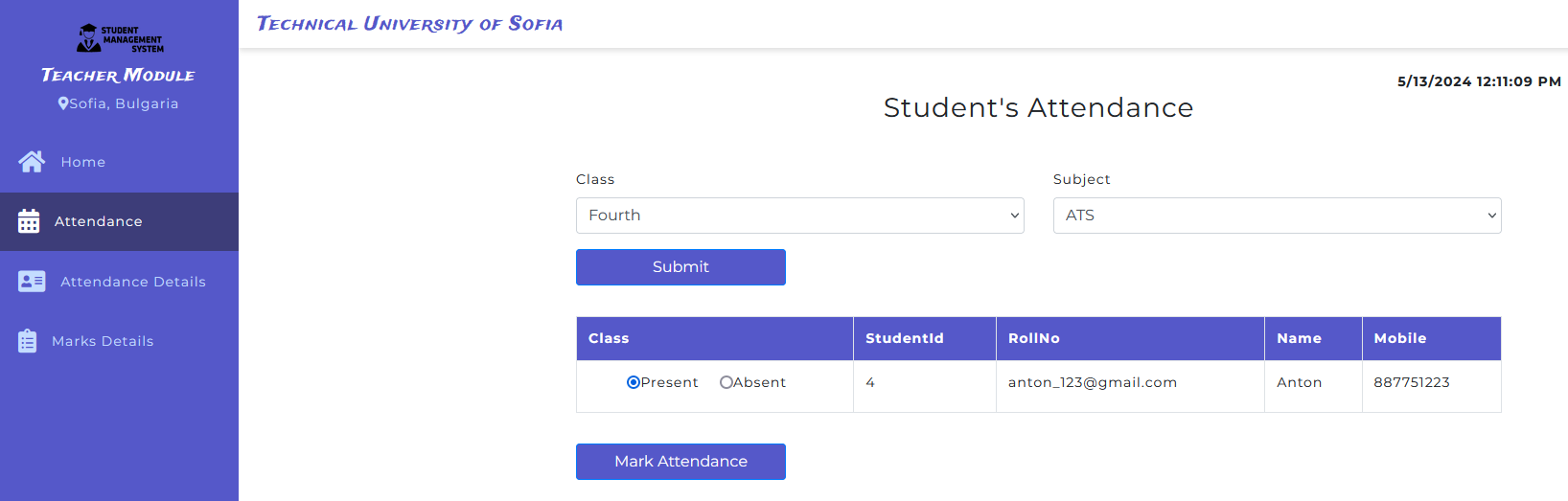
*Фиг.3.36. Начална страница на учител*



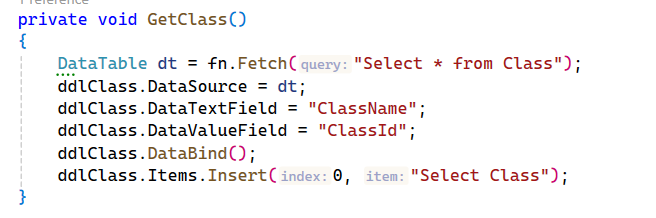
*Фиг.3.37. Изходен код на начална страница на учител*

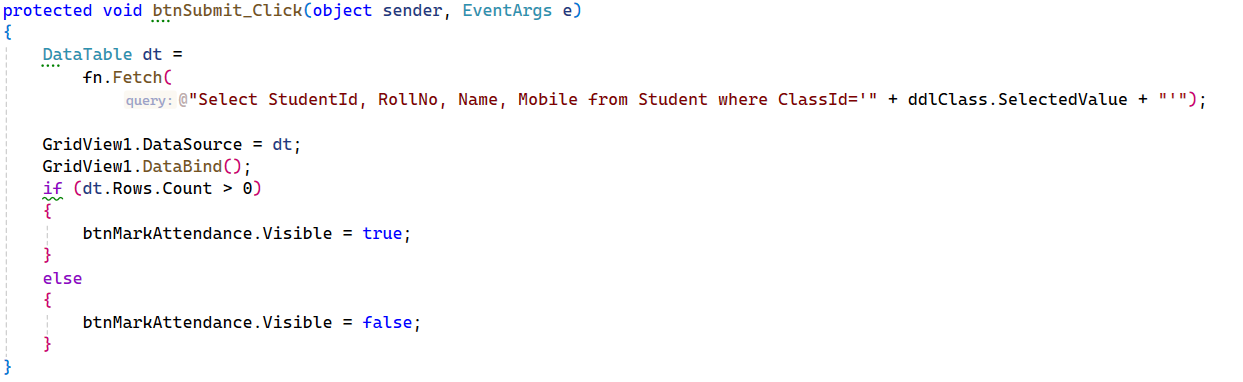
В случай, че потребител е с роля на учител, на екрана на потребителя ще се появи началната страница “Teacher”. Там могат да се видят всички права и функционалности на учителя, които са:

## *преглед и добавяне на присъствие на ученик*



*Фиг.3.38. Страница за добавяне на присъствие на студент*





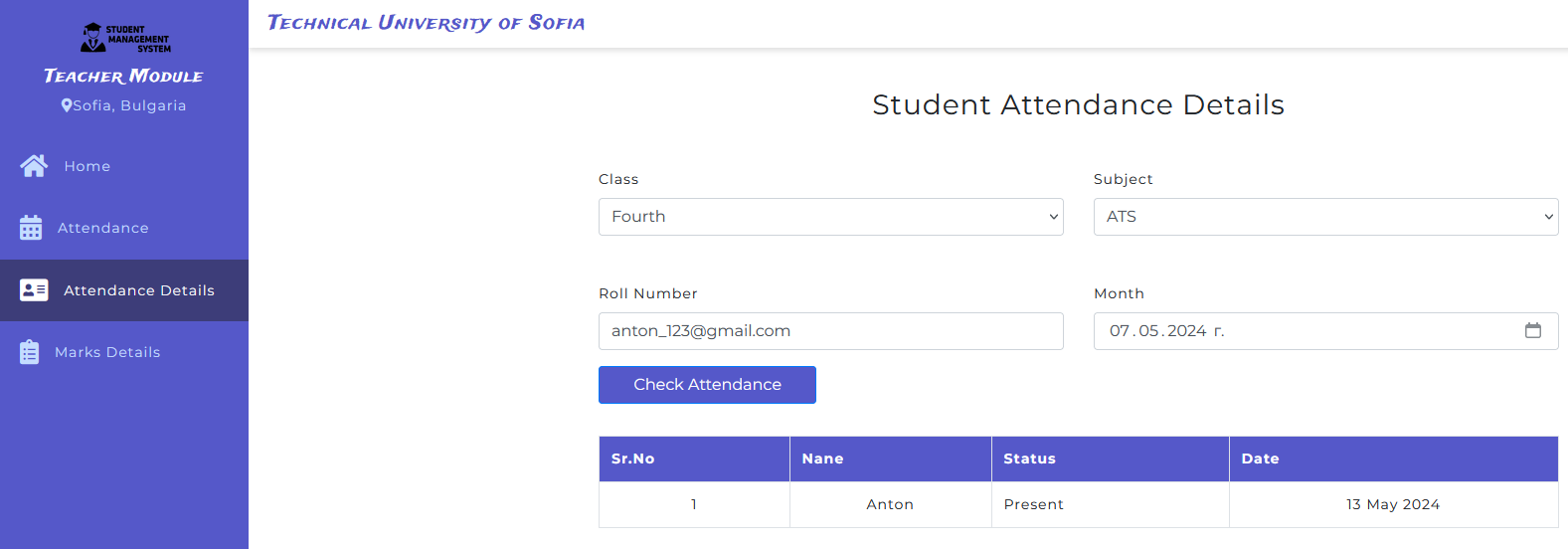
*Фиг.3.39. Функция за добавяне на присъствие на студент и получаването на класове от база данни*



*Фиг.3.40. Функция за добавяне на присъствие*

Тази функционалност се изпълнява като се използват няколко метода, сред които са GetClass(), btnSubmit\_Click(), btnMarkAttendance\_Click(). GetClass() метода използва FETCH заявка към базата данни, която взема цялата информация записана в таблицата за класовете – “Class”. След натискане на бутона “Mark Attendance” се изпълнява FETCH заявка, която избира всичките нужни параметри от таблицата със студенти – “Student”.

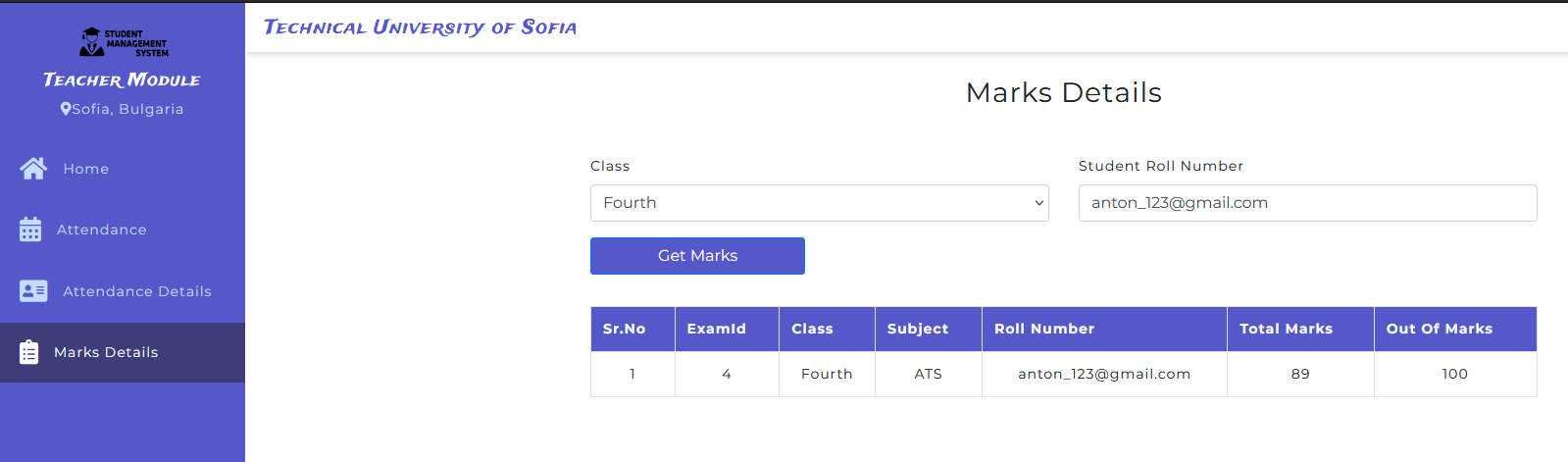
## *проверка на присъствие на студент*



*Фиг.3.41. Страница за детайли за присъствие на студент*

Детайли за присъствие на ученик се вземат като учителя въведе класа, учебния предмет, Roll No (имейл), и месеца, в който иска да види присъствието. Чрез натискането на бутон “Check Attendance” се прави FETCH заявка към базата данни, чрез която се проверява дали за този месец има въведено присъствие на студента. Ако има такова ще се визуализира неговото присъствие под бутона. Ако обаче такова не съществува, под бутонът ще излезе съобщението „No record to display“. То показва, че в базата данни не е записано присъствие на студента през този месец и в зададеният предмет.

## *проверка(детайли) на оценката на студент*



*Фиг.3.42. Страница за детайли на оценките*

Детайли за оценката на ученик се вземат като учителя въведе класа и Roll No (имейл) на студента. Чрез натискането на бутон “Get Marks” се прави FETCH заявка към базата данни, чрез която се проверяват оценките на студента и дали има въведени такива. Ако има ще се визуализират само неговите оценки под бутона. Ако обаче не съществуват, под бутонът ще излезе съобщението „No record to display“. То показва, че в базата данни не са записани оценки на студента, който е от зададения клас и със зададения Student Roll Number – студентски имейл.

# **Глава 4 – Анализ на получените резултати, приложимост и изводи**

В рамките на дипломната работа бяха поставени четири основни задачи, насочени към проучване и реализация на уеб сървър, базиран на GNU/Linux операционна система.

В първата задача бе проучено състоянието на операционните системи, базирани на GNU/Linux, и техните възможности за реализация на уеб сървър. Бяха разгледани различни дистрибуции като Ubuntu Server, Debian и други, които предлагат стабилност, сигурност и поддръжка на широк спектър от софтуерни решения за уеб хостинг. Специално внимание бе обърнато на уеб сървър - Apache.

Проучването показа, че GNU/Linux дистрибуциите са изключително подходящи за реализация на уеб сървър поради тяхната гъвкавост, мащабност и високо ниво на сигурност. Системите като Apache, Nginx и други уеб сървъри са напълно съвместими и оптимизирани за работа под GNU/Linux. Тези операционни системи могат да бъдат използвани в широк спектър от приложения, включително корпоративни уеб сайтове, електронна търговия, блогове и форуми. Те предлагат стабилна и сигурна платформа за хостинг на различни уеб базирани услуги. GNU/Linux дистрибуциите са доказано ефективни за реализиране на уеб сървър, осигурявайки необходимите инструменти и поддръжка за успешно изпълнение на различни уеб хостинг проекти.

Втората задача бе свързана с предлагането на конкретен подход за реализиране на уеб сървър, който осигурява хостинг на разработваната платформа чрез използване на GNU/Linux операционна система. Предложеният подход включва използването на Ubuntu Server като основна операционна система, Apache като уеб сървър, MSSQL за управление на бази данни и за сървърна логика – глава 3, фиг. 3.5. Този подход може да бъде приложен в различни сценарии, като се започне от малки лични проекти до големи корпоративни платформи. Комбинацията от тези технологии осигурява висока производителност, сигурност и лесна поддръжка. Apache, като дългогодишен лидер в уеб сървърите, допълнително засилва стабилността и надеждността на предложеното решение. Предложеният подход се доказва като ефективен и надежден за реализиране на уеб сървър, базиран на GNU/Linux, предоставяйки всички необходими компоненти за успешното функциониране на уеб платформа.

Третата задача включваше реализацията на сървър, следвайки предложения подход. Сървърът беше успешно реализиран, като всички компоненти, включително Apache, бяха инсталирани и конфигурирани. Беше осигурен достъп до уеб платформата от различни устройства и операционни системи, което демонстрира платформената независимост – глава 3, фиг. 3.8, фиг. 3.36. Реализираният сървър може да бъде използван за хостинг на разнообразни уеб приложения (фиг. 3.3), предоставяйки на потребителите гъвкав и достъпен начин за взаимодействие с платформата от различни устройства. Реализацията на сървъра по предложения подход потвърждава неговата ефективност и способност да осигури платформена независимост, което е критичен фактор за съвременните уеб приложения.

Последната задача бе насочена към осигуряване на контрол на достъпа до разработваната електронна платформа и защита на информацията. Бяха внедрени механизми за автентикация и авторизация, използвайки технологии като двуфакторна автентикация – глава 3, фиг. 3.6. Допълнително бяха конфигурирани защитни стени и инструменти за мониторинг на сигурността – глава 3, фиг. 3.4. Тези мерки могат да бъдат прилагани в различни уеб платформи, където сигурността на данните и контролът на достъпа са от съществено значение. Те осигуряват защита срещу неоторизиран достъп и потенциални атаки. Внедрените механизми за сигурност и контрол на достъпа доказват, че е възможно да се осигури високо ниво на защита на информацията, което е от критично значение за всяка уеб платформа.

Изпълнението на поставените задачи в рамките на дипломната работа доведе до успешното реализиране на уеб сървър, базиран на GNU/Linux операционна система. Получените резултати демонстрират висока приложимост и ефективност на използваните технологии и подходи. Изводите подкрепят значимостта на GNU/Linux като стабилна и сигурна платформа за уеб хостинг, като същевременно подчертават важността на добре планираните механизми за контрол на достъпа и защита на информацията.

# **Използвана литература**

1. GeeksforGeeks. Какво е данни? [онлайн]. Налично на: <https://www.geeksforgeeks.org/what-is-data/> [Достъп: 24.01.2024г.].
2. Investopedia. Система за подкрепа на вземане на решения [онлайн]. Налично на: <https://www.investopedia.com/terms/d/decision-support-system.asp> [Достъп: 24.01.2024г.].
3. LiveAbout. Системи за управление на процесите (PCS) [онлайн]. Налично на: <https://www.liveabout.com/process-control-systems-pcs-2221184> [Достъп: 05.02.2024г.].
4. Camu Digital Campus. Система за управление на университет [онлайн]. Налично на: <https://camudigitalcampus.com/university-management-system> [Достъп: 10.02.2024г.].
5. GeeksforGeeks. Въведение в програмни езици [онлайн]. Налично на: <https://www.geeksforgeeks.org/introduction-to-programming-languages/> [Достъп: 15.02.2024г.].
6. Microsoft. Миграции и внедряване с Entity Framework в ASP.NET MVC приложение [онлайн]. Налично на: <https://learn.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/migrations-and-deployment-with-the-entity-framework-in-an-asp-net-mvc-application> [Достъп: 15.02.2024г.].
7. Microsoft. Основни и външни ключове [онлайн]. Налично на: <https://learn.microsoft.com/en-us/sql/relational-databases/tables/primary-and-foreign-key-constraints?view=sql-server-ver16> [Достъп: 15.03.2024г.].
8. GeeksforGeeks. Въведение в Visual Studio [онлайн]. Налично на: <https://www.geeksforgeeks.org/introduction-to-visual-studio/> [Достъп: 15.02.2024г.].
9. Microsoft. SQL Server Management Studio (SSMS) [онлайн]. Налично на: <https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16> [Достъп: 19.02.2024г.].
10. Microsoft. Запитвания към многомерни данни с MDX [онлайн]. Налично на: <https://learn.microsoft.com/en-us/analysis-services/multidimensional-models/mdx/querying-multidimensional-data-with-mdx?view=asallproducts-allversions> [Достъп: 22.02.2024г.].
11. Microsoft. Data Mining Extensions (DMX) - референция [онлайн]. Налично на: <https://learn.microsoft.com/en-us/sql/dmx/data-mining-extensions-dmx-reference?view=sql-server-ver16> [Достъп: 25.02.2024г.].
12. Microsoft. Портване, мигриране и обновяване на проекти във Visual Studio [онлайн]. Налично на: <https://learn.microsoft.com/en-us/visualstudio/releases/2022/port-migrate-and-upgrade-visual-studio-projects> [Достъп: 03.03.2024г.].
13. Negus, C. Linux Bible. 2020.
14. Gonzalez, A. Linux Server Cookbook. 2023.
15. Harkness, D.J. Apache Essentials: Install, Configure, Maintain. 2022.