****

**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА**

**Факултет по Изчислителна Техника и Автоматизация**

**КАТЕДРА „Софтуерни и интернет технологии“**

**„Разработка на програмни модули за съставяне на Учебен разпис –**

**Uni-Scheduler“**

**Дипломна работа за придобиване на образователно-квалификационна степен „бакалавър“**

**Дипломант: Научен ръководител:**

Николай Цветанов Георгиев / х.доц. М. Митев /

Факултетен № 61562162

Специалност: Софтуерни и Интернет Технологии

**Вapнa**

**2019**

[**„Разработка на програмни модули за съставяне на Учебен разпис –** 1](#_Toc18282072)

[**Uni-Scheduler“** 1](#_Toc18282073)

[Глава 1. Увод 5](#_Toc18282074)

[Глава 2. Обзор и анализ на алгоритмите и програмите за разработка програмни модули за съставяне на учебен разпис 6](#_Toc18282075)

[2.1 Анализ на подобен тип системи 6](#_Toc18282076)

[2.1.1 Система за изготвяне на учебен график “StudyGizmo” 6](#_Toc18282077)

[2.1.2 Система за изготвяне на учебен график “Schedule My Teachers” 8](#_Toc18282078)

[2.1.3 Система за изготвяне на учебен график “ Schedule Builder Online” 9](#_Toc18282079)

[2.2 Цели 10](#_Toc18282080)

[2.2 Таргет група от хора, към които е фокусирано приложението. 11](#_Toc18282081)

[2.3 Общи изисквания на потребителя 13](#_Toc18282082)

[2.3 Общи изисквания на потребителя 13](#_Toc18282083)

[2.4 Use-case 13](#_Toc18282084)

[2.4.1 Преподавател 13](#_Toc18282085)

[2.4.3 Администратор (Разпореждащия учебни занятия) 14](#_Toc18282086)

[2.5 Прототип на Uni-scheduler 15](#_Toc18282087)

[Глава 3: Технологична схема за разработка на учебни разписи 18](#_Toc18282088)

[3.1 Описание на данните 18](#_Toc18282089)

[3.1.1 Представяне на роли 18](#_Toc18282090)

[3.1.4 Представяне на бизнес правила 19](#_Toc18282091)

[3.1.5 Определяне на обектите с техните атрибути и релациите 21](#_Toc18282092)

[3.1.6 Релационна диаграма 28](#_Toc18282093)

[3.2 Използвани технологии 29](#_Toc18282094)

[3.2.1 Уеб приложение 29](#_Toc18282095)

[3.2.1.1 Клиент-сървър 29](#_Toc18282096)

[3.2.1.2 Клиент 30](#_Toc18282097)

[3.2.1.3 Сървър 30](#_Toc18282098)

[3.2.2 Структура на уеб приложението 31](#_Toc18282099)

[3.2.2.2 Обектно-релационно картографиране(ORM) 33](#_Toc18282100)

[3.2.2.3 Технологична рамка(framework) 34](#_Toc18282101)

[3.2.2.4 Виртуални машини: 35](#_Toc18282102)

[3.2.2.4 Vagrant: 36](#_Toc18282103)

[3.2.2.5 Ansible: 36](#_Toc18282104)

[3.2.2.6 YAML 36](#_Toc18282105)

[3.2.2.7 Secure Shell (SSH): 37](#_Toc18282106)

[3.2.2.8 Система за контрол на версиите (Git): 37](#_Toc18282107)

[3.2.2.9 „S.O.L.I.D” принципи: 38](#_Toc18282108)

[3.2.2.10 PHP (Hypertext Preprocessor) 38](#_Toc18282109)

[3.2.2.11 „Symfony“ (Технологична рамка) 39](#_Toc18282110)

[3.2.2.12 MySQL 40](#_Toc18282111)

[3.2.2.13 HTML (HyperText Markup Language) 41](#_Toc18282112)

[3.2.2.14 CSS (Cascading style sheets) 42](#_Toc18282113)

[3.2.2.15 Bootstrap 44](#_Toc18282114)

[3.2.2.15 XML(Extensible Markup Language) 44](#_Toc18282115)

[3.2.2.16 Twig 44](#_Toc18282116)

[3.2.2.17 Webpack 45](#_Toc18282117)

[3.2.2.18 NGINX 45](#_Toc18282118)

[3.2.2.19 Symfony Bundle(пакет) 46](#_Toc18282119)

[3.2.2.19 Hyper-V 46](#_Toc18282120)

[Глава 4. Разработка на програмните модули за разработка на учебен разпис 47](#_Toc18282121)

[4.1 Конфигурация на сървърната машина 47](#_Toc18282122)

[4.1.1 Технологии нужни на разработчика за инициализация на виртуалната машина 47](#_Toc18282123)

[4.1.2 Инициализация на виртуалната машината 48](#_Toc18282124)

[4.1.3 Устанояване на връзка между сървъра и разработчика 49](#_Toc18282125)

[4.1.4 Набавяне на нужните за разработка на приложението технологии 49](#_Toc18282126)

[4.1.5 Конфигурация на технологиите 50](#_Toc18282127)

[4.2 Конфигурация на средатата за програмиране 50](#_Toc18282128)

[4.2.1 Избор и настройки на текстов редактор 50](#_Toc18282129)

[4.2.2 Интеграция на Git 51](#_Toc18282130)

[4.3 Инициализация на Symfony проект 51](#_Toc18282131)

[4.3.1 Архитектура на приложението 52](#_Toc18282132)

[4.3.2 Конфигурация на проект 53](#_Toc18282133)

[4.4 Реализация на приложението 54](#_Toc18282134)

[4.4.1 Израждане на моделните обекти 54](#_Toc18282135)

[4.4.2 Изграждане на базата данни 60](#_Toc18282136)

[4.4.3 Изграждане на клиент-контролери 67](#_Toc18282137)

[4.4.4 Изграждане на админ-контролери 69](#_Toc18282138)

[4.4.5 Изграждане обекти на хранилища 69](#_Toc18282139)

[4.4.6 Създаване на изгледи 69](#_Toc18282140)

[4.4.7 Налагане на правила/валидации 69](#_Toc18282141)

# Глава 1. Увод

В наши дни хората почти не могат да си представят живота си без невероятните технологии и чудните устройства, които само преди десетилетия се срещаха единствено на страниците на научнофантастичните романи. Компютри, смартфони, таблети, лаптопи – всеки от нас, ежедневно използва тези „плодове на научно-техническия прогрес". Всички тези устройства са направени с цел улеснение на човешките дейности. Както ни дават възможността да извършваме сложни и времеемки задачи, свързани със изчисление или извличане на информация, така и осигуравя до голяма степен достоверност или автентичност на извършената с помоща на тези технологии, задача. Друга услуга която ни придоставят е способността за експресното разгласяването на важна информация.

Един страхотен пример за нуждата от създаване на технология, която предостовя иммено бързина, автентичност и достъп на информацията е тази която дава възможност за разпореждане на задълженията на един преподавател във висшите учебни заведения. В някои от тях, тази дейност се извършва от хора, които нямат в наличие, подобна технология и единствения начин за справяне с такава задача е прибягването към старите методи за състаяване, разгласяне и осигуряването на автентичност, което е именно изписването им на хартиен носител. Това само по себе си води до допускането на грешки, което пък води до още по-голяма загуба на време, поради факта, че поправянето на една грешка в един график може да доведе до много други проблеми, също така предоставянето на информацията е отново чрез разгласяване на многобройни листи от хартия, които в последвствие ще трябва да се подновят при разрешаване на проблема. Прост пример за това са колизиите в графиците на преподавателите между упражнения които се водят в една и съща стая по едно и също време. Избягване на такъв тип проблеми възникнали поради човешка грешка при съставянето на графика за един преподавател не само влияе на самите преподаватели но и на студентите които губят ценно време от занятието, което би трябвало в този момент да имат. За осъществяването на тази задачата, чиято цел е да състави учебните разписи на всички преподаватели във висшето учебно заведение, за кратък период от време, са нужни доста хора. При наличието на такава система, броят на тези хора ще бъде сведен до един човек.

# Глава 2. Обзор и анализ на алгоритмите и програмите за разработка програмни модули за съставяне на учебен разпис

## 2.1 Анализ на подобен тип системи

### C:\Users\nikol\Desktop\screencapture-freecollegeschedulemaker-2019-08-27-21_02_02.png2.1.1 Система за изготвяне на учебен график “StudyGizmo”

Фиг.1 – Начална странца на уебсайта за система създаване на графици “StudyGizmo“.

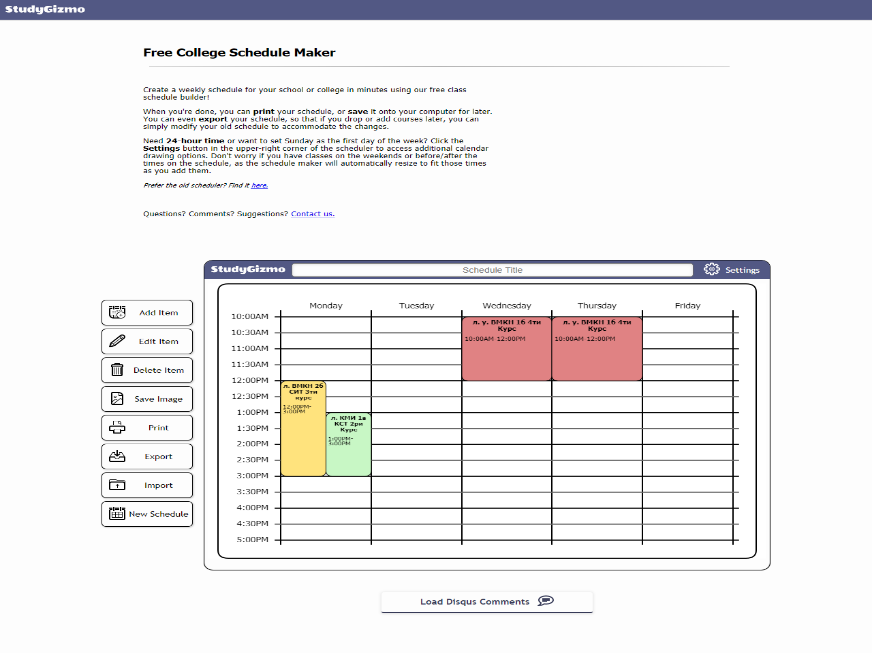
Една от доста срещаните системи, разработени с цел, съставянето на учебни разписание е иммено уеб платформата „StudyGizmo“. Тази система е доста често използвана, не само от преподаватели, но и от студенти. Достъпна е от всякакви съвременни устройства които имат достъп до интернет мрежа, като, смартфони, лаптопи, компютри и таблети. Простия дизайн на системата предоставя възможостта на потребителите да съставят без да изпитват затруднения, графици, които са лесно разчитими.

**Плюсове**:

* Опростен и интуитивен дизайн за бързо и лесно ползване.
* Безплатна
* Достъпна
* Даваща възможност за създаване, редактиране или изтриване на занятия или събития
* Позволява запазване на изготвения график в формат на снимка
* Позволява въвеждането на вече съществуващи графици

**Минуси:**

* Системата не предоставя възможността от изготвянето на комплексни графици.
* Системата е несъвместима с повечето мобилни устройства
* Липсват алгоритми, осигуряващи автентичност на графиците

Фиг. 2 – Примерен график, изготвен от преподавател

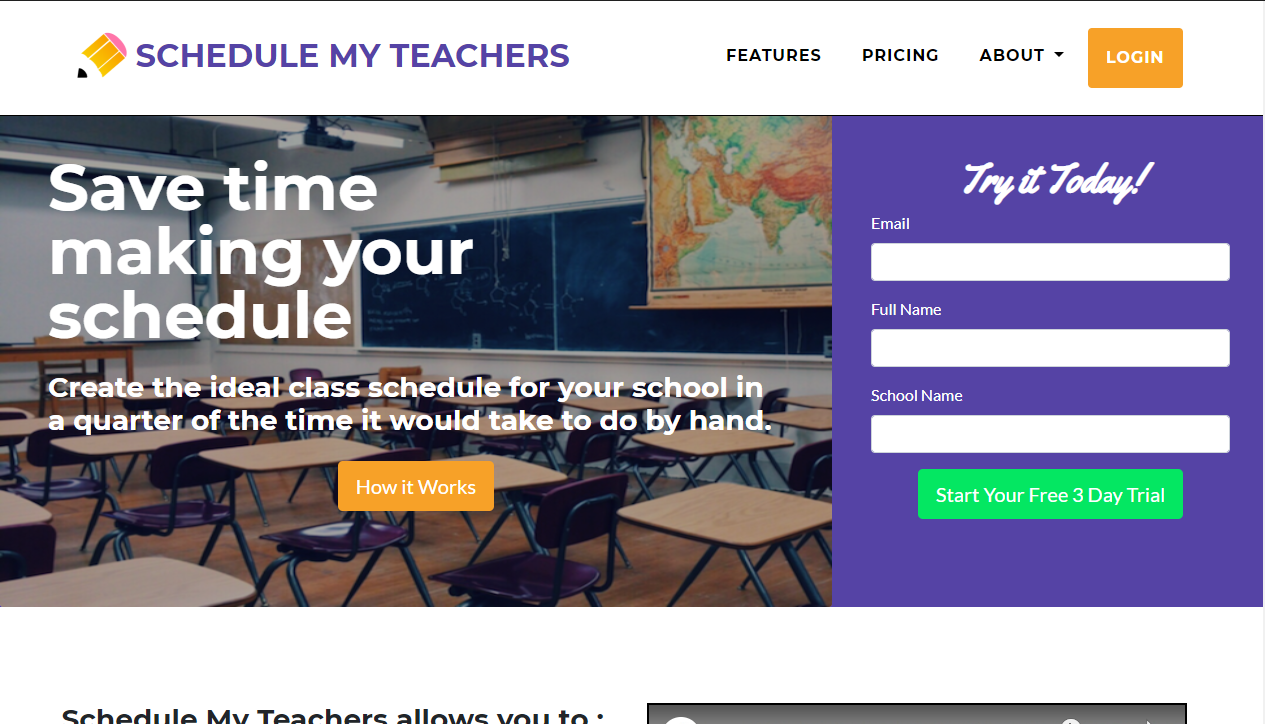
### 2.1.2 Система за изготвяне на учебен график “Schedule My Teachers”

Друга доста разпространена сред учебните заведения, система, която за разликата от “StudyGizmo”. Целта на приложението е да съставя графици на занятията за преподаватели и ученици. Дава възможност за

**Плюсове:**

* Съставяне на множество от графици за учители в дадено учебно заведение
* Автоматично генериране на график
* Изготвяне на график по свой вкус, в дни удобни за теб.
* Равномерно разпределяне на занятията през седмицата.
* Опростен и интуитивен дизайн за бързо и лесно ползване.

**Минуси:**

* Системата не е достъпна за външни лица.
* Услугата е платена
* Не е предназначена за висшите училищни заведения.
* Не предоставя възможност за смяна на езика на работа

Фиг. 3 – Начална страница на “Schedule My Teachers”

### 2.1.3 Система за изготвяне на учебен график “ Schedule Builder Online”

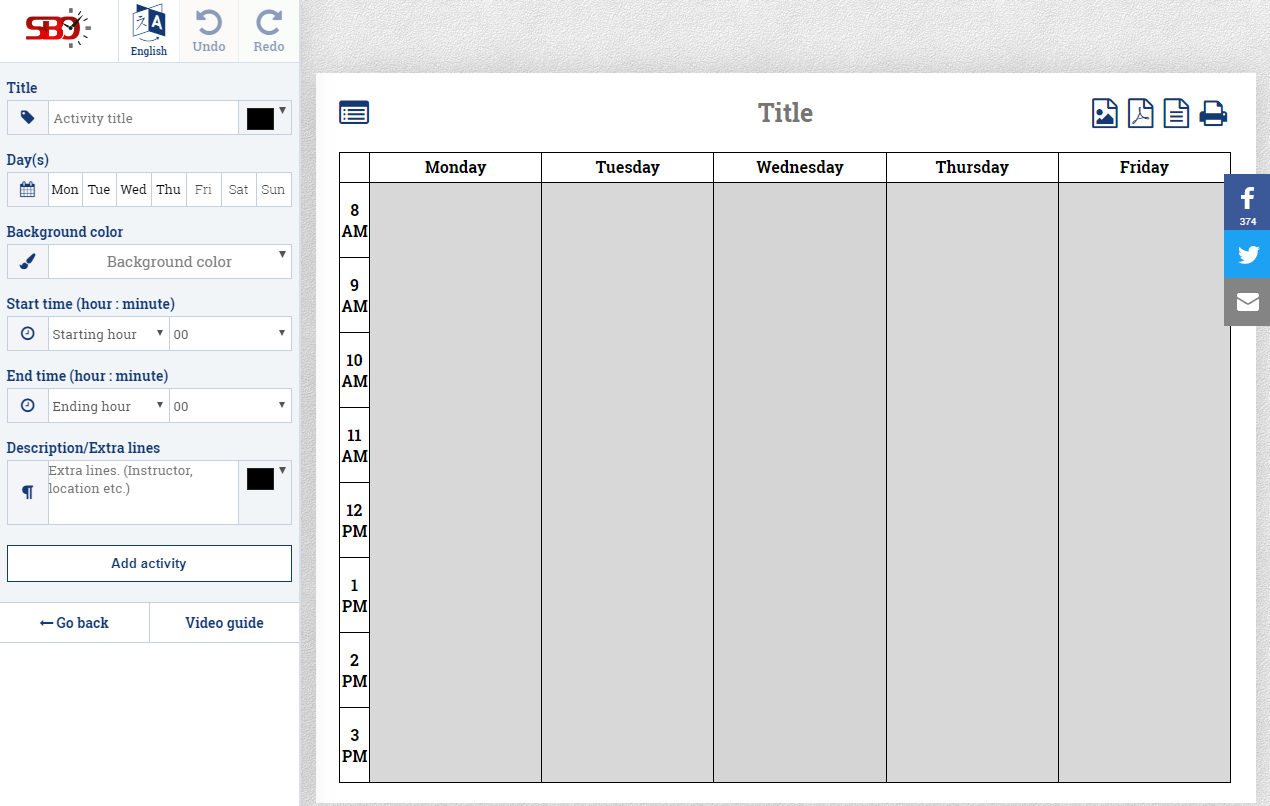
**“Schedule Builder Online”** не е много популярна система за изготвяне на учебни графици. Сравнение със другите подобни на тази технология за съставяне на разписания системи, “Scheduler Builder Online” предлага не малко допълнителни услуги като:

* Запазване на изготвения график под формата на: jpg(графично изображение), pdf(преносим формат на документа) или разпечатване на хартиен носител при наличието на принтер от страна на клиента.
* Създаване на безплатен профил, позволяващ общуване между регистрираните потребители чрез вградена чат система.
* Допълнително „разкрасяване“ на изготвения график като например задаването на фон на изготвения график.

**Минуси:**

* Не толкова удобен за ползване, но пък е предоставено ръководство за употреба.
* Според потребителите, предоставели обратна връзка, свързана със функционалността на системата, се разбира, че има доста технически дефекти.
* Системата не е приспособена за работа от смартфони, таблети или устройства с по-малък екран.

Фиг. 4 – Начална страница на **“Schedule Builder Online”**



**2.1.4 Извод от анализа на представените системи за изготвяне на учебен график**

Според разгледаните по-горе приложения, става ясно, че повечето безплатни за употреба, разработени системи за съставяне на учебни графици са:

* Не завършени
* Не добре оптимизирани от гледна точка на дизайн
* За разработката на системите са използвани доста остарели версии на езици за програмиране, които водят до технически дефекти и възможни пробиви във сигурността.
* Липсващ алгоритъм за автентичност на графиците, който следи за конфликти между занятията или събитията, които биват разпоредени в даден ден и час.
* Липсващ алгоритъм за следене на дублиране на занятия или събития, въведени от потребителите

Но от друга страна, те ни придоставят доста услуги, които сами по себе си са достатъчни за изготвянето на макар и индивидуален, функциониращ учебен график. Подобни безплатни типове системи като “Schedule Builder Online” и „StudyGizmo“ са достъпни и до голяма степен биха свършили работа ако става на въпрос за изготвянето на сравнително прост, индивидуален учебен седмичен график. Друга положителна страна е гъвкавостта им. Те могат да бъдат ползвани както за учебни занятия, така и за най различни събития от ежедневието.

Възможни и подобрения на анализираните системи:

* Интегриране на алгоритъм за автентичност на графиците.
* Подобряване графичния интерфейс
* Разширяване на системите с цел даване възможност на потребителите да използват приложенията от мобилни устройства като смартфони и таблети

## 2.2 Цели

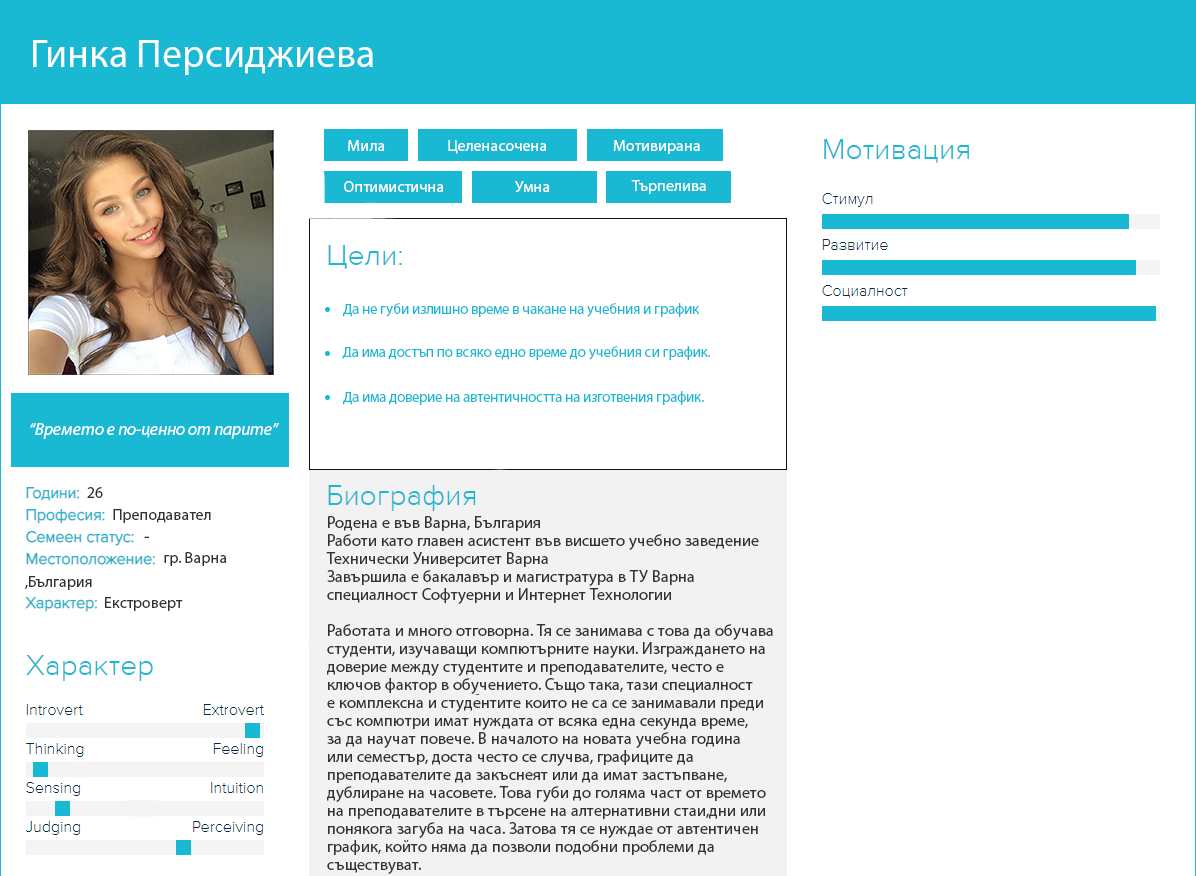
Целта на разработката е създаване на система, която ще предоставя възможност на персонала, отговорен за изготвянето на учебните разписания за преподавателите във висше учебно заведение, да изготвят за кратък период от време, автентични графици на учебните занятията. Едно от най-важните функционалности на системата е проверката за автентичност на даден график. Блогадарение на която, не позволява на персонала, изготвящ дадено разписание да допускат грешки при създаването. Системата не само ще намали до голяма степен времето за изготвяне на разписание, но и ще намали заетоста на персонала, на който ще му бъде възложена тази задача. Всеки изготвен график ще бъде публикуван на уеб платформа, която е достъпна от устройства, ползващи интернет, като смартфони, таблети, лаптопи, компютри. Тази платформа е достъпна единствено от оторезирани преподаватели. Всеки един преподавател има създаден профил, в който се намират неговите разпоредени от оторизиран персонал на висшето учебно заведение, занятия. Той, както и други регистрирани във системата преподаватели, имат достъп до информацията помежду си. Благодарение на скороста и автентичноста на изготвените графици, преподавателите ще избегнат проблеми, като застъпване на стаи, упражнения, дублиране, закъснение при разгласяване на разписанията и други възникнали през годините проблеми.

Системата предлага:

* Опростен и интуитивен дизайн за бързо и лесно ползване от страна на преподавателите
* Опростен и интуитивен дизайн за бързо и лесно ползване от страна на персонала, отговорен за изготвянето на учебния разпис.
* Възможност за достъп от преподавателите до издадените графици по всяко едно време, без значение от устройството, стига то да бъде свързано с интернет мрежата.
* Автентичност, изразяваща се във алгоритъм за валидиране при създаване на графиците
* Моментална актуализация при промяна на график
* Възможност за работа разстояние, касаеща изготвящия графици персонал

## 2.2 Таргет група от хора, към които е фокусирано приложението.

Приложението е насочено за хората, част от висшите учебни заведения, като студенти, преподаватели и персонала, който се занимава със изготвянето на учебните разписания. Разработената система е предназначена само и единствено от този оторизирани потребители. Хора които не са част от висшето учебно заведение нямат достъп до тази функционалност.

Фигура.5 - Потребител, който е част от таргета, към който е фокусирана платформата.

Фигура 5 е един пример за потребител, който вероятно ще се нуждае от услугите, които платформата предлага или с чиято цел е създадена. Накратко това е потребител, чиято работа е да бъде преподавател във висше учебно заведение. Занимава се със обучението на студенти и най-важното нещо за него е да не губи времето на студентите си. Често се случва, в разписанието за часовоте да има грешки или да се получават конфликти с разписанията на други преподаватели, което довежда в загубата на времето на преподавателя и студентите които имат занятие.

## 2.3 Общи изисквания на потребителя

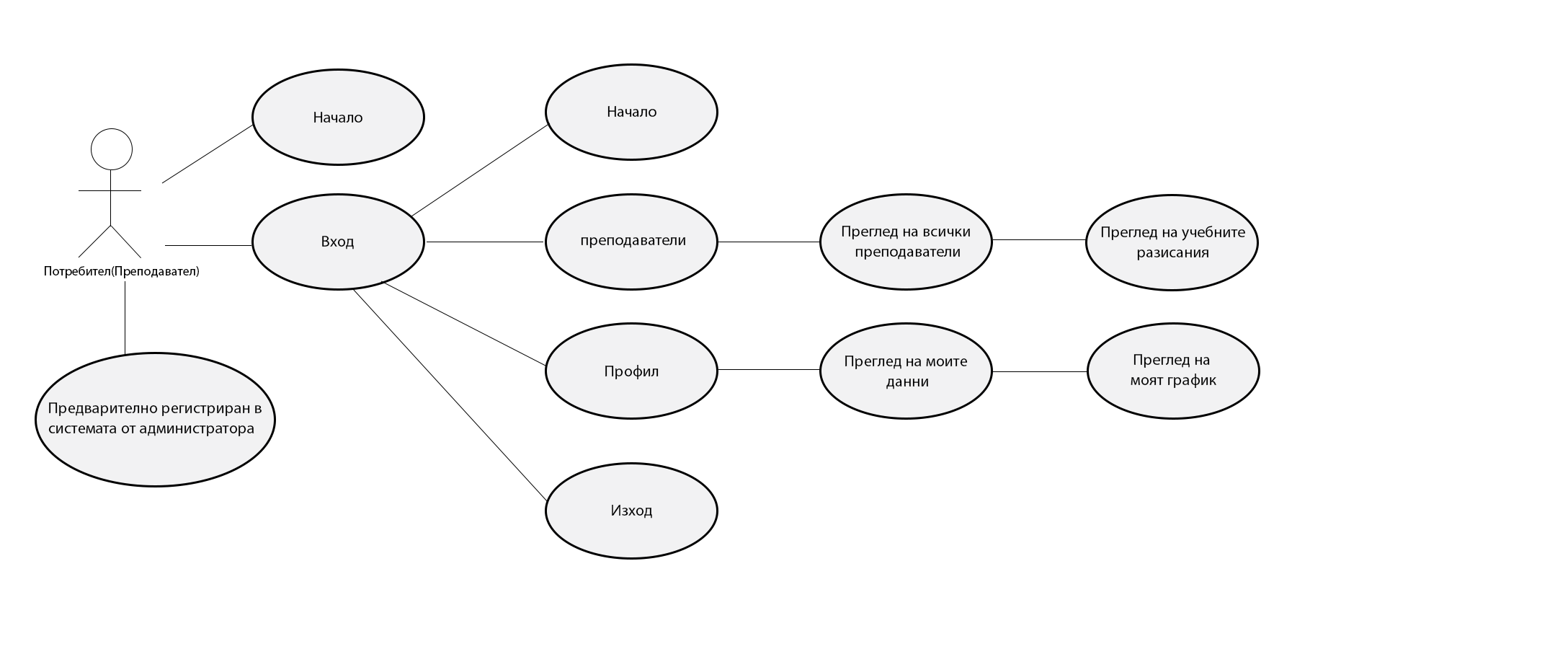
* Опростен и интуитевен дизайн за бързо и лесно ползване.
* Постигане на женалия резултат с възможно най-малко стъпки.
* Платформата да е лека и оптимизирана
* Данните на всеки един потребител да бъдат защитени.
* Уеб платформата да покрива всички защитни механизми с цел защита на информацията на потребителите

## 2.3 Общи изисквания на потребителя

* Автентичност на учебните разписания
* Лесен достъп до графиците
* В случай на промяна. Възможността за моментална проверка на новостите в разписанието

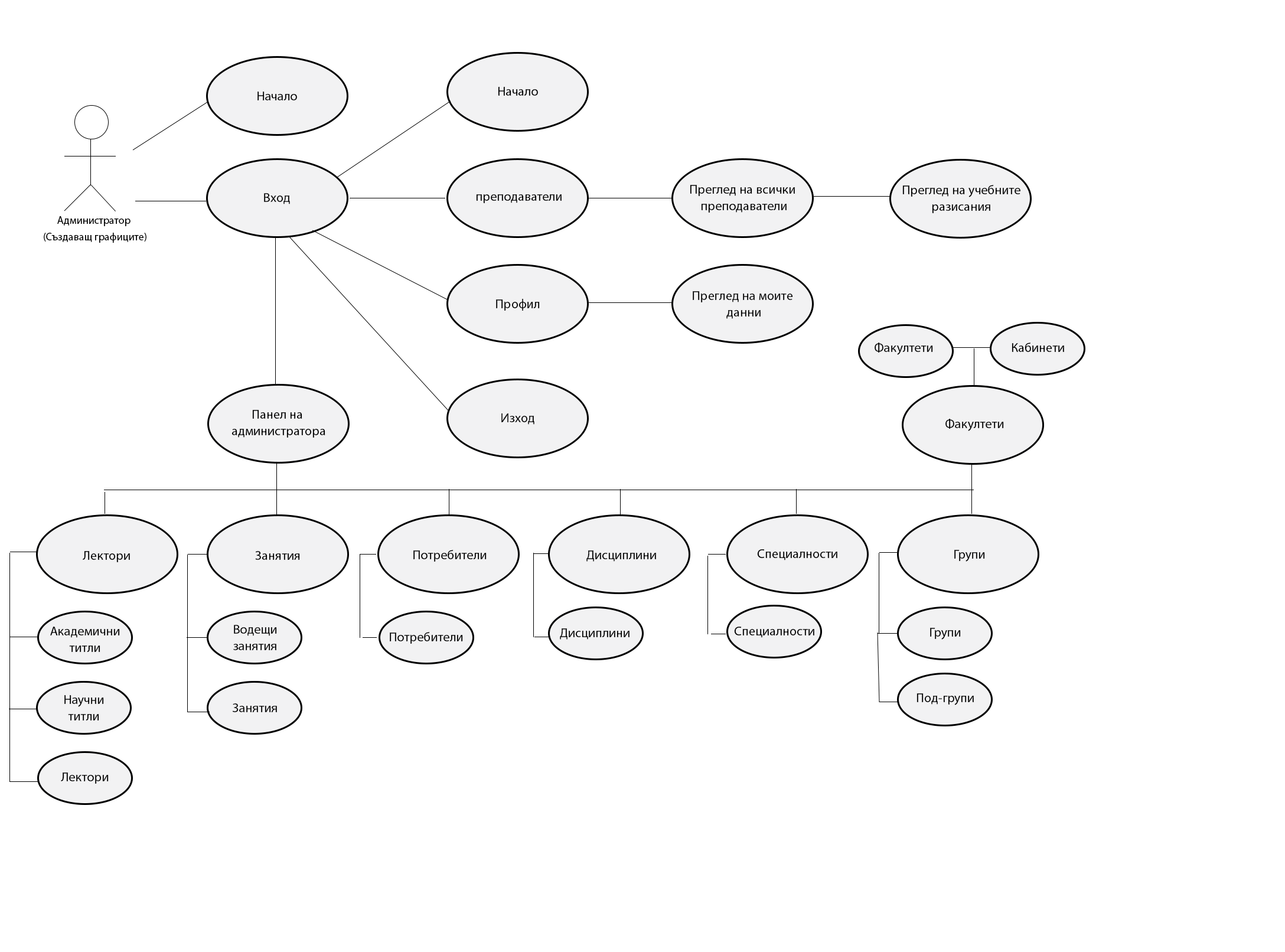
## 2.4 Use-case

### 2.4.1 Преподавател



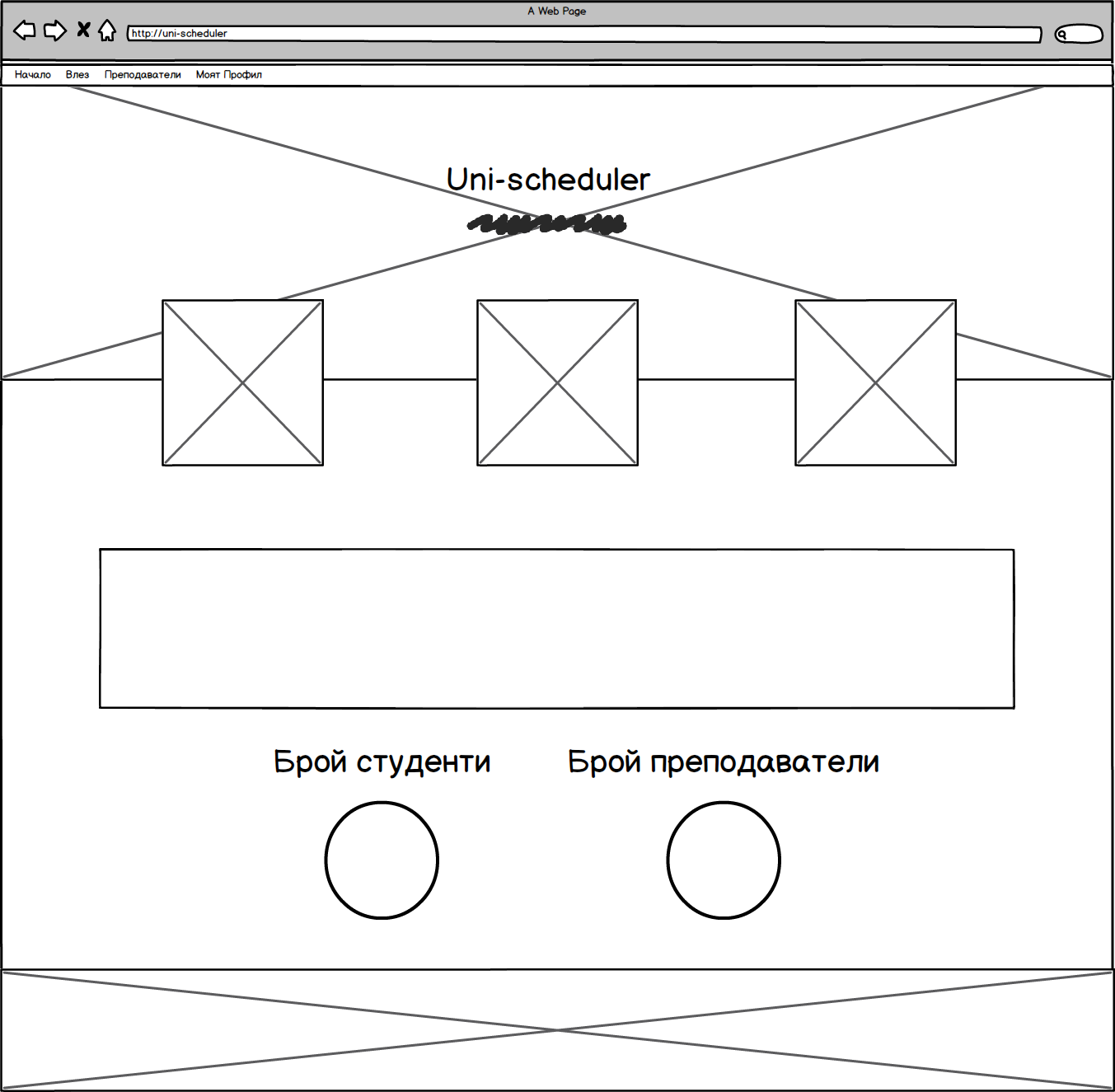
Фиг. 6 – Use-case диаграма за преподавателите.

### 2.4.3 Администратор (Разпореждащия учебни занятия)

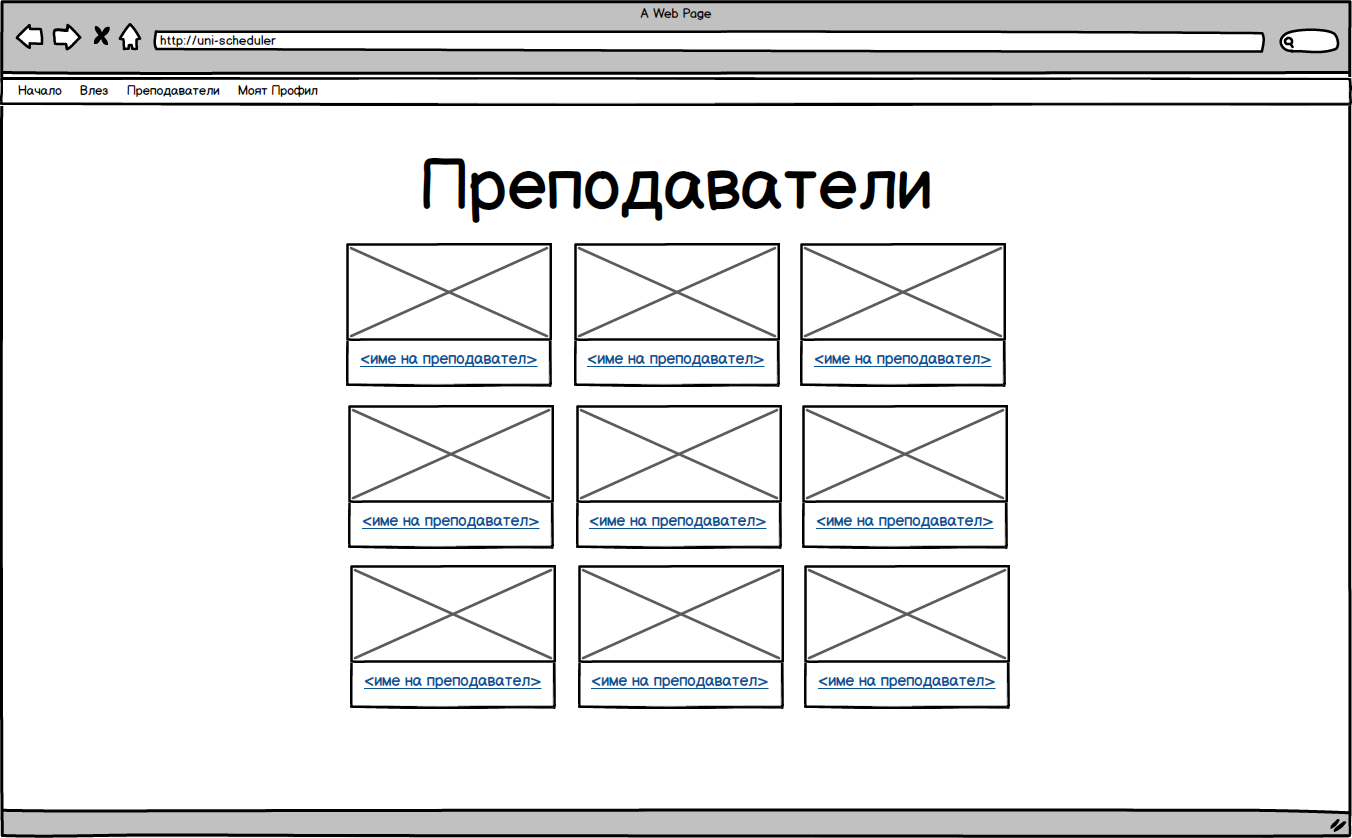


Фиг. 6 – Use-case диаграма за администратор.

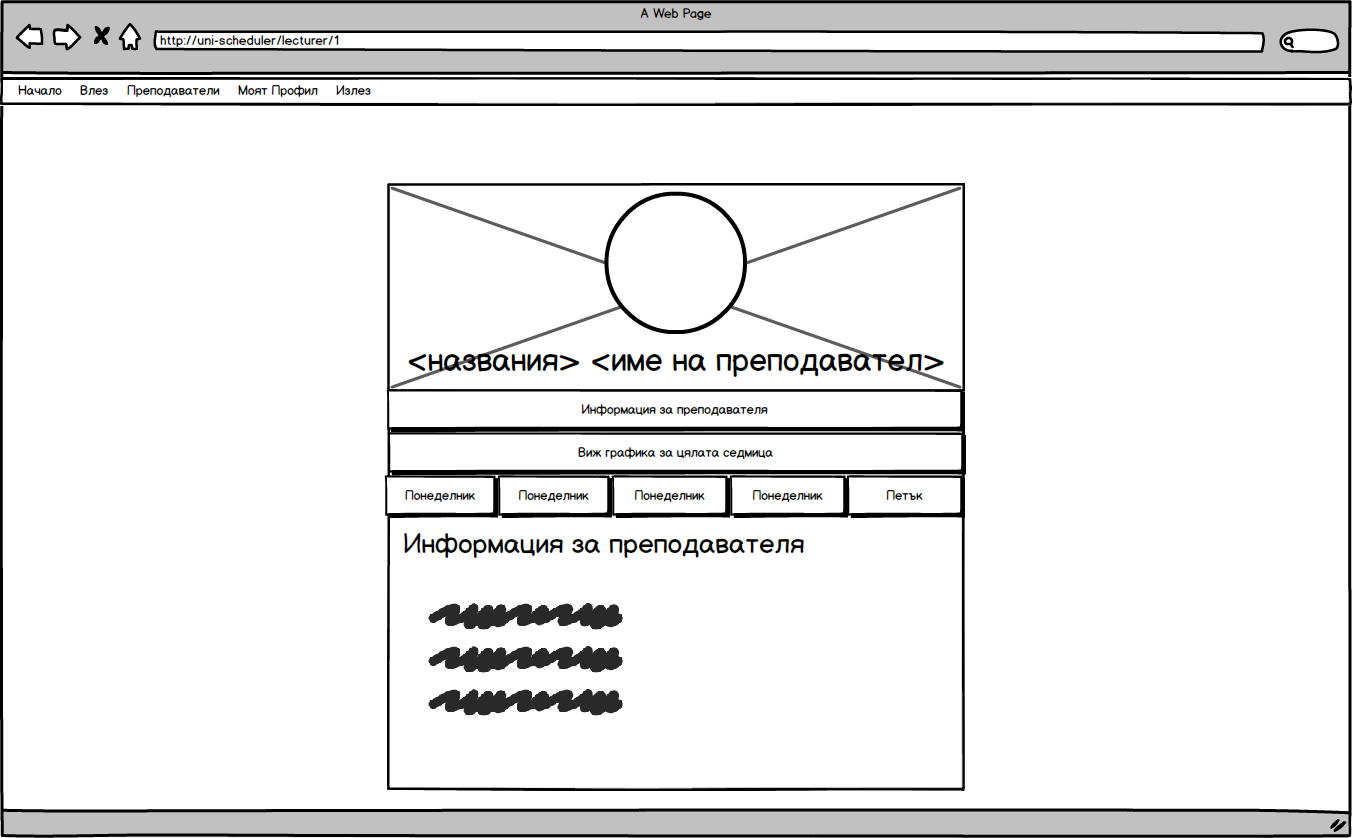
## 2.5 Прототип на Uni-scheduler

Фиг. 7 Протип на начална страница

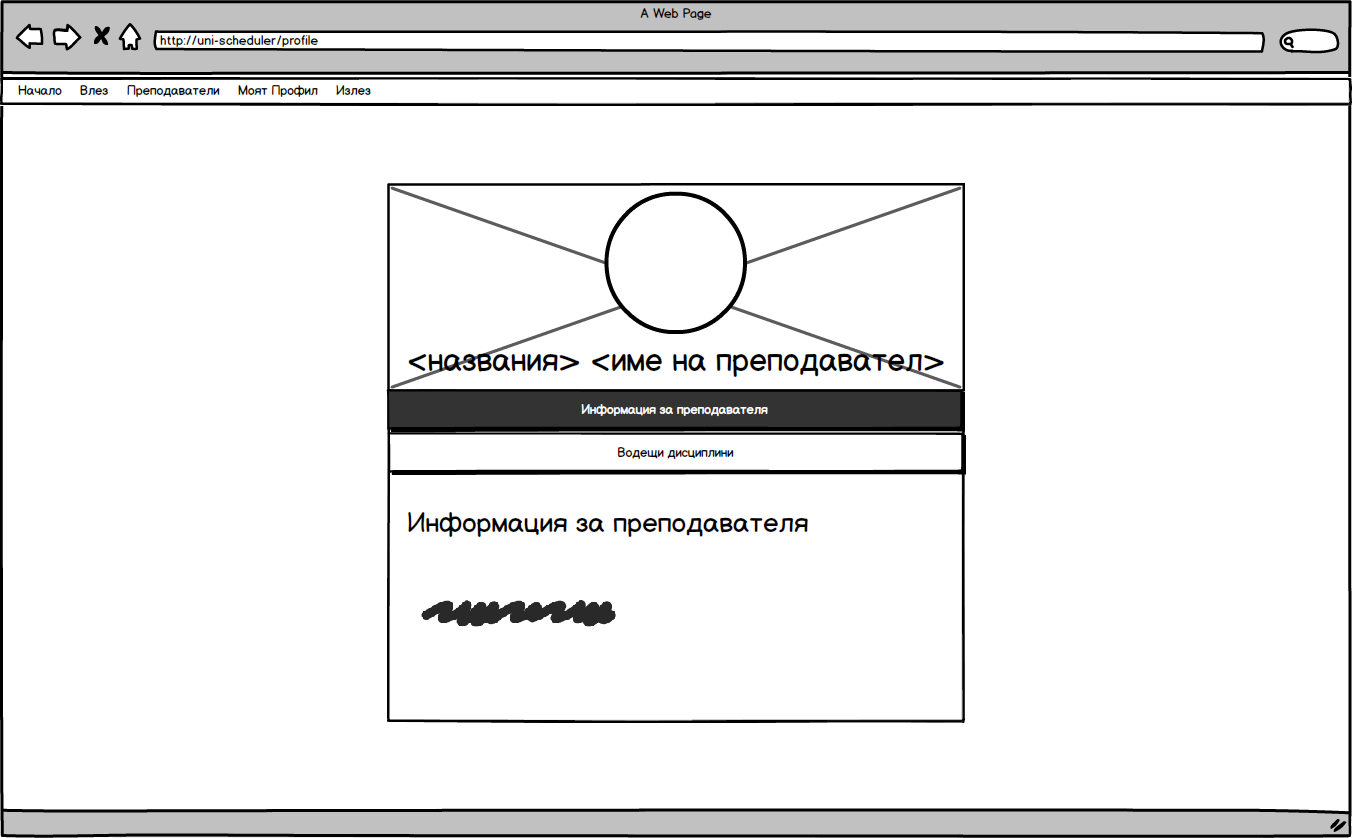
Фиг. 8 Прототип на страница за преглед на преподавателите



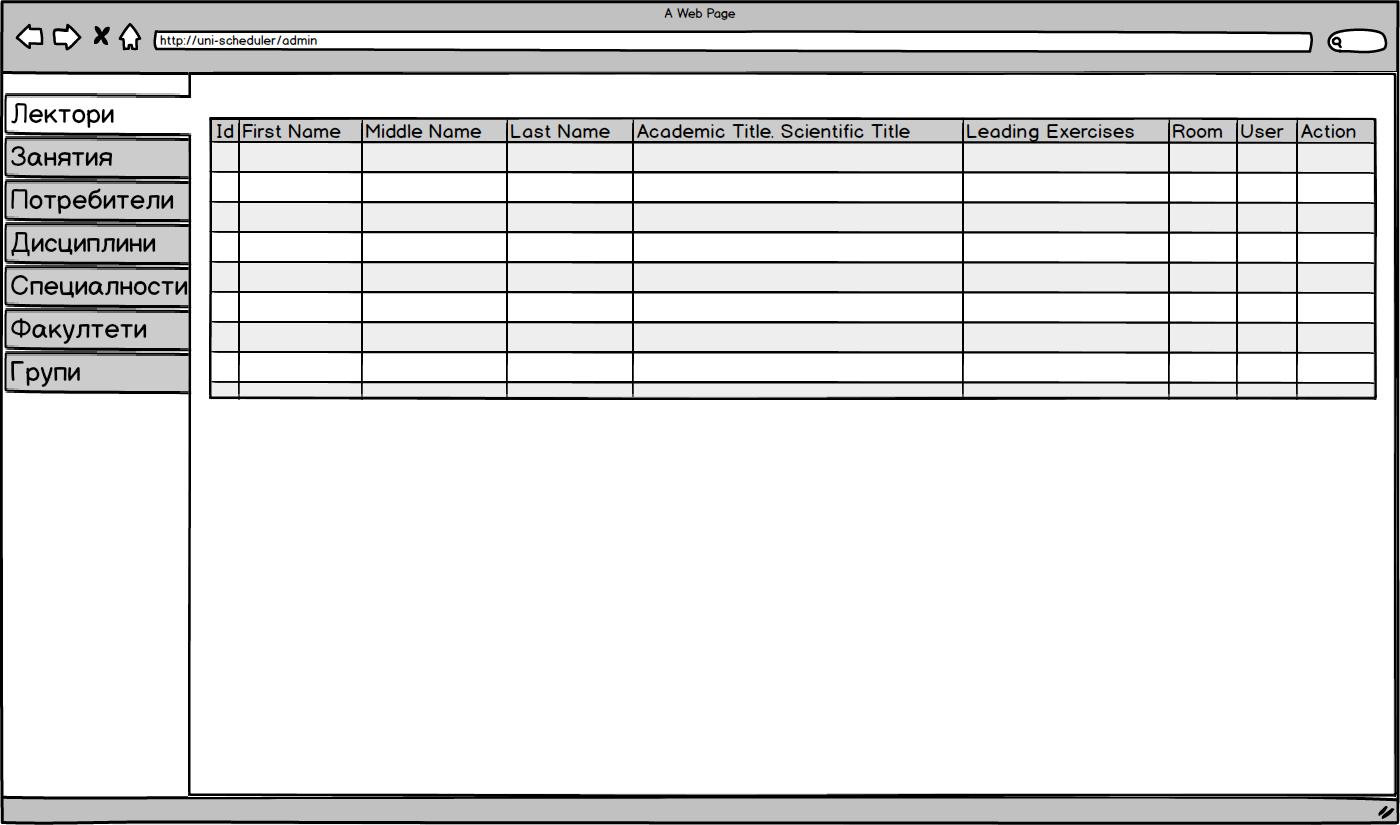
Фиг. 9 Прототип на страницата за даден преподавател



Фиг. 10 Прототип на страницата за информация на потребителя



Фиг. 11 Протип на администраторският панел



# **Глава 3: Технологична схема за разработка на учебни разписи**

## 3.1 Описание на данните

### 3.1.1 Представяне на роли

Потребителска роля е предварително зададена категория, които могат да бъдат присвоявани на потребителите въз основа на техните длъжност или други критерии. Роли обикновено се използват за представяне на персонализирани версии или изгледи на шаблон на формуляр към различни типове потребители. Има много сценарии, където потребителските роли са от полза за потребители. Например с потребителски роли, може:

* Показвате уникален изглед за всяка потребителска роля
* Показвате различна секция за всяка потребителска роля
* Задаване на проверка на данни за поле, който се базира на всяка потребителска роля
* Подаване на данни на формуляр към външен източник на данни, който се базира на всяка потребителска роля

**Преподавател**

Потребителите с роля „Преподавател“ имат достъп до списък със всички преподаватели и до техните учебни разписи. Всеки преподавател може да разгледа своите данни, както и поетите учебни занятия. Потребителите с тази роля нямат достъп до администраторският панел, който позволява да се съдават, изтриват и корегират информация.

**Администратор**

Потребителите с роля „Админстратор“ имат възможността да ползват всички услуги които един преподавател може да използва. Този потребител има достъп до администраторския панел. Работата на администратора е да създава, редактира и изтрива информация. Той се занимава със разпореждането на занятията. Има достъп до информацията на всички регистрирани преподаватели. Има правото да възлага администраторски привилегии на потребители.

### 3.1.4 Представяне на бизнес правила

Бизнес правилата са много важна част от изграждането на базата данни. Според тях се създават обектите с техните специфични атрибути, връзките между самите обекти и валидациите(ограничения). Има два вида бизнес правила – структурни и процедурни.

* Структурните – определят информацията която се съхранява.
* Процедурни – представляват изискванията на работния процес на даден бизнес.

**Структурни**

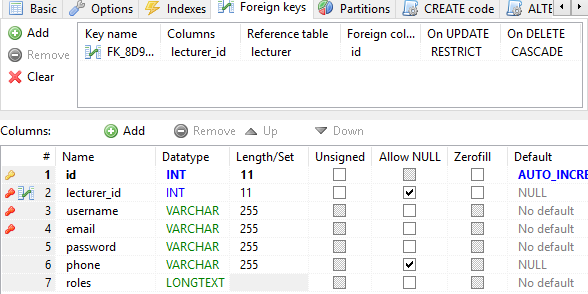
* Един преподавател може да има само един потребителски акаунт
* Един потребителски акаунт може да бъде използван само от един преподавател
* Един преподавател може да има само една академична титла
* Много преподаватели може да имат много научни титли
* Един преподавател може да има много водещи занятия
* Много водещи занятия могат да принадлежат само на една специалност
* Много водещи занятия могат да принадлежат само на една дисциплина
* Едно водещо занятие може да има много занятия
* Една група може да има в себе си много под-групи
* Много групи могат да имат само една специалност
* Много под-групи могат да имат само една група
* Един факултет може да съдържа в себе си много стаи
* Много стаи могат да бъдат съдържани в един факултет
* Много занятия могат да имат една стая
* Много занятия могат да имат едно водещо занятие
* Много занятия могат да имат една под-група

**Процедурни**

* Потребители могат да се впишат в системата, използвайки имейл и парола.
* Преподавателите могат да разглеждат други преподаватели и техните учебни разписания
* Администратор може да разпорежда занятия на преподавател, ако този преподавател е рапореден да води на съответната дисциплина
* Администратор разпоредил занятие за даден преподавател, може да го изтрие или поправи
* Търсене на занятия по конкретен преподавател, начало на занятие и ден с цел проверка за автентичност
* Търсене на водещо занятие по конкретен преподавател, конкретна дисциплина и тип на водещото занятие с цел избягване на дублиращи се разпоредени водещи занятия
* Търсене на водещо занятие по конкретна дисциплина и тип на водещото занятие с цел избягване от разпореждане на водещо занятие от тип лекция, на повече от един преподавател
* Търсене на дисциплина по конкретно име и абревиатура с цел избягване на дублиращи се дисциплини
* Търсене на занятие по начало на занятието, водещо занятие, ден, край на занятието, стая и под-група с цел избягване на вече поети/дублиращи се занятия
* Търсене на факултети по име и абревиатура с цел избягване на дублиращи се факултети
* Търсене на групи по име с цел избягване създаване на групи, които вече съществуват за съответната специалност
* Търсене на стаи по номер на стая и факултет с цел избягване от дублиране на стаи които вече принадлежат на съответният факултет.
* Търсене на специалност по начало на година(поток), име и абревиатура с цел избягване дублиране на специалности с еднакви имена и година(поток) които вече съществуват.
* Търсене на под-групи по име, принадлежаща на съответна група с цел избягване дублиране на вече съществуваща под-група.
* Търсене на потребители по потребителско име, имейл и преподавател с цел избягване дублиране на потребители
* Търсене не преподаватели с цел визуализиране броя на преподавателите
* Търсене на конкретен преподавател по конкретен ден от седмицата с цел визуализация на занятията поети в този ден и този преподавател

### 3.1.5 Определяне на обектите с техните атрибути и релациите

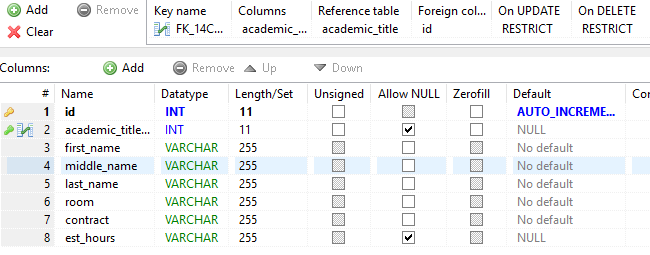
**Потребители (User)**



Фиг. 12 – Таблица User, която съдържа информация за потребителите.

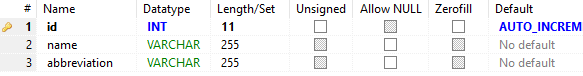
* Идентификационен номер(id) – първичен ключ
* Вторичен клюз за преподавател(lecturer\_id) свързан с първичния ключ към таблица на преподавателите, уникално поле
* Потребителско име(username) – уникално и задължително поле
* Имей (email) – уникално и задължително
* Парола (password) – криптирана с „Bcrypt“ алгоритъм
* Телефонен номер (phone) – незадължително поле
* Роля (role) – задължително поле

**Преподавател (Lecturer)**



Фиг. 13 Таблица Lecturer, която съдържа информация за преподавателите

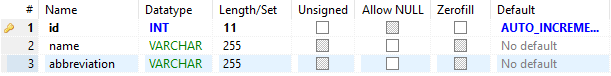
* Идентификационен номер(id) – първичен ключ
* Вторичен ключ за академично название на преподавателя, свързан с таблица „academic\_title“
* Име (first\_name) – задължително поле
* През име(middle\_name) - задължително поле
* Фамилия(last\_name) - задължително поле
* Стая на преподавателя (room) – задължително поле
* Тип на договора (contract) - задължително поле



**Академично название (academic\_title)**

Фиг. 14 Таблица „academic\_title”, която съдържа видовете академични названия

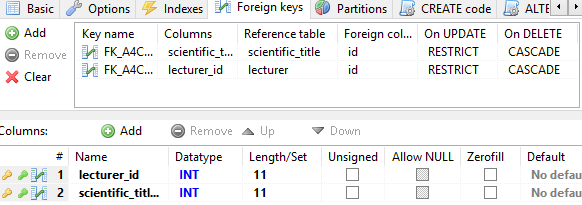
* Идентификационен номер(id) – първичен ключ
* Име (name) – задължително поле
* Абревиатура (abbreviation) – задължително поле

**Научно название (scientific\_title)**

Фиг. 14 Таблица „scientific\_title“, която съдържа информация за всички научни названия

* Идентификационен номер(id) – първичен ключ
* Име (name) – задължително поле
* Абревиатура (abbreviation) – задължително поле

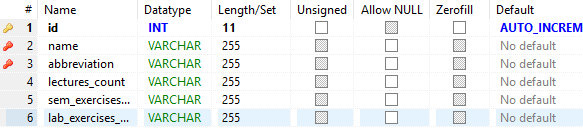
**Научно название / Преподаватели (lecturer\_scientific\_title)**



Фиг. 15 Таблица „lecturer\_scientific\_title“, която играе ролята на междинна таблица, свързваща таблицата на преподавателите и научните названия

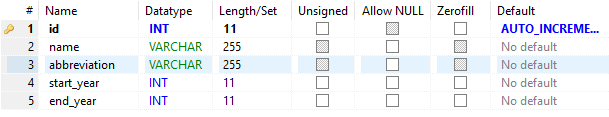
* Вторичен ключ за научно название, свързан с таблица „scientific\_title“
* Вторичен ключ за преподавател, свързан с таблица „lecturer“

**Дисциплини (discipline)**



Фиг. 16. Таблица „discipline”, която съдържа информация за дисциплините

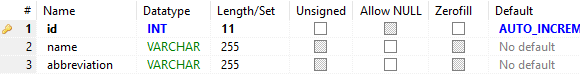
* Идентификационен номер(id) – първичен ключ
* Име (name) – задължително поле
* Абревиатура (abbreviation) – задължително поле
* Брой лекции за семестъра (lectures\_count) – задължително поле
* Брой семинарни упражнения за семестъра (sem\_exercise\_count) – задължително поле
* Брой лабораторни упражнения за семестъра (lab\_exercise\_count) – задължително поле

**Специалност (specialty)**

Фиг. 17. Таблица „specialty”, която съдържа информация за специалностите.

* Идентификационен номер(id) – първичен ключ
* Име (name) – задължително поле
* Абревиатура (abbreviation) – задължително поле
* Първа година на специалността/поток (start\_year) – задължително поле
* Последна година на специалността/поток (end\_year) – задължително поле

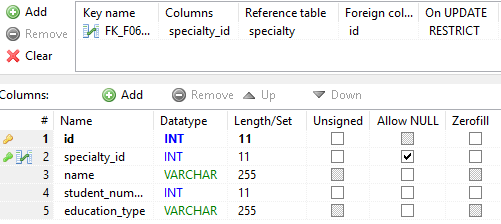
**Факултет (faculty)**



Фиг. 18 Таблица „faculty ”, която съдържа информация за факултетите.

* Идентификационен номер(id) – първичен ключ
* Име (name) – задължително поле
* Абревиатура (abbreviation) – задължително поле

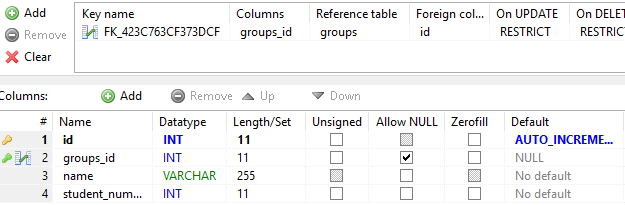
**Групи (groups)**



Фиг. 19 Таблица „groups ”, която съдържа информация за групите.

* Идентификационен номер(id) – първичен ключ
* Вторичен ключ за специалност, свързан с таблица „specialty“
* Име (name) – задължително поле
* Брой студенти (student\_number) – задължително поле
* Тип на обучение (education\_type) – задължително поле

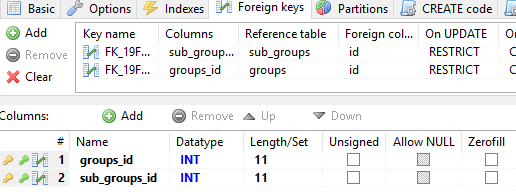
**Под-групи (sub\_groups)**



Фиг. 20 Таблица „sub\_groups ”, която съдържа информация за под-групите.

* Идентификационен номер(id) – първичен ключ
* Вторичен ключ за група, свързан с таблица „groups“
* Име (name) – задължително поле
* Брой студенти (student\_number) – задължително поле

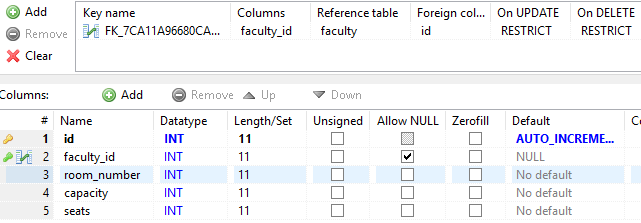
**Групи / Под-групи (sub\_groups)**



Фиг. 21 Таблица „groups\_sub\_groups”, която играе ролята на междинна таблица, свързваща таблицата „groups” и „sub\_groups“

* Вторичен ключ за група, свързан с таблица „groups“
* Вторичен ключ за под-група, свързан с таблица „sub\_groups“

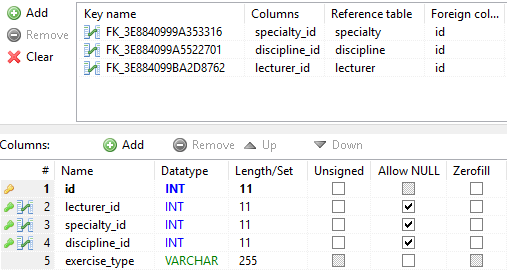
**Стаи (rooms)**



Фиг. 22 Таблица „rooms”, която съдържа информация за стаите.

* Идентификационен номер(id) – първичен ключ
* Вторичен ключ за факултет, свързан с таблица „faculty“
* Номер на стая (room\_number) – задължително поле
* Капацитет (capacity) – задължително поле
* Работни места (seats) – задължително поле

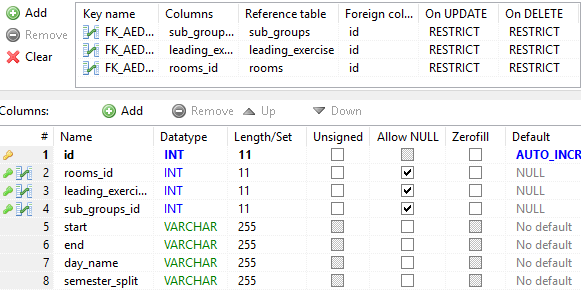
**Водещи занятия (leading\_exercises)**



Фиг. 23 Таблица „leading\_exercises”, която съдържа информация за водещите занятия.

* Идентификационен номер(id) – първичен ключ
* Вторичен ключ за преподавател, свързан с таблица „lecturer“
* Вторичен ключ за специалност, свързан с таблица „specialty“
* Вторичен ключ за дисциплина, свързан с таблица „discipline“
* Вид на занятието – задължително поле

**Занятия (exercises)**

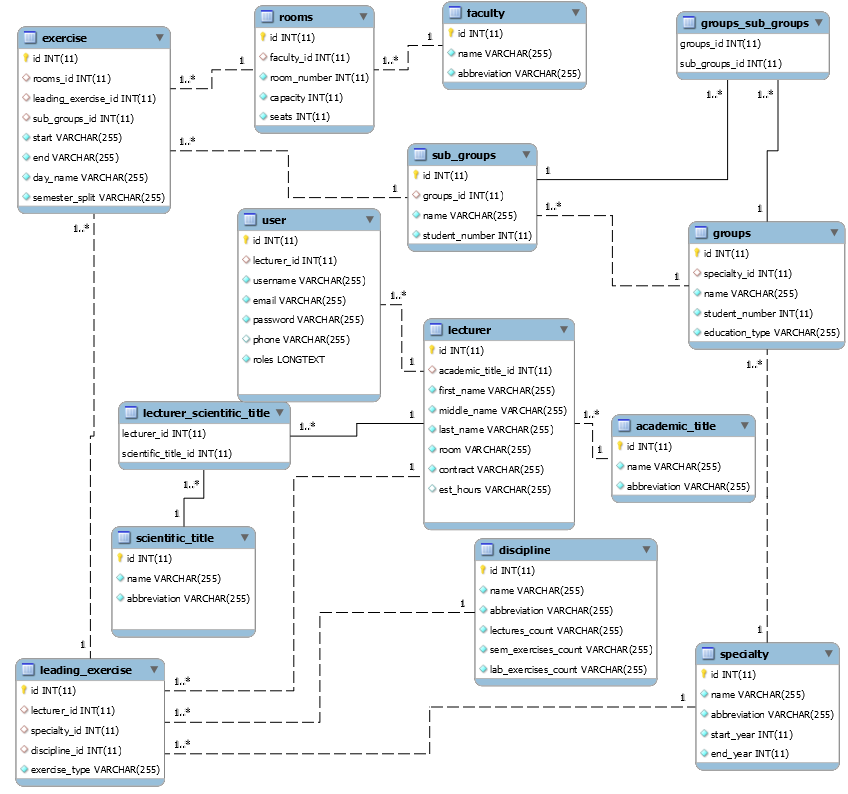


Фиг. 24 Таблица „exercises”, която съдържа информация за занятията.

* Идентификационен номер(id) – първичен ключ
* Вторичен ключ за стаи, свързан с таблица „rooms“
* Вторичен ключ за водещи занятия, свързани с таблица „leading\_exercises“
* Вторичен ключ за под-групи, свързани с таблица „sub-groups“
* Начало на занятието – задължително поле
* Край на занятието – задължително поле
* Име на ден от седмицата – задължително поле
* Зимен/Летен семестър – задължително поле

### 3.1.6 Релационна диаграма

Фиг. 25 – Релационна диаграма



## 3.2 Използвани технологии

### 3.2.1 Уеб приложение

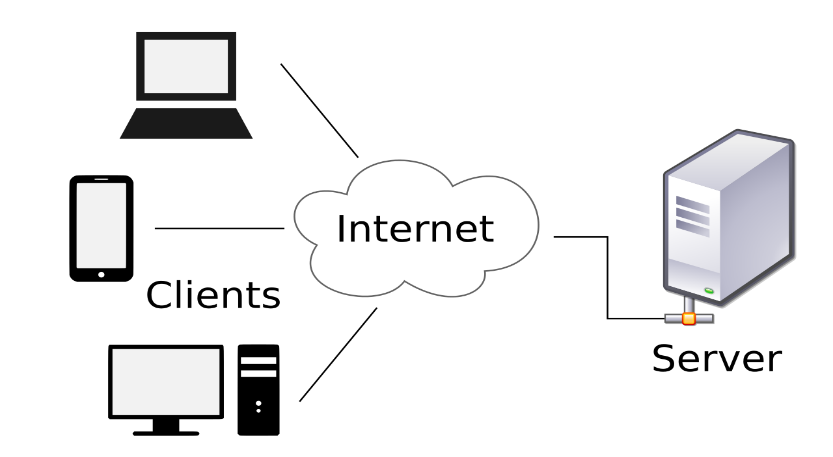
Уеб приложение (Уеб приложение) е приложна програма, която се съхранява на отдалечен сървър и се достъпва през Интернет чрез браузърен интерфейс. Уеб услугите са уеб приложения по дефиниция и много, макар и не всички, уебсайтове съдържат уеб приложения. Те могат да бъдат проектирани за голямо разнообразие и могат да бъдат използвани от организация до физическо лице по множество причини. Често използваните уеб приложения могат да включват уеб поща, онлайн калкулатори или магазини за електронна търговия. Някои уеб приложения могат да бъдат достъпни само от конкретен браузър; повечето обаче са достъпни независимо от браузъра.

**Ползи:**

Уеб приложенията имат много различни приложения, а с тези приложения има много потенциални ползи. Някои техни общи предимства включват:

* Разрешаване на достъп на множество потребители до една и съща версия на приложение.
* Не е необходимо да се инсталират уеб приложения.
* Достъпа до уеб приложенията може да се осъществява чрез различни платформи като десктоп, лаптоп или мобилни.
* Достъпът може да се осъществява чрез множество браузъри.

### 3.2.1.1 Клиент-сървър

Това е архитектура на компютърна мрежа, в която много клиенти (отдалечени процесори) искат и получават услуга от централизиран сървър (хост компютър). Клиентските компютри предоставят интерфейс, който позволява на компютърния потребител да изисква услуги на сървъра и да Фиг. 26 – Клиент-сървър архитектура

показва резултатите, които сървърът връща. Сървърите изчакват заявките да пристигнат от клиентите и след това отговарят на тях. В идеалния случай сървърът предоставя стандартизиран прозрачен интерфейс за клиентите, така че те не трябва да са запознати със спецификата на системата (т.е. хардуера и софтуера), която предоставя услугата. Често са разположени на работни станции или на персонални компютри, докато сървърите са разположени другаде в мрежата, обикновено на по-мощни машини. Този изчислителен модел е особено ефективен, когато клиентите и сървърът имат различни задачи, които рутинно изпълняват. Например в болничната обработка на данни клиентският компютър може да работи с приложна програма за въвеждане на информация за пациента, докато сървърният компютър работи с друга програма, която управлява базата данни, в която информацията е трайно съхранявана. Много клиенти могат да получат достъп до информацията на сървъра едновременно и в същото време клиентският компютър може да изпълнява други задачи, като изпращане на електронна поща.

### 3.2.1.2 Клиент

Клиентът може да бъде обикновено приложение или цяла система, която получава достъп до услуги, предоставяни от сървър. Клиентът може да се свърже със сървър чрез различни средства като домейни сокети, имена, споделена памет или чрез интернет протоколи, което е най-разпространеният метод, използван след широкото приемане на Интернет.

### 3.2.1.3 Сървър

Почти всички персонални компютри могат да служат като мрежови сървъри. Въпреки това обикновено компютрите със софтуерна / хардуерна система имат функции и конфигурации само за тази задача. Например, специализираните сървъри може да имат високоефективна „RAM“ памет, по-бърз процесор и няколко твърди диска с голям капацитет. В допълнение, специализирани сървъри могат да бъдат свързани към излишни източници на захранване, няколко мрежи и други сървъри. Такива функции и конфигурации на връзката са необходими, тъй като много клиентски машини и клиентски програми могат да зависят от тях, за да функционират ефективно, правилно и надеждно.

За да работи в уникалната мрежова среда, в която много компютри и хардуер / софтуерни системи зависят само от един или няколко сървърни компютъра, сървърът често има специални характеристики и възможности, включително:

* Възможността за актуализиране на хардуер и софтуер без рестартиране.
* Разширена възможност за архивиране на критични данни.
* Напреднала производителност в мрежата.
* Автоматичен (невидим за потребителя) пренос на данни между устройства.
* Висока сигурност на ресурси, защита на данните и паметта.

Сървърните компютри често имат специални операционни системи, които обикновено не се срещат на персонални компютри. Някои операционни системи се предлагат както в сървърна, така и в настолна версия и използват подобни интерфейси.

### 3.2.2 Структура на уеб приложението

**3.2.2.1** Модел-Изглед-Контролер (MVC)

Фиг. 27 – Графично представяне на Модел-Изглед-Контролер архитектура

Шаблонът определя не само ролите, които обектите играят в приложението, той определя начина, по който обектите общуват помежду си. Всеки от трите типа обекти е отделен от останалите чрез абстрактни граници и комуникира с обекти от другите типове през тези граници. Колекцията от обекти от определен тип „MVC“ в едно приложение понякога се нарича слой - например моделен слой. MVC е в основата на добрия дизайн за приложение. Ползите от възприемането на този модел са многобройни. Много обекти в тези приложения са по-често използваеми и техните интерфейси са по-добре дефинирани. Приложенията, използващи този дизайн също са по-лесно разширяващи се от другите приложения. Освен това много технологии и архитектури са базирани на MVC и изискват използването на този шаблон.

**Моделни обекти(Model):**

Тези обекти защитават данните, специфични за дадено приложение и определят логиката и изчисленията, които манипулират и обработват тези данни. Например, моделен обект може да представлява герой в игра или контакт в адресната книга.Той може да има връзки към един и към много други подобни обекти и затова понякога моделния слой на приложението ефективно е една или повече обектни графики. Голяма част от данните, които са част от устойчивото състояние на приложението, трябва да пребивават в моделните обекти след зареждането на данните в приложението. В идеалния случай обектът на модела не трябва да има изрична връзка с обектите на изглед (View), които представят неговите данни и позволяват на потребителите да редактират тези данни - не трябва да се занимават с потребителски интерфейс и проблеми с представянето.

* **Комуникация:**

Действията на потребителя в изгледания слой, които създават или променят данни, се предават чрез контролер и водят до създаването или актуализирането на моделен обект. Когато моделен обект се промени (например нови данни се получават през мрежова връзка), той уведомява контролера, който актуализира съответните изгледи.

**Изгледни обект(View):**

Oбект в приложение, което потребителите могат да виждат. Той знае как да визуализира себе си и може да реагира на действията на потребителя. Основна цел на обектите за изглед е да се показват данни от моделните обекти на приложението и да се даде възможност за редактиране на тези данни. Въпреки това, обектите на изглед обикновено се отделят от моделни обекти в MVC приложение.

* **Комуникация:**

Изгледите научават, че данните в модела са се променили чрез контролер обектите. Всяка една проямяна от потребителя в едно приложение бива съобщавано на котролер, който се грижи за отразяване на информацията в модел обекта

**Контролер обект:**

Контролера действа като посредник между един или повече изгледи на приложение и един или повече от неговите контролер обекти. По този начин те игрят ролята на проводник, чрез който изгледите научават за промените в моделните обекти и обратно. Обектите на контролера също могат да изпълняват задачи за настройка и координиране на приложение и да управляват жизнените цикли на други обекти.

* **Комуникация:**

Контролера интерпретира действията на потребителя, направени в изгледите и съобщава новите или променени данни на моделния слой. Когато обектите на модела се променят, контролера съобщава тези нови данни на изгледите, така че те да могат да визуализират тези промени.

### 3.2.2.2 Обектно-релационно картографиране(ORM)

В компютърните науки означава техника на програмиране за преобразуване на данни между несъвместими типове системи с помощта на обектно-ориентирани езици за програмиране. Това всъщност създава "виртуална база данни на обекти", която може да се използва от езика на програмиране.

В обектно-ориентираното програмиране, задачите за управление на данни действат върху обекти, които почти винаги са не скаларни стойности. Например запис в адресната книга, който представлява един човек заедно с нула или повече телефонни номера и нула или повече адреси. Това би могло да бъде моделирано в обектно-ориентирана реализация от обект „Личност“ с атрибути, за да съдържа всеки елемент от данни, който вписването съдържа: името на лицето, списък с телефонни номера и списък с адреси. Самият списък с телефонни номера би съдържал обект „PhoneNumber“ и т.н. Различни методи могат да бъдат свързани с обекта, като метод за връщане на предпочитания телефонен номер, домашния адрес и т.н.

Въпреки това много популярни продукти на базата данни като системи за управление на бази данни „SQL“ (СУБД) могат да съхраняват и манипулират само скаларни стойности като цели числа и низове, организирани в таблици. Програмистът трябва или да преобразува обектните стойности в групи от по-прости стойности за съхранение в базата данни (и да ги преобразува обратно при извличане), или да използва само прости скаларни стойности в рамките на програмата. Обектно-релационното картографиране реализира първия подход

**Doctrine (ORM)**

„Doctrine2“ е обектно-релационен картограф (ORM) за PHP 7.1+, който осигурява прозрачна устойчивост за PHP обекти. Той цели пълно отделяне на бизнес логиката от постоянството в системата за управление на релационни бази данни. Ползата от „Doctrine“ за разработчика е възможността да се съсредоточи върху обектно-ориентираната бизнес логика.

### 3.2.2.3 Технологична рамка(framework)

Платформа за разработване на софтуерни приложения. Тя осигурява основа, на която разработчиците на софтуер могат да изграждат програми за конкретна платформа. Например, една рамка може да включва предварително създадени класове и функции, които могат да се използват за обработка на вход, управление на хардуерни устройства и взаимодействие със системния софтуер. Това опростява процеса на разработка, тъй като програмистите не трябва да преоткриват колелото всеки път, когато разработят ново приложение.

Технологичната рамка е подобна на интерфейс за приложно програмиране (API), макар че технически рамката включва API. Както подсказва името, една рамка служи като основа за програмиране, докато API осигурява достъп до поддържаните от рамката елементи. Една рамка може също да включва кодови библиотеки, компилатор и други програми, използвани в процеса на разработка на софтуер.

### 3.2.2.4 Виртуални машини:

Виртуална машина (VM) е софтуерна програма или операционна система, която не само показва поведението на отделен компютър, но също така е способна да изпълнява приложения и програми като отделен компютър. Виртуалната машина, обикновено известна като гост, се създава в друга изчислителна среда, наречена "хост". Няколко виртуални машини могат да съществуват в рамките на един хост наведнъж.

Виртуалните машини се реализират чрез методи за емулация на софтуер или техники за хардуерна виртуализация. В зависимост от използването им и нивото на кореспонденция с всеки физически компютър, виртуалните машини могат да бъдат разделени на две категории:

* Системни виртуални машини - системна платформа, която поддържа споделянето на физическите ресурси на хост компютъра между множество виртуални машини, всяка от които работи със собствено копие на операционната система. Техниката за виртуализация се осигурява от софтуерен слой, известен като хипервизор, който може да работи или върху чист хардуер, или операционна система.
* Процесни виртуални машини – Създадени са за осигуряване на независима от платформа среда за програмиране, която маскира информацията на базисния хардуер или операционна система и позволява изпълнението на програмата да се извършва по същия начин във всяка дадена платформа.

**Предимства:**

Позволява множество среди на операционната система на един физически компютър без никаква намеса Виртуалните машини са широко достъпни и са лесни за управление и поддръжка. Предлага възможности за осигуряване на приложения и възстановяване след бедствия

**Недостатъци:**

Те не са толкова ефективни като физическия компютър, защото хардуерните ресурси се разпределят по косвен начин. Множество виртуални машини, работещи на една физическа машина, могат да осигурят нестабилна производителност

### 3.2.2.4 Vagrant:

Vagrant е инструмент за изграждане и управление на среди на виртуални машини в един работен процес. С лесен за използване работен процес и съсредоточаване върху автоматизацията, Vagrant понижава времето за настройка на средата за разработка. Осигурява лесна за конфигуриране, възпроизводима и преносима работна среда, изградена на базата на стандартната за индустрията технология и контролирана от единен последователен работен процес, за да помогне за увеличаване на производителността и гъвкавостта при разработката на приложение. С помоща на скриптови езици като „Ansible“ може да се издигат и конфигурират със всички нужни за разработката на съответното приложение, виртуални машини.

### 3.2.2.5 Ansible:

Ansible е инструмент за автоматизация на информационните технологии. Той може да конфигурира системи, да „установи“ софтуер и да организира по-напреднали ИТ задачи, като непрекъснато внедряване или нулеви актуализации за прекъсване.

Основните цели на Ansible са простотата и лекотата на използване. Той също така има силен акцент върху сигурността и надеждността, като включва минимум движещи се части, използване на „OpenSSH“ за транспорт. Работи съвместно със „YAML“

### 3.2.2.6 YAML

YAML е език за сериализиране на данни, който съответства на очакванията на потребителя по отношение на данните. Той е проектиран така, че да е удобен за хората и да работи перфектно с други езици за програмиране. Обикновено се използва за конфигурационни файлове, но може да се използва в много приложения, където данните се съхраняват.

**Предимства**:

* Съответства на собствените структури от данни на гъвкавата методология и нейните езици като Perl, Python, PHP, Ruby и JavaScript
* Данните YAML са преносими между програмните езици
* Лесно четим от хората
* Поддържа обработка в една посока

### 3.2.2.7 Secure Shell (SSH):

Известен също като „Secure Shell“ или „Secure Socket Shell“, е мрежов протокол, който предоставя на потребителите, особено на системните администратори, сигурен начин за достъп до компютър през незащитена мрежа. SSH също се отнася до пакета от помощни програми, които реализират „SSH“ протокола. „Secure Shell“ осигурява силна автентификация и криптирана комуникация с данни между два компютъра, свързващи се чрез отворена мрежа, като например интернет. „SSH“ се използва широко от мрежовите администратори за дистанционно управление на системи и приложения, което им позволява да влизат в друг компютър по мрежа, да изпълняват команди и да преместват файлове от един компютър на друг. „SSH“ позволява:

* Сигурен отдалечен достъп до мрежови системи или устройства с активиран „SSH“, както за потребители, така и до автоматизирани процеси
* Сигурни и интерактивни сесии за прехвърляне на файлове
* Автоматизирани и защитени прехвърляния на файлове
* Зищитено управление чрез команди към отдалечени устройства или системи
* Зищитено управление на компоненти на мрежовата инфраструктура.

„SSH“ също често се използва в скриптове и друг софтуер, за да се даде възможност на програми и системи за отдалечен и сигурен достъп до данни и други ресурси.

### 3.2.2.8 Система за контрол на версиите (Git):

Git е разпределена система за контрол на версиите за проследяване на промените в изходния код по време на разработката на софтуер. Той е предназначен за координиране на работата между програмистите, но може да се използва за проследяване на промените във всеки набор от файлове. Целите му включват скорост, цялост на данните и поддръжка за разпределени, нелинейни работни процеси. Както при повечето други разпределени системи за контрол на версиите, и за разлика от повечето системи клиент-сървър, всяка Git директория на всеки компютър е пълноценно хранилище с пълна история и пълни възможности за проследяване на версиите, независимо от мрежовия достъп или централен сървър.

### 3.2.2.9 „S.O.L.I.D” принципи:

„SOLID“ е съкращение за пет на брой важни принципа на дизайна, при работа с обектно ориентирано програмиране

* “**S**” Принцип на единна отговорност - при програмирането Принципът на единната отговорност гласи, че всеки модул или клас трябва да носи отговорност за една част от функционалността, осигурена от софтуера.
* „**O**“ Отворен / затворен принцип - софтуерните единици (класове, модули, функции и т.н.) трябва да бъдат отворени за разширения, но затворени за модификация.
* „**L**“ Принцип на заместване на Лисков - той заявява, че обектите в дадена програма трябва да бъдат заменими с екземпляри от техните подтипове, без да променят правилността на тази програма.
* „**I**“ Принцип на разделяне на интерфейса - никой клиент не трябва да бъде принуждаван да зависи от методи, които не използва.
* „**D**“ Принцип на инверсия на зависимостта - модулите от високо ниво не трябва да зависят от модулите с ниско ниво. И двете трябва да зависят от абстракциите. Абстракциите не трябва да зависят от детайлите. Детайлите трябва да зависят от абстракциите.

Прилагайки тези 5 принципа, които правят SOLID съкращението, ние се възползваме от многократна употреба, поддръжка, мащабируемост и лесна тестваема кодова база.

### 3.2.2.10 PHP (Hypertext Preprocessor)

PHP е скриптов език от страна на сървъра, създаден специално за уеб разработка. Може лесно да бъде вграден в HTML файлове, а HTML кодовете могат да бъдат записани и във PHP файл. Нещото, което го разграничава от езика на клиентска страна като HTML е че PHP кодовете се изпълняват на сървъра, докато HTML кодовете се изобразяват директно в браузъра. PHP кодовете първо се изпълняват на сървъра и след това резултатът се връща в браузъра. Единствената информация, която клиентът или браузърът знае, е резултатът, върнат след изпълнение на PHP скрипта на сървъра, а не действителните PHP кодове, присъстващи във PHP файла. Също така, PHP файловете могат да поддържат други клиентски скриптови езици като CSS и JavaScript.

PHP всъщност може да направи всичко, свързано със скриптове от страна на сървъра. Например, PHP може да:

* Получава данни от формуляри,
* Да генерира динамично съдържание на страници,
* Да работи с бази данни,
* Да създава сесии,
* Да изпраща и получава бисквитки,
* да изпраща имейли и т.н.

В PHP също има много хеш функции, за да криптират данните на потребителя, което прави PHP защитен и надежден за използване като скриптов език от страна на сървъра. Това са някои от способностите на PHP, което го прави подходящ за използване като скриптов език от страна на сървъра.

### 3.2.2.11 „Symfony“ (Технологична рамка)

Symfony е PHP рамка с отворен код , който следва „MVC“ архитектура. Използва се за изграждане на високоефективни сложни уеб приложения. По същество опростява работата на разработчиците и им позволява да създават уеб приложения без монотонно и широко кодиране.

Основната идея на „Symfony Framework“ е да премахне забързаното кодиране и да спести време за развитие. Основава се на философията за създаване на софтуер от потребителите за техните собствени нужди. Разработчиците могат да добавят допълнителни модули, докато продуктът расте. Улеснява живота на програмиста с лесно достъпни компоненти на рамката и конфигурация от висок клас. Рамката осигурява:

* Бързо и лесно разработване на приложения – „Symfony“ изразходва по-малко памет, а рамката позволява на потребителите да създават стабилни и високоефективни приложения.
* Комфорт и удобство – позволява на разработчиците, ползващи рамката, да се съсредоточат повече върху основните и разширени функционалности на приложението, а не върху основните функции. Има и „Symfony“ инструменти за разрешаване на кодиращи грешки и проблеми със сигурността.
* Лесна за употреба и гъвкава - дава пълен контрол върху конфигурацията и други критични характеристики на развитието. Рамката е съвместима с други системи от бази данни. Лесният за потребителя интерфейс с ефективна методология помага за създаването на сложни и динамични приложения с голяма гъвкавост на количеството. Допълнителните функционалности могат лесно да се добавят с множество функции. Той е адаптиран към специфичните изисквания на потребителите.
* Надеждна поддръжка
* Възможност за персонализиране – „Symfony“ предлага страхотни персонализирани функции и функционалности за разработчиците и бизнеса
* Възможност за раширяване - Всичко в „Symfony” се представя като пакет. Всеки пакет има уникална функционалност. Позволява преиползване на пакети. Можете да промените или промените всичко, дори сърцевината на системата, без да конфигурирате отново цялата рамка. Позволява добавяне на функционалности.
* Лесно тестване на програмни фрагменти/функционалности - Улеснява тестването с независима библиотека на „PHP Unit“. Той симулира „HTTP“ заявки и проверява изхода без никакъв скрипт, използвайки инструменти за тестване. Той автоматизира функционалното тестване, което спестява значително време и усилия на разработчиците. Symfony предлага и някои наистина добри инструменти за функционално, поведенческо и единично тестване.

### 3.2.2.12 MySQL

MySQL е подкрепена от „Oracle“ система за управление на релационни бази данни с отворен код, базирана на структуриран език на заявките (SQL). „MySQL“ работи на почти всички платформи, включително „Linux“, „UNIX“ и „Windows“.

SQL е език, чиято основна цел е създаване, обработка, извличане и други операции с данни от релационни системи за управление на бази от данни. Освен стандартния SQL, дефиниран от ANSI и ISO, съществуват още много негови разширения и вариации. Почти всички разработчици предоставят различни вариации и разширения на SQL, които се описват като SQL диалекти. Част от тези разширения са със затворен характер – Oracle PL/SQL и други.

SQL е създаден със специфична цел – заявки за данни в релационна база от данни. Като такъв той е декларативен език за програмиране, а не процедурен. Разширения като PL/SQL намаляват разликата добавяйки процедурни елементи и контролни структури.

**Извличане на данни**

SELECT се използва за извличането на редове от една или повече таблици от базата данни. Това е една от най-често използваната команда. При извличането на данни чрез SELECT е необходимо да бъде описан желания резултат, но без операциите, които трябва да се извършат, за да бъде постигнат той. Често срещани ключови думи, които се използват при SELECT заявка са:

* FROM – използва се за да се определи от кои таблици трябва да бъдат извлечени данните.
* WHERE – за описание на определени условия, които трябва да бъдат изпълнени, за да се смята, че конкретния запис трябва да бъде върнат.
* GROUP BY – служи за групиране на резултата от редове по някакви данни.
* ORDER BY – служи за сортиране на резултата от редове в някакъв ред

**Основни ключови думи са CREATE и DROP:**

* CREATE – създава обект (пр. таблица) в базата данни
* DROP – изтрива съществуващ обект от базата данни.

**Обработка на данни**

Има няколко стандартни групи от запазени думи в SQL, като една от тях е Езика за обработка на данни (DML). Той се използва да добавяне, модифициране и изтриването на такива.

* INSERT – добавя запис в съществуваща таблица.
* UPDATE – модифицира данните на вече съществуващ запис.
* MERGE – комбинира данни от множество таблици.
* TRUNCATE – изтрива всички данни от таблица.
* DELETE – премахва записи от таблица.

.

### 3.2.2.13 HTML (HyperText Markup Language)

„HTML“ е компютърен език, създаден да позволява създаването на уебсайт. След това тези уебсайтове могат да бъдат разглеждани от всеки друг, свързан към Интернет.

Дефиницията на HTML е HyperText Markup Language. HyperText е методът, чрез който се придвижвате в интернет - като кликнете върху специален текст, наречен хипервръзки, който ви отвежда към следващата страница. Фактът, че е хипер просто означава, че не е линеен – дава възможност за препрадки към всяко едно място в Интернет, когато искате, като кликнете върху връзки - няма зададен ред за извършване на неща.Маркирането е това, което HTML маркерите правят на текста вътре в тях. Те го маркират като определен тип текст (например курсив). HTML е език, тъй като има кодови думи и синтаксис като всеки друг език.

**Начин на работа**

HTML се състои от поредица кратки кодове, въведени в текстов файл от автора на сайта - това са маркерите. След това текстът се записва като „html“ файл и се гледа през браузър, като „Internet Explorer“ или „Netscape Navigator“. Този браузър чете файла и превежда текста във видима форма. Можете да бъде използван от всякакви текстови редактори - от рудиментарен до мощен графичен редактор, за да създавате HTML страници. За да създаваме красиви сайтове, използвайки този език, не е нужна връзка с интернет. Езикът ни дава възможност за работа без наличието на интернет вързка.

**Тагове**

Таговете са това, което разграничава нормален текст от HTML код. Самите тагове не се появяват, когато преглеждате страницата си през браузър, но ефектите им се появяват. Най-простите тагове не правят нищо повече от прилагането на форматиране към някакъв текст.

### 3.2.2.14 CSS (Cascading style sheets)

Език за описване на презентацията и стиловете на елементите в един „HTML/XML“ документ. „CSS“ е една от основните технологии, използвани в Уеб, редом с „HTML“ и „JavaScript“. При създаване на „HTML“ страница, съдържанието й се описва с „HTML“ код, а презентацията на това съдържание, тоест как ще изглежда то в браузъра, се описва със „CSS“ код (стил).

Кодът се поставя във файл с разширение „.css“. След това този файл може да се използва във всяка една страница от уеб сайта. Така че стилът ще се намира на едно място и при промяна ще се отразява на всички уеб страници.

**Структура**

Структурата на CSS файла включва название на елемента (селектор), за който ще се приложи стила - в случая това е body, и след това параметрите (свойствата), които се ограждат в големи скоби - { }. Когато в големите скоби се поставят няколко свойства (както е в случая), те се отделят един от друг чрез точка и запетая. Всички параметри, намиращи се в големите скоби {..}, се наричат правило, а то се състои от селектора (като например „body“) и свойствата (като например „background-color“). За свойствата се задават стойности, като някои стойностите се различават една от друга.

**Начин на работа**

Съществуват няколко начина за прилагане на „CSS“ стила:

* Външен стил – Използва се когато трябва да се контролират множество HTML документи, като нужните параметри се задават във външен файл (file.css).
* Вътрешен стил - Съществува и вътрешен за HTML документа стил (Internal Style Sheet), който се използва за да се зададе вида на един отделен HTML документ, като нужните свойства се задават със специалния таг <style> в секцията <head> на HTML страницата.
* Вграден стил - Другият начин за налагане на стил е Inline Styles - вътрешни за HTML таговете стилове. CSS стилът се разполага като атрибут директно в HTML тага например <p>.

**Разиширения**

**SCSS** е препроцесор, който ви позволява да използвате функции, които все още не са част от по-широкия CSS стандарт и осигурява по-добри работни процеси за поддържане на стилове. Дава възможност за намаляне на броя на повторенията и позволява писането на поддържаем код. Той може да използва доректно CSS код, без да има нуждата от преработка.

### 3.2.2.15 Bootstrap

Bootstrap е безплатна колекция от инструменти за създаване на уеб сайтове и уеб приложения. Тя се съдържа от HTML и CSS базирани темплейти за дизайни на типография, форми, бутони, навигации и други компоненти на потребителския интерфейс. В пакета има и JavaScript компоненти, които могат да бъдат използвани по желание. Bootstrap поддържа изграждане на гъвкави(responsive) интерфейси, което означава, че страницата се адаптира динамично към размера на потребителския екран.

### 3.2.2.15 XML(Extensible Markup Language)

Използва се за описание на данните. Стандартът XML е гъвкав начин за създаване на информационни формати и електронно споделяне на структурирани данни чрез Интернет, както и чрез корпоративните мрежи.

XML кода е подобен на HTML. И двете технологии съдържат символи за маркиране, за да опишат съдържанието на страница или файл. HTML описва съдържанието на уеб страниците (главно текстови и графични изображения) само по отношение на начина, по който трябва да се показва и взаимодейства. XML данните са известни като самоописващи се или самоопределящи се, което означава, че структурата на данните е вградена с данните, така че при пристигането на данните няма нужда от предварително изграждане на структурата за съхраняване на данните; динамично се разбира

### 3.2.2.16 Twig

Twig е конструиран във файлове, които имат разширение на файла .twig (известно като изгледи). Той зарежда динамично съдържание (като име на продукт или изображение на страницата на продукта). По-голямата част от изгледа е нормален HTML код, използваме Twig, за да вмъкваме динамично съдържание в страницата, гледана в браузъра на потребителя.

Twig използва комбинация от тагове, обекти ,филтри и функции, за да зареди и покаже това динамично съдържание в необходимия формат.

### 3**.2.2.17 Webpack**

”Webpack” е JavaScript инструмент за пакетиране на модули, който е с отворен код. Webpack ви дава възможността да оптимизирате сервирането на front-end асети като картинки, скриптове, стилове и други. Освен, че взима само информацията, нужна за приложението да визуализира правилно данните си, той също така се грижи и за начина по който се изобразяват данните на на различните браузъри. Някои остарели браузъри имат изменени разбирания за от модерния код на писане, те не знаят как да разчетат някои от новите атрибути на съответните езици. Пример за това е писането на SCSS/CSS.

### NGINX

NGINX е високопроизводителен [уеб сървър](https://bg.wikipedia.org/wiki/%D0%A3%D0%B5%D0%B1_%D1%81%D1%8A%D1%80%D0%B2%D1%8A%D1%80) и прокси. Подобно на други приложения от този вид, архитектурата на nginx е модулна – при компилиране на софтуера се определя кои модули да бъдат вградени в него. Съществуват и над 20 потребителски модула. Той може да се използва като обратен прокси сървър, който прехвърля всички или само определени заявки към други физически сървъри. Крайните сървъри могат се избират от „nginx“ на ротационен принцип, но решенията кой от тях да се използва могат да се взимат и чрез по-сложни алгоритми, благодарение на допълнителни модули.

Често срещана употреба на приложението е за обработване на заявки за статично съдържание и прехвърляне на по-сложните заявки за динамично съдържание към по-сложен уеб сървър, например „Apache“. Въпреки това, „nginx“ има пълна „FastCGI“ поддръжка и може да изпълнява скриптове на всеки език за програмиране, който поддържа този стандарт. Софтуерът може да се използва и като „SMTP“, „POP3“ и „IMAP“ прокси сървър

При определени ситуации, особено при обслужване на заявки за статично съдържание, „nginx“ е по-бърз и заема по-малко ресурси от конкурентния софтуер – „Apache“.

### 3.2.2.19 Symfony Bundle(пакет)

Пакетът е просто структуриран набор от файлове в директория, който реализира една функция. Дава възможност да се създават различни пакети по преценка на разработчкиа или да се използват вече съществуващи такива. Всяка директория съдържа всичко, свързано с тази функция, включително PHP файлове, шаблони, таблици стилове, JavaScript файлове, тестове и всичко друго. Всеки аспект на функция съществува в пакет и всеки елемент живее в пакет.

Symfony работи изцяло с подобни видове пакети, като почти всичко в технологичната рамка е пакет, включително както основната функционалност на рамката, така и кода, написан за даденото приложение.

### Hyper-V

Hyper-V е технологичен инструмент за виртуализация от „Microsoft”, който се предлага в операционна система „Windows 10”. Hyper-V ни позволява да създаваме една или няколко виртуални машини за инсталиране и стартиране на различни ОС на един компютър с „Windows 10“.

**Софтуерни изисквания:**

* Операционни система: Windows 10, Windows 10 Pro, Windows 10 Enterprise или Windows 10 Education.

**Хардуерни изисквания:**

* 64-битов процесор с превод на адреси от второ ниво
* Процесорът трябва да поддържа VM Monitor Mode Extension (VT-c на Intel чипове).
* 4GB RAM памет минимум..

# Глава 4. Разработка на програмните модули за разработка на учебен разпис

## 4.1 Конфигурация на сървърната машина

В основата стои нашият сървър, на който е качен приложението. Сървърът се намира на виртуална машина, което означава, че за него няма значение, физически къде се намира (на компютър в офиса или вкъщи). Това от коеото има нужда е достъп до интернет мрежата. Единствения начин за разработчика да се свърже със сървърната машина и да има правата да конфигурира според нуждите си, е да изгради сигурна и надеждна връзка. Предимствата на подобен подход са:

* Изолирана среда на разработване и работа на сървъра
* Дава възможност за по лесната интеграция на нови разработчици, които искат да бъдат част от разработката на приложението
* Свеждане до минимум на грешките, възникващи при повечето приложения които се разработват, като несъвместимост в операционни системи или различия в изпълнението на приложението при разработка

### 4.1.1 Технологии нужни на разработчика за инициализация на виртуалната машина

За да започнем работата по инициализацията на виртуалната машина. Разрабочика има нуждата да направи някои конфигурации на машината си, както и да изтегли технологиите нужни за инициализация на виртуалната машина. Осъществяването на тази фаза се извършва, чрез извършване на следните последователни стъпки:

1. Изтегляне на технологията Vagrant за Windows 10
2. Инсталиране на Vagrant – след инсталацията, ще имам възможността да ползваме команди свързани с vagrant, във всеки един терминал.
3. Включване на услугата Hyper-V
   1. Отваряне „Контролния панел“ / “Control panel”
   2. Посочваме „Програми“ / “Programs”
   3. В прозореца с “Функции на Windows” / „Windows Features” – проследавяме и отбелязваме Hyper-V както и всички други дъщерни папки
   4. Запазваме промените като изберем „OK”
   5. Ще бъдете подканени да рестартирате компютъра си с цел запазване на промените.
4. След като сте рестартирали, за да проверите дали Vagrant e успешно инсталиран, може да стартирате терминал и да въведете командата: ***vagrant –version****.* Ако терминалът ви върне текст, който съдържа информация за текущата версия на Vagrant, значи, че ще го инсталирали успешно.

### 4.1.2 Инициализация на виртуалната машината

За инициализацията на виртуалната машина, използваме Vagrant, който ни позволява да изградим виртуални машини за кратък период от време, на базата на предварително създаден кофигурационен файл, описващ характеристиките на машината, която ще бъде създадена. Процеса на създаване, при правилното конфигуриране е напълно автоматизиран.

**Конфигуриране:**

1. Създаване на конфигурационен файл на Vagrant - Конфигурационният файл се създава, като стигнем, чрез терминал до желаната от нас директория на проект и въведем командата:
2. ***vagrant ini***

Това ще създаде празен файл, на име „Vagrantfile”

1. Конфигурационният файл контролира създаването на вирталната машина, като задава нужните за сървара ресурси, за да функционира успешно.
2. Нужни конфигурации и методи за приложението
   1. Активиране на SSH протокол, нужен за свързването ни с машината
   2. Проверка за наличието на добавката „Hostmanager”, която позволява автоматичното вписване адреса на новосъздадената виртуална машина в „host” файла на машината на разработчика. Това се прави с цел установяване на локална връзка между машината на разработчика и виртуалната машина.
   3. Настройват се обема на ресурсите, които ще бъдат използвани от сървъра
   4. Синхронизират се директориите и файловете – всичко което се намира в една директория назад от директорията на кофигурационният файл се копира на виртуалната машина.
   5. Задаване адрес на мрежата на виртуалната машина, чрез DHCP за да бъде възможна по-късно, установяване на връзка чрез SSH
   6. Изискват се скриптови файлове които служат за инициализацията и конфигурацията на технологиите, както и установяване на сигурна връзка между двете машини

### 4.1.3 Устанояване на връзка между сървъра и разработчика

Установяването на връзката се осъществява по време на инициализацията на виртуалната машина, след като бъдат извикани скриптовете за автоматично набавяне и конфигуриране на технологиите. Тези скриптове са реализирани с помоща на технологията Ansible, която предлага предварителни „рецепти“ (готово скриптове).

Един от тези скриптове се грижи за осъществяване на сигурна връзка между машината на разработчика и виртуалната машина. Този скрип изгражда SSH връзка, като генерира ключове за двете машини, съдържанието на които е шифриран и единственият начин за дешифриране е като единият съдържа алгоритъма за дешифриране. Той „запознава“ машините, като записва в конфигурационният файл на агента/протокола, дешифриращия алгоритъм. Този алгоритъм е наличен, когато двете страни са си обеменили ключовете. Метода е сигурен, защото самия обмен на ключовете е криптиран и няма начин да бъде „симулиран“ от друго устройство.

### 4.1.4 Набавяне на нужните за разработка на приложението технологии

Набавянето на тези технологии става по време на изграждането на виртуалната машина на която ще бъде сървъра. Отново чрез скриптови файлове, които изтеглят от интернет, качват на машината и конфигурират нужните технологии. Скриптовете за реализацията са написани на езика YAML.

В началото на всеки един скриптов файл трябва да се упомене адреса на машина в която ще бъдат стартирани скриптовете, както и ще бъдат настроени правата на достъп, за да може да се конфигурират технологиите.

**Скриптове:**

1. Първоначален скрип извикан от конфигурационният файл на Vagrant е “requirements.yml” в него се съхраняват технологиите разпоредени в няколко пакета. Тези пакети са свързани със хранилище, което се намери в интернет. За да бъдат свалени и инсталирани, трябва всеки един от тях да му бъде зададен линк(адрес) водещ до хранилището.
2. Извикване на “setup.yml” – Той съдържа в себе си препратки към други два скирпта, които се извикват последователно.
3. Извикване на „services.yml“ – Той съдръжа динамична информация, касаеща конфигурацията на вече свалените технологии от хранилището.
   1. Пример за такава конфигурация на технология е уеб сървъра NGINX, на който трябва да се окаже информация като: на кой порт да слуша, името на сървъра, типа на връзка, оказване на директорията от която ще се зареждат ресурсите (като html документи, снимки и т.н). Скрипа разпознава технологията и замества в конфигурационният и фаил, специфичните настройки зададени от разработчика
   2. Настойка на дата, час и година на виртуалната машина
   3. Съхранява информация за това коя версия на PHP ще бъде ползвана
   4. Извикване на „app.yml” - Той набавя всички технологии и пакети, нужни за успешната работа на проект, използващ Symfony.

### 4.1.5 Конфигурация на технологиите

* Конфигурация на NGINX
* Настройване на версията на PHP
* Конфигурация на правата на свързания със виртуалната машина разработчик
* Инсталация на пакетите, които се изпозлват от Symfony рамката

## 4.2 Конфигурация на средатата за програмиране

### 4.2.1 Избор и настройки на текстов редактор

От изключително значение е изборът на текстов редактор, преди да се започне със изработването на приложение. Текстовият редактор и много важен, защо може понижи времето на разработка на приложението, като се грижи за операции, които често ще се прилагат от разработчика. Такива операции са автоматичното изпращане на файлове към виртуалната машина на която е сървъра, с цел обновяване на приложението. Тази операция, без подходящият текстов редактор е много времеемка.

**PhPStorm:**

Този текстов редактор решава именно този проблем. Той позволява автоматичното запазване на файловете при обновление. За да се изпращат автоматично към сървара, е нужно да се изгради връзка, чрез която, да се изпращат файловете. PhPStorm позволява използването на SSH, което означава, че единственото нещо което ни трябва е да изпозлваме вече генерираният от инициализацията на виртуалната машина, ключ. Той обикновенно се намира в директорията, която съхранява споделените файлове между машината на разработчика и виртуалната машина.

За връка със виртуалната машина от PhPStorm са нужни следните данни:

* Тип на връзката
* Адрес на сървъра
* Съществуващ „потребител“ с администраторски права, намиращ се на сървъва.
* Ключ, който е издаден при инициализацията на виртуалната машина
* Задаване на директорията в която ще се записват/обновяват файловете.

### 4.2.2 Интеграция на Git

За инициализация на Git, трябва преди това да създадем хранилище, което се пази при сървърите им. За целта е нужно да имаме акаунт. След създаването на акаунт, ще имаме възможността за направата на виртуално хранилище. След като създадем такова, ще трябва да направим копие на проекта(в локалното хранилище), което ще се изпрати към хранилището. За да инициализириме локалното хранилище и качим проекта с в Git хранилището, ще са ни нужни следните команди в дадената последователност:

* git clone <път към създаденото хранилище на Git> - тази команда автоматично ще извлече всичката информация която се съхранява в конкретното хранилище, правейки директорията ни, локално хранилище
* git add <име на промениеният файл> ще подготви файловете за изпращане към сървъра
* git commit –m <някакво съобщение> запази състоянието на промените файлове.
* git push – ще изпрати запазеното състояние на файловете към Git хранилището

## 4.3 Инициализация на Symfony проект

Инициализацията често се приема много досадна част от разработката на проект. През повечето време, разработчика трябва ръчно да създава директориите в които ще отседнат файловете, общи настройки или тестови файлове.

Symfony замества всичко това със само една команда:

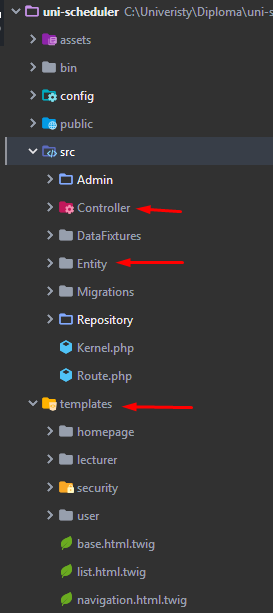
***> composer create-project symfony/website-skeleton <име на проекта>***

Тя създава всички нужни директории и конфигурира всички пакети за направата на едно уеб приложение.

### 4.3.1 Архитектура на приложението

Фиг. 28 Архитектура на директориите.

Разработеното приложение използва MVC архитектура



* Папка “Controller” съхрянва всички контролер обекти от които моделите ни се нуждаят.
* Папка „Entity” съхрянава бизнес логиката или така наречените, модели.
* Папка „templates“ съхрянява изгледите, които се извикват от контролера, при зареждане на страница.
* Папка „Admin” съдържа контролери, които са специфични за администраторският панел. Отделени са, за да може ясно да се разгреничават от контролерите за потребители
* Папка „public“ съдържа файловете които са обработени чрез “Webpack”-а.
* Папка „Repository“ съхранява всичко обекти, които играят роля на хранилища. Те са обекти, които при извикване от контролер, се свързват с базата данни и извличат информация от нея.
* Папка „Migrations“ съдържа файлове които се създават при инициализацията на базата данни. Обикновенно тези файлове съдържат заявка, която при

стартиране, изгражда базата данни или я актуализира.

* Папката „DataFixtures“ съхранява файлове които съдържат в себе си набор от данни които се попълват при инициализацията на базата данни. Извикват се само при поискване от разработчика
* Папката „config” съдържа всички конфигурационни файлове за пакетите, които се използват от приложението. Тя съдържа също ORM на моделите, които служат за изграждане на базата данни.
* Папката „assets” съдържа в себе CSS и JavaScript Файлове, които “Webpack”

„пакетира“

### 4.3.2 Конфигурация на проект

Конфигурацията на проекта до голяма степен се поема от Symfony. При инсталиране на нови пакети, рамката автоматично създава нужните конфиурационни файлове и рядко се налага да се конфигурират допълнително. Всички конфигурационни файлове се намират в директорията „config“.

Допълнителни пакети използвани за проекта:

* webpack – конфигурацията за която не се извършва автоматично
* symfony-sonata-bundle – конфигурацията до голяма степен, но не напълно е автоматична
* doctrine/orm – има нужда от „малка“ ръчната конфигурация
* styleLint – ръчна конфигурация
* security-permissions – ръчна конфигурация

Файлът който съдържа информация за пакетите и техните версии се нарича composer.json. Чрез него се контролират пакетите, версиите им, както и самата версия на Symfony. При промяна на този файл, в зависимост от това дали искаме да бъде инсталиран или премахнат даден пакет, може да извикаме командата „***composer install”*** или ***„composer update”.*** Тя ще вземе под внимание промените по файла и автоматично ще изтегли/изтрие желание пакет, като конфигурира или разконфигурира конкретния пакет.

**Webpack**

Конфигурационният файл е именуван „webpack.config.js“. В себе си съдържа код, който посочва:

* файловете които ще обработва
* начина на обработка
* къде ще записва обработените файлове.

**StyleLint**

Използва се по време на работата на Webpack. Той има за цел да налага правила, по които да се пише CSS код. Правила като отстояние на кода, дължина на селектори, подреждане на атритбути по азбучен ред и т.н. Тези правила се описват във файла “.stylelintrc”. Правилата се задават от разработчика на приложението.

**Symfony-sonata-bundle:**

Това е пакет, който ни придоставя почти готов администраторски панел. Това което ни придоставя е автоматичното изобразяване на създадените от нас модели, които искаме да се визуализират в администраторският панел. Пакета да възможност да иползваме готови дизайни. Той използва предварително създадени стилове от Bootstrap. Конфигурацията на който много лесна, поради факта, че Symfony се заема с по-голямата част. Единственото нещо което трябва да се направи е да се добави пътят на администраторските контролери в конфигурационния файл на “symfony-sonata-bundle” (services\_admin.yml)

**Doctrine/orm:**

Този пакет е пряко свързан със топологията на Doctrine. Конфигурацията нужна да се направи за да бъде успешно ползването на пакета е да се избере типа на базата данни, версията и, пътят до ORM файловете, типът на ORM файла (xml,yaml,php), както и пътят до папката със самите модели.

**Security:**

Този пакет се конфигурира по желание на разработчика. Служи за ограничаване достъпа според ролята до дадени страници, методи за криптиране на данните като пароли, настройки на данните които се изпращат от формата за аутентикиране на потребител.

## 4.4 Реализация на приложението

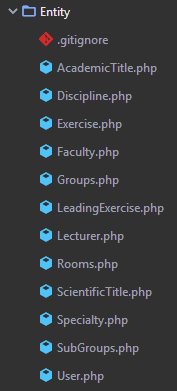
### 4.4.1 Израждане на моделните обекти

Обектите представляват класове, които имат в себе си частни член променливи. Тези променливи не са достъпни за външни обекти. Това предава сугорност на данните, енкапсулира ги. Те са така наречената „бизнес логика“ на приложението. Иползват се за изграждането на базата данни, като имат пряка връзка със ORM файловете. В Symfony, тези класове, се наричат “Entity”, поради тази причина и директорията в която се намират се нарича „Entity”.

В моделите, се счита за лоша практика, ако имат във себе си функции, които променят по някакъв начин данните на обекта.

Фиг. 29 Показва моделите

**User.php** – използва пакет „Security“ което му дава достъп до методи като хеширане на пароли и потребителски данни.



Той има следните полета:

* + - * id – служи за идентификационен номер на обекта
      * username – потребителско име
      * email – имейл
      * phone – телефонен номер
      * lecturer – референция към обект преподаватели
      * password – парола
      * roles – роля на потребителя

Той има следните методи:

* construct() – служи за задаване на първоначална роля на всеки един нов потребител.
* set() – всяко едно поле има подобно поле, което служи за задаване стойност на полето на обекта (аналогично и за другите полета)
* get() – служи за визуализация/взимане на вече запазените стойност в обекта (аналогично и за другите полета)
* addRole() – служи за добавяне на роли. Възможно е един user да има повече от една роля
* \_\_toString() – това е „магически“ метод, който се изпълнява при извикването на обекта. Когато даден обект се извика/инстанцира, този метод му задава презентационен вид, който зависи от полетата които връща метода.
* serialize() – служи за криптиране на паролата, идентификационният номер и потребителското име
* unserialize() – служи за декриптиране на паролата, идентификационният номер и потребителското име, като приема алгоритъма който е използван за криптирането им

**Lecturer.php:**

Използва пакета “ArrayCollection” и „Collection“.

Публични константни променливи:

* public const USER\_ROLE\_LECTURER = 'ROLE\_LECTURER';
* public const USER\_ROLE\_ADMIN = 'ROLE\_ADMIN';

Той има следните полета:

* id – служи за идентификационен номер на обекта
* firstName – Име
* middleName - Презиме
* lastName – Фамилия
* email - имейл
* room – номер на стаята на преподавателя
* leadingExercises – референция към обект Водещи занятия
* academicTitle – академични титли
* scientificTitle – научни титли
* contract - договор
* user – референция към обект Потребител
* estHours – брой полагащи му се часове

Той има следните методи:

* construct () – служи за инстанциране на обект от тип „колекция от масиви“ с цел избягване на грешка при създаване на обекта.
* set() – всяко едно поле има подобно поле, което служи за задаване стойност на полето на обекта (аналогично и за другите полета)
* get() – служи за визуализация/взимане на вече запазените стойност в обекта (аналогично и за другите полета)
* addLeadingExercises() – Служи за добавяне на обекти от тип Водещи Занятия в масив
* \_\_toString () – използва се за същите цели както при обекта Потребител.

**Discipline.php**

Той има следните полета:

* id – служи за идентификационен номер на обекта
* name – име на дисциплината
* abbreviation - абревиатура на дисциплината
* lecturesCount – брой лекции за семестъра
* semExercisesCount – брой семинарни упражнения за семестъра
* labExercisesCount – брой лабораторни упражнения за семестъра

Той има следните методи:

* set() – всяко едно поле има подобно поле, което служи за задаване стойност на полето на обекта (аналогично и за другите полета)
* get() – служи за визуализация/взимане на вече запазените стойност в обекта (аналогично и за другите полета)
* \_\_toString ()

**Specialty.php**

Той има следните полета:

* id – служи за идентификационен номер на обекта
* name – име на специалността
* abbreviation – абревиатура
* startYear – година на започване
* endYear – година на приключване

Има сходни методи като на останалите класове

**AcademicTitle.php**

Има следните полета:

* id – служи за идентификационен номер на обекта
* name – име на академичната титла
* abbreviation – абревиатура

Има сходни методи като на останалите класове

**ScientificTitle.php**

Има следните полета:

* id – служи за идентификационен номер на обекта
* name – име на научната титла
* abbreviation – абревиатура

Има сходни методи като на останалите класове

**Faculty.php**

Има следните полета:

* id – служи за идентификационен номер на обекта
* name – име на факултета
* abbreviation – абревиатура
* rooms – рефернция към обект Стаи

Има сходни методи като на останалите класове

**Exercise.php**

Публични константни променливи:

* public const EXERCISE\_SEMESTER\_SPLIT\_WINTER = 'зимен'
* public const EXERCISE\_SEMESTER\_SPLIT\_SUMMER = 'летен'

Има следните полета:

* id – служи за идентификационен номер на обекта
* subGroups – референция към обект Под Групи
* rooms – референция към обек Стаи
* dayName – име на ден от седмицата
* start – начало на занятието
* end – край на занятието
* leadingExercise – референция към обект Водещи Занятия
* semesterSplit – тип на семестъра ( Зимен/Летен )

**LeadingExercises.php**

Публични константни променливи:

* public const LEADING\_EXERCISE\_TYPE\_SEMINAR = 'семинарно';
* public const LEADING\_EXERCISE\_TYPE\_LABORATORY = 'лабораторно';
* public const LEADING\_EXERCISE\_TYPE\_LECTURE = 'лекция';
* public const LEADING\_EXERCISE\_TYPE\_COURSE = 'курсова';

Има следните полета:

* id – служи за идентификационен номер на обекта
* specialty – референция към обект Специалност
* discipline - референция към обект Дисциплина
* lecturer - референция към обект Преподавател
* exercise - референция към обект Занятие
* exerciseType – тип на занятието

Има сходни методи като на останалите класове

**Groups.php**

Публични константни променливи:

* public const GROUPS\_EDUCATION\_TYPE\_REGULAR = 'Редовно';
* public const GROUPS\_EDUCATION\_TYPE\_ABSENTIA = 'Задочно';

Има следните полета:

* id – служи за идентификационен номер на обекта
* name – име на групата
* subGroups- референция към обект Под Групи
* specialty - референция към обект Специалности
* studentNumber – брой студенти
* educationType – форма на обучение

Има сходни методи като на останалите класове

**SubGroups.php**

Има следните полета

* id – служи за идентификационен номер на обекта
* name – име на под групата
* studentNumber – брой студенти
* groups - референция към обект Групи

Има сходни методи като на останалите класове

**Rooms.php**

Има следните полета

* id – служи за идентификационен номер на обекта
* roomNumber – номер на стая
* faculty – референция към обек Факултет
* seats – работни станции

Има сходни методи като на останалите класове

### 4.4.2 Изграждане на базата данни

Изграждането на базата данни става с помоща ORM файловете които са пряко свързани със моделите. Тези файлове са описани с помоща на XML. Всеки файл трябва да описва модела който ще свързва. Всяко едно поле в ORM-а трябва да е реално, ако не е, ще възникне грешка при изграждането на дадената таблица.

Изграждането на базата данни става с помоща на командата, която Symfony ни предоставя: ***php bin/console doctrine:schema:update***

Видове тагове:

* <field name=”<име на поле” type=“тип на полето“/>
* <one-to-one> - тип на релацията към реферираният обект.
  + field – описващ името на полето
  + target-entity – касаещ даден обект
  + join-column – задължителни атрубити са “name” , „referenced-column-name“. Служи за сливане на таблици
* <one-to-many> - тип на релацията към реферираният обект
  + field
  + target-entity
  + mapped-by
* <many-to-one> - тип на релацията към реферираният обект
  + field
  + target-entity
* <many-to-many> - тип на релацията към реферираният обект
  + field
  + target-entity

**Exercise.orm.xml**

<entity name="App\Entity\Exercise">

<id name="id" type="integer" column="id">

<generator strategy="AUTO"/>

</id>

<many-to-one field="rooms" target-entity="Rooms"/>

<many-to-one field="leadingExercise" target-entity="LeadingExercise" inversed-by="Exercise">

<join-column name="leading\_exercise\_id" referenced-column-name="id" />

</many-to-one>

<many-to-one field="subGroups" target-entity="SubGroups"/>

<field name="start"/>

<field name="end"/>

<field name="dayName"/>

<field name="semesterSplit"/>

</entity>

**User.orm.xml**

<entity name="App\Entity\Specialty">

<id name="id" type="integer" column="id">

<generator strategy="AUTO"/>

</id>

<field name="name"/>

<field name="abbreviation"/>

<field name="startYear" type="integer"/>

<field name="endYear" type="integer"/>

</entity>

**LeadingExercise.orm.xml**

<entity name="App\Entity\LeadingExercise">

<id name="id" type="integer" column="id">

<generator strategy="AUTO"/>

</id>

<many-to-one field="lecturer" target-entity="Lecturer" inversed- by="leadingExercises">

<join-column name="lecturer\_id" referenced-column-name="id" />

</many-to-one>

<many-to-one field="specialty" target-entity="Specialty"/>

<many-to-one field="discipline" target-entity="Discipline"/>

<field name="exerciseType"/>

<one-to-many field="exercise" target-entity="Exercise" mapped-by="leadingExercise"/>

</entity>

**Rooms.orm.xml**

<entity name="App\Entity\Rooms">

<id name="id" type="integer" column="id">

<generator strategy="AUTO"/>

</id>

<field name="roomNumber" type="integer"/>

<many-to-one field="faculty" target-entity="App\Entity\Faculty"/>

<field name="capacity" type="integer"/>

<field name="seats" type="integer"/>

</entity>

**Discipline.orm.xml**

<entity name="App\Entity\Discipline">

<id name="id" type="integer" column="id">

<generator strategy="AUTO"/>

</id>

<field name="name" unique="true"/>

<field name="abbreviation" unique="true"/>

<field name="lecturesCount"/>

<field name="semExercisesCount"/>

<field name="labExercisesCount"/>

</entity>

**Specialty.orm.xml**

<entity name="App\Entity\Specialty">

<id name="id" type="integer" column="id">

<generator strategy="AUTO"/>

</id>

<field name="name"/>

<field name="abbreviation"/>

<field name="startYear" type="integer"/>

<field name="endYear" type="integer"/>

</entity>

**AcademicTitle.orm.xml**

<entity name="App\Entity\AcademicTitle">

<id name="id" type="integer" column="id">

<generator strategy="AUTO"/>

</id>

<field name="name"/>

<field name="abbreviation"/>

</entity>

**ScientificTitle.orm.xml**

<entity name="App\Entity\ScientificTitle">

<id name="id" type="integer" column="id">

<generator strategy="AUTO"/>

</id>

<field name="name"/>

<field name="abbreviation"/>

</entity>

**Faculty.orm.xml**

<entity name="App\Entity\Faculty">

<id name="id" type="integer" column="id">

<generator strategy="AUTO"/>

</id>

<field name="name"/>

<field name="abbreviation"/>

<one-to-many field="rooms" target-entity="Rooms" mapped-by="faculty"/>

</entity>

**SubGroups.orm.xml**

<entity name="App\Entity\SubGroups">

<id name="id" type="integer" column="id">

<generator strategy="AUTO"/>

</id>

<field name="name"/>

<field name="studentNumber" type="integer"/>

<many-to-one field="groups" target-entity="Groups" inversed-by="subGroups">

<join-column name="groups\_id" referenced-column-name="id" />

</many-to-one>

</entity>

**Lecturer.orm.xml**

<entity name="App\Entity\Lecturer">

<id name="id" type="integer" column="id">

<generator strategy="AUTO"/>

</id>

<field name="firstName"/>

<field name="middleName"/>

<field name="lastName"/>

<one-to-one field="user" target-entity="User" mapped-by="lecturer">

<cascade>

<cascade-all/>

</cascade>

<join-column name="user\_id" referenced-column-name="id" on-delete="CASCADE"/>

</one-to-one>

<one-to-many field="leadingExercises" target-entity="LeadingExercise" mapped-by="lecturer">

<cascade>

<cascade-all/>

</cascade>

</one-to-many>

<many-to-one field="academicTitle" target-entity="AcademicTitle"/>

<many-to-many field="scientificTitle" target-entity="ScientificTitle"/>

<field name="room"/>

<field name="contract"/>

<field name="estHours" nullable="true"/>

</entity>

**Groups.orm.xml**

<entity name="App\Entity\Groups">

<id name="id" type="integer" column="id">

<generator strategy="AUTO"/>

</id>

<field name="name"/>

<one-to-many field="subGroups" target-entity="SubGroups" mapped- by="groups"/>

<many-to-one field="specialty" target-entity="Specialty"/>

<field name="studentNumber" type="integer"/>

<field name="educationType"/>

</entity>

### 4.4.3 Изграждане на клиент-контролери

**Начин за извикване на контролер**

Контролера се активира и извършва своето предназначение, когато бъде извикан от клиент в сайта. Това става когато клиента се опитва да достигне някоя страница. Обикновенно страницата си има наименование като [***http://www.unischeduler.local/***](http://www.unischeduler.local/)

Тогава контролера, отговарящ за тази страница се активира и дава възможност на потребителя да наслади на визуализираната от контролера страница

**Задаване на пътя до контролера**

/\*\*

\* @App\Route("/", name="homepage")

\*/

**Основни методи:**

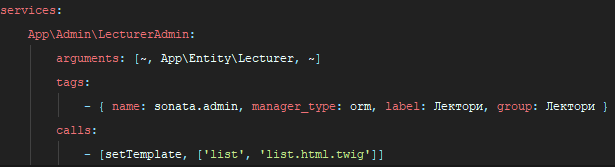
* \_\_construct() – инициализира хранищен обект, който съдържа информация от базата данни
* \_\_invoke() - Методът ще бъде извикан, когато се извика обект по начин на извикване на функция.
* render() – Визуализира изглед, като му се даде пътя до този изглед, и параметри с които излгеда може да работи(като например обект от базата данни)

**Използвани контролери:**

* HomepageController
  + Път до контролера - @App\Route("/", name="homepage")
  + Цел – визуализира изгледа за „началната страница“, като му подава информация от хранилището на този контролер.
* SecurityController
  + Път до контролера - @Аpp\Route("/login", name="login")
  + Цел – визуализира изгледа за вписване на потребител.
* LecturerListController
  + Път до контролера - @App\Route("/lecturer/list", name="list\_lecturer")
  + Цел - визуализира изгледа за списък на лекторите,като му подава информация от хранилището на лекторите.
* LecturerShowController
  + Път до контролера - @App\Route("/lecturer/{id}", name="show\_lecturer", methods="GET")
  + Цел - визуализира изгледа за конкретен лектор, като му подава информация от хранилището на лекторите.
* ShowUserController
  + Път до контролера - @App\Route("/profile", name="show\_user\_profile", methods="GET")
  + Цел - визуализира изгледа за потребител

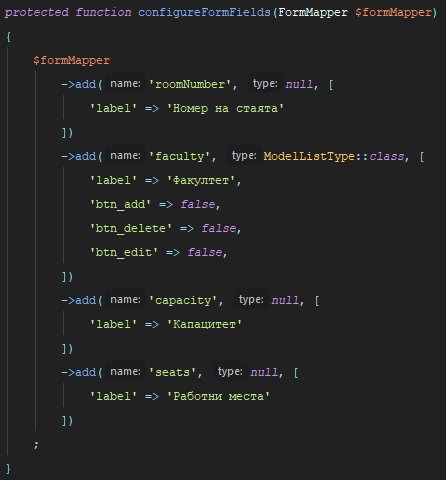
### 4.4.4 Изграждане на админ-контролери

Админ контролерите се обработват с помоща на „symfony-sonata-bundle”. Достъпа до тези контролери, се конфигурира автоматично от пакета. При създаването на един контролер, той трябва да бъде регистриран във конфигурационният файл на пакета. По този начин symfony разбира, че този обект е част от конкретният пакет. След като контролера бива регистраран във „services\_admin.yaml”. Той вече може да бъде достигнат от администратора.



Фиг. 30 На снимката е показана примерна конфигурация на админ контролер за преподаватели

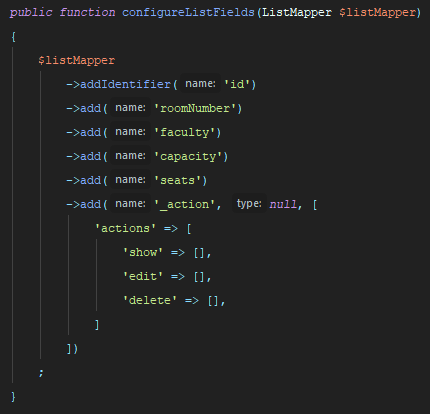
**Основни методи:**



* configureFormFields() – конфигурира полетата от съответният модел на конкретният админ контролер.
  + Всяко поле трябва да бъде конфигурирано по различен начин, поради типът на полето. Ако е обект, най-често се използва ModelListType тип на полето

Фиг.31 Показана е примерна конфигурация на полетата за един админ контролер

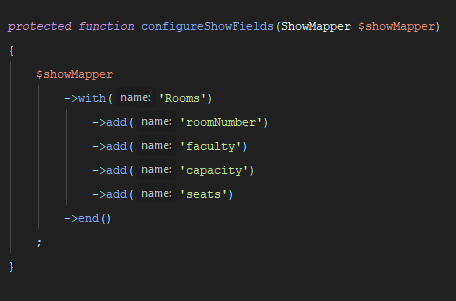
* configureListFields()



* + Тази част от контролера се извиква, когато искаме да видим всички записи от базата данни. Обикновенно пътят до този контролер се конфигурира автоматично, затова не е нужна допълнителна конфигурация

Фиг. 32 Показана е конфигурация на админ контролер за показване на конкретни полета на съществуващ обект от базата.

* configureShowFields()



* + Тази част от контролера се извиква, когато искаме да данните за конкретен обект.

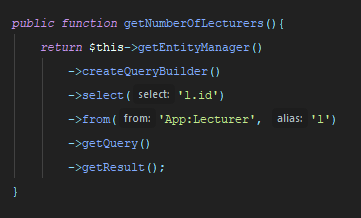
Фиг. 33 Показана е конфигурация на админ контролер за показване на конкретни полета на съществуващ обект от базата.

### 4.4.5 Изграждане обекти на хранилища

Използването на хранилища за всеки контролер е много добра практика, защото по този начин държим контролерите си чисти, без да има написани заявки. За да може един контролер да подаде обект от базата данни, първо трябва да се свърже с нея. Хранилищата са тази връзка. Те са обекти, които в себе си имат методи, които представляват отделна заявка към базата данни.

**LecturerRepository**

* Този метод представлява заявка към базата данни която има за цел да намери и извлече преподавател с конкретен идентификационен номер(id)

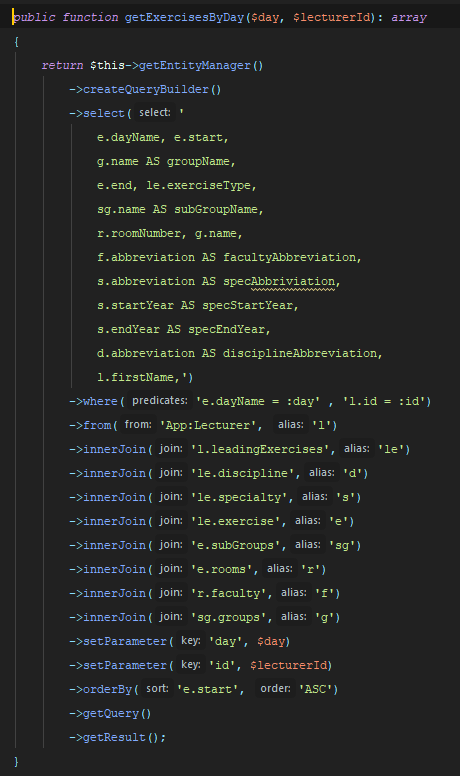


* Използва се от HomepageController с цел преброяване на преподавателите в базата данни и изобразяване на резултата в изгледа.

Фиг. 34 Показана е снимка на метод от

хранилищен обект

* Този метод има за цел да да намери всички записи за преподаватели, които имат занятия в даден ден от седмицата



* + Той един от най-важните за приложение метод. Благодарение на него, се изготвят седмичните учебни разписания за преподавателите. Той се използва от ShowController.

Фиг. 35 Показана е една от най-важните заявки за извличане на информация за учебните разписания за конкретен преподавател

### 4.4.6 Създаване на изгледи

Всеки изглед отговаря на конкретен контролер. Излгедите не са задължени да бъдат извикани от контролер. Те могат да извикват или да разширяват други изгледи.

**Изгледи:**

* Начална страница
  + Съхранява се в -templates/homepage/index.html.twig
  + Извиква се от HomepageController
  + Подават му се като параметри – броят на преподавателите в базата данни
  + Разширява базовият изглед „base.html.twig”
* Форма за вписване на потребителите
  + Съхранява се в – templates/user/login.html.twig
  + Разширява базовият изглед „base.html.twig”
  + Ивиква SecurityController
* Навигация/меню
  + Съхранява се в – templates/navigation.html.twig
  + Позволява навигацията между страниците
  + Съдържа проверка за вписан потребител, което на базата на тази проверка, показва различни елементи от менюто, водещи до съответните страници.
* Страница на всички преподаватели
  + Съхранява се в – templates/lecturer/list.html.twig
  + Извиква се от Lecturer/ListController
  + Подават му се като параметри, взетите от базата данни обекти за преподаватели
* Страница за конкретен преподавател
  + Съхранява се в – templates/lecturer/show.html.twig
  + Извиква се от Lecturer/ShowController
  + Подават му се като параметри разписания на учебните занятия по дни от семицата за конкретен преподавател, подредени по часове във възходящ ред.

### 4.4.7 Налагане на правила/валидации

Валидациите на дадените обекти във симфони става със помоща на пакета „validator”. Това което той предоставя е описване в XML файл, всички полета на даден обект, които бихме искали да бъдат валидирани. Използваните ограничения, които този пакет преглада са:

* <constraint name="Length">
  + <option name="min">
  + <option name="max">
* < constraint name=” Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity”>
  + <option name="fields"> - засягащо се поле