

ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

ДИПЛОМНА РАБОТА

Тема: Социална мрежа за организиране и управление на събития

Дипломант:

Карина Козарова

Научен ръководител:

Красимир Николов

СОФИЯ

2018



**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ
СИСТЕМИ КЪМ ТЕХНИЧЕСКИ
УНИВЕРСИТЕТ - СОФИЯ**

Дата на заданието: 06.11.2018 г.

Утвърждавам:.....

Дата на предаване: 06.02.2019 г.

/проф. д-р инж. Т. Василева/

ЗАДАНИЕ

за дипломна работа

на ученика Карина Красиминова Козарова от 12 А клас

1. Тема: Социална мрежа за организиране и управление на събития

2. Изисквания:

2.1 Могат да се създават акаунти, които могат да създават събития.

2.2 Събитията имат различна видимост в зависимост от настройки по тях.

2.3 Създателят на събитието може да добавя други потребители като част от организационния екип на събитието.

2.4 Хората от екипа могат да си добавят задачи за вършене към самото събитие, които са видими само от тях.

2.5 Може да се добавя потребител, който да свърши дадена задача, и статус на задачата.

3. Съдържание

3.1 Обзор

3.2 Същинска част

3.3 Приложение

Дипломант :.....

/Карина Козарова/

Ръководител:.....

/Красимир Николов/

Директор:.....

/доц. д-р инж. Ст. Стефанова/

Мнение на научния ръководител

Темата на дипломната работа изискваше от Карина още от самото начало да учи и опита на практика много нови за нея концепции и технологии, за да може да ги използва в реализацията. Това включваше научаване на нови програмни езици (Python и JavaScript), работа с Django framework, Django REST Framework, нови шаблони за програмиране, планиране и изпълнение на проект за изработка на приложение и др.

Тя анализира проблемната област, планира обхвата от функционалности, обема работа и сложността ѝ. Тя изработи приложението в очаквания срок и с очакваното качество. Добро впечатление остави начинът и на работа и усилията, които положи през последните 3 месеца преди датата на предаване. Кодът, разработен от нея, спазва добри стандарти за качество и в него тя е приложила успешно и правилно шаблони за програмиране. Карина проявяваше интерес към процеса на разработване на софтуерни проекти като приложи някои от практиките в реализацията на проекта. Писането на дипломната работа беше прекрасна възможност за нея да вникне и опита реалистично разработката на софтуер и Карина я използва пълноценно!

Дипломантът се справи успешно не само с оформяне на концепцията и разработката на софтуерното приложение, но и със самата дипломна работа!

Научен ръководител:

/Красимир Николов/

Увод

В настоящата дипломна работа се има за цел да се разработи уеб приложение, което да предоставя възможност на потребителите да създават и управляват събития. По този начин освен социална част (присъединяване към различни мероприятия), потребителите могат и да управляват прогреса си по задачите за различните събития.

Организирането на едно събитие е трудна и стресираща задача. Именно по този повод Лизи Гаскин, директор в *Right Angle Events*, споделя, че нещото, което е ключово за работата на организаторите на събития е, че често им се налага да следят за изпълнението на множеството аспекти при създаването на събитие¹ - като се започне от това дали е поръчана достатъчно храна за гостите, до това кои ще са посетителите на събитието, та дори и доставката на всеки един продукт, нужен, за да бъде събитието успешно.

Именно цялото това “жонглиране” на задачите се цели да бъде олекотено в настоящата дипломна работа - трябва да бъде създадена социална мрежа, чрез която освен да можем да намерим събития, подходящи за нас, да имаме възможността и да менажираме събитията ни.

¹ 9 Event Planners Share Their Biggest Frustrations

Източник - <https://www.eventbrite.co.uk/blog/event-planning-frustrations-ds00/>

Това може да се постигне чрез съчетаване на функционалности за намиране и присъединяване към събития, както и създаване на задачи по събития, разпределянето им на определени хора от екипа, отговорни за провеждането и предварителната организация на събитието, и следенето на прогреса по всяка една задача. По този начин освен да водят комуникацията си с посетителите, организаторите ще имат възможността да водят и комуникацията помежду си в рамките на уеб приложението.

Глава Първа

Методи и технологии за реализиране на уеб приложения

1.1 Уеб приложение, уеб сайт, уеб страница

1.1.1 Уеб приложение

В днешно време един от най-използваните типове софтуер са именно уеб приложенията, защото за да бъдат използвани, обикновено единственото нужно е устройство с уеб браузър и връзка с Интернет, което в 21-ви век е сравнително лесно за постигане откъм изисквания. Почти всеки потребител разполага с поне едно устройство, което може да се справи с тези изисквания.

Уеб приложението е компютърна програма, която използва уеб браузъри и уеб технология за изпълнение на задачи през интернет връзка. Уеб приложенията използват комбинация от скриптове от страна на сървъра, за да обработват съхранението и извличането на информацията и скриптове от страна на клиента, за да предоставят информация на потребителите.²

² What is a web application?

Източник: <https://www.maxcdn.com/one/visual-glossary/web-application/>

Скриптовете за уебсайт приложенията се изпълняват на едно от две възможни места - client side (така нареченият front-end), или server side (така нареченият back-end).

Клиентът на един уеб сайт на практика е уеб браузърът, чрез който потребителят гледа приложението.

Back-end-ът на едно уеб приложение е сървърът, на който е хостнато самото приложение.

Повечето сайтове използват и client side, и server side езици. Макар че има неща, които могат да се правят и от страната на сървъра, и от страната на клиента, някои неща могат да се правят само и единствено от страната на сървъра или само от страната на клиента.

Най-често фронтенд скриптове се използват за всичко, за което е нужно взаимодействие с потребителя. Бекенд скриптовете от своя страна се използват за всичко, при което е нужно динамично зареждане на информация като например информация, че потребителят успешно се е логнал.³ След изпълняването на дадената команда на бекенд частта, резултатите се предават на фронтенда и се визуализират на екрана.

1.1.2 Уеб сайт

Уеб сайтът е колекция от страници, групирани заедно и обикновено свързани една с друга по различни начини. Или казано

³ Client Side vs. Server Side

Източник: <https://www.codeconquest.com/website/client-side-vs-server-side/>

по-просто - уеб сайтът е съвкупност от уеб страници с връзки помежду им. Често уеб сайтът е наричан накратко просто сайт.

Уеб сайтовете се делят основно на 2 вида:

- Статични уеб сайтове - уеб сайтове, които не правят обработка на информация и не позволяват промяна на съдържанието на уеб страниците.
- Динамични уеб сайтове - уеб сайтове, които правят обработка на информация и позволяват промяна на съдържанието им.

Повечето по-сложни сайтове са именно динамични - например събират информация от потребителя, има акаунти и в зависимост от това дали си логнат потребител или не имаш различни права на достъп, или например има различно съдържание.

1.1.3 Уеб страница

Уеб страницата е документ, който може да бъде рендериран в уеб браузър (например *Firefox*, *Google Chrome*, *Safari* и т.н.) Най-често използвания формат за уеб страници е *Hypertext Markup Language (HTML)*.

Уеб страниците се делят основно на 2 вида според съдържанието си:

- Статични уеб страници - това са страници, чието съдържание се определя предварително и се съхранява в този вид на уеб сървъра.
- Динамични уеб страници - страници, чието съдържание се създава в зависимост от състоянието на въведената информация от потребители, промяна в базата данни и т.н. ⁴

1.2 Начини за разработка на уеб приложение, frameworks, MVC

Разработването на уеб приложение е възможно да се осъществи и чрез използване на *framework*(софтуерна рамка), и без. Освен използването на софтуерна рамка, има и други решения като използване на *CMS (Content Management System)* и техните модули, както и професионални решения като *CRM, e-commerce solutions* и т.н. ⁵

1.2.1 Софтуерна рамка

В повечето случаи използването на софтуерна рамка доста улеснява работата на програмиста, предоставяйки готови функционалности като библиотеки (колекция от подпрограми, които се използват за разработка на софтуер), *API-та (Application Programming Interface)*, което всъщност е софтуерен посредник, който позволява на

⁴ What is the difference between Dynamic and Static Content

Източник: <https://vwo.com/knowledge/what-is-the-difference-between-dynamic-and-static-content/>

⁵ When should I use a framework?

Източник: <https://symfony.com/when-use-a-framework>

две приложения да разговарят помежду си)⁶ и т.н.⁷ Повечето уеб софтуерни рамки предоставят библиотеки за улеснен достъп до базата данни, шаблонни рамки и управление на сесиите.

Някои от по-известните софтуерни рамки за уеб приложения са *Angular*, *Ruby on Rails*, *MeteorJS*, *Django*, *Laravel* и т.н.⁸ Основната разлика между всяка една от тези софтуерни рамки е езикът, който се ползва - например при *Ruby on Rails* се използва *Ruby*, в *Laravel* - *PHP*, а в *Django* - *Python*.

1.2.2 Model-View-Controller

Модел-Изглед-Контролер (*Model-View-Controller* или *MVC*) е архитектурен шаблон за дизайн при уеб приложенията, основаващ се на разделянето на бизнес логиката от графичния интерфейс и данните в дадено приложение.

MVC като цяло е доста популярен начин за организиране на кода на уеб приложенията. Главната идея зад *MVC* е, че всяка секция от кода има определена и различна от останалите функция. Част от кода държи информацията на приложението, друга част държи дизайна и

⁶ What is an API? (Application Programming Interface)

Източник: <https://www.mulesoft.com/resources/api/what-is-an-api>

⁷ Core Features of Web Application Frameworks

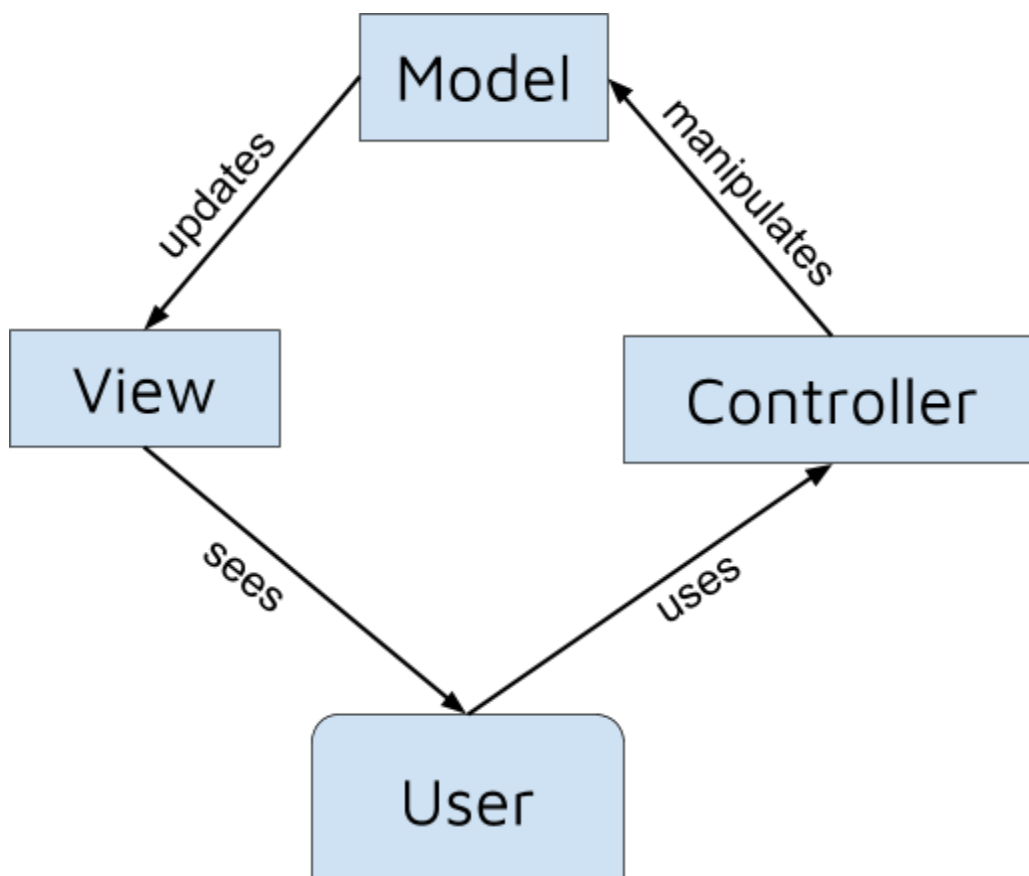
Източник: <https://www.upwork.com/hiring/development/understanding-software-frameworks/>

⁸The Top Web Development Frameworks in 2018

Източник: <https://expertise.jetruby.com/the-top-web-development-frameworks-in-2018-b31dc7263875>

изгледите, друга организира как точно функционира приложението. По този начин мисленето за, разглеждането на кода и споделянето на приложението с други хора е много по-лесно и по-чисто, защото всичко е разделено на компоненти. Моделите в *MVC* обикновено отразяват неща от реалния свят.

Кодът за моделите може да съдържа необработени данни или да дефинира основните компоненти на приложението. Кодът за изгледите (*views*) е съставен от всички функции, които директно взаимодействат с потребителя - тук се включва дизайнът на приложението и се дефинира какво потребителят вижда и с какво взаимодейства. Контролерът действа като връзка между модела и изгледа, като получава потребителски данни и решава какво да прави с тях. Взаимодействието им е онагледено на **Фиг. 1.2.1.**



Фиг. 1.2.1 MVC взаимодействие

1.2.2.1 MVT

MVT (Model-View-Template) също е архитектурен шаблон за дизайн при уеб приложенията и наподобява MVC. MVT също е колекция от три важни компонента - модел, изглед и шаблон. Компонентите изглед и модел имат аналогична роля като при MVC. Тук обаче на мястото на Контролера имаме Шаблон.

Шаблонът е слой за презентация, който напълно обработва частта на потребителския интерфейс. Шаблонът е HTML файл, смесен с Django Template Language (DTL).

1.2.3 API

Интерфейсът за програмиране на приложения (*Application programming interface* или *API*) е набор от дефиниции на подпрограми, комуникационни протоколи и инструменти за изграждане на софтуер. В общи линии, това е набор от ясно определени методи за комуникация между различните компоненти. Едно добро *API* улеснява разработването на компютърна програма, предоставяйки всички заявки, които могат да бъдат ползвани.

Точно както графичният потребителски интерфейс (*GUI*) улеснява използването на софтуер от хората, така и приложните програмни интерфейси улесняват разработчиците. Приложните програмни интерфейси (*API*-та) дават достъп до определен набор от функции (обикновено чрез тях се изпълняват заявки за извличане на данни от друго приложение) и ясно дефинират точно как дадена програма ще взаимодейства с друга като така се спестяват ценни ресурси. Използването на *API* опростява програмирането като абстрахира основното изпълнение и дава достъп до обекти или действия, от които разработчикът се нуждае.⁹

⁹ Measuring API Usability

Източник: <http://www.drdoobs.com/windows/measuring-api-usability/184405654>

1.3 Текстови редактори. Интегрирани среди за програмиране.

В наши дни за писане на код се използват основно текстови редактори или интегрирани среди за програмиране. Това дали за разработването на един продукт ще се използва интегрирана среда за програмиране или най-обикновен текстов редактор е личен избор на всеки един програмист.

Текстовите редактори се използват за създаване и изменение на текстови файлове, представляващи кодът на една програма. Някои текстови редактори поддържат функции, улесняващи програмирането на съответния език: цветово различаване на служебни думи, функции, класове, автоматична табулация и т.н.

Друг често срещан избор при програмистите за редактиране на кода е използването на интегрирана среда за програмиране - *IDE*. В повечето случаи *IDE*-то е текстов редактор с добавени допълнителни функционалности, които подпомагат писането на код. Много от тях ще подчертаят някои синтактични грешки или ще ни помогнат да допишем кода си, използвайки функции за автоматично довършване.

Почти всяко *IDE* има някакъв автоматизатор на компилацията, който компилира кода и изгражда изпълнима програма, добавяйки всички необходими библиотеки. Една от най-често използваните функции на повечето *IDE*-та е дебъгерът. Обикновено дебъгерът

представлява набор от инструменти, които помагат на разработчиците да откриват грешки в кода.¹⁰

1.3.1 Visual Studio Code

Visual Studio Code е интегрирана среда за програмиране, разработена от *Microsoft* за *Windows*, *Linux* и *macOS*, включваща дебъгер, вграден *Git*, синтактично оцветяване, автоматично довършване на кода и функционалности за рефакториране на кода. Също така може да се персонализира, тоест потребителите могат да променят темата на редактора, клавишните комбинации и предпочитанията. Той е свободен за ползване и с отворен код, въпреки че официалното изтегляне е с лиценз за собственост.

Visual Studio Code има поддръжка за почти всички популярни езици за програмиране. Някои от тях идват вградени като например *JavaScript*, *TypeScript*, *CSS* и *HTML*, но много други езици могат да бъдат добавени използвайки разширения от *VS Code Marketplace*.

Също така, в проучването на *StackOverflow* за 2018 година, *Visual Studio Code* беше класиран като най-популярната интегрирана среда за програмиране като 34,9% от 75 398 хора, попълнили анкетата, твърдят, че го използват.¹¹

¹⁰ To IDE or Not to IDE?

Източник: <http://radar.oreilly.com/2014/01/to-ide-or-not-to-ide.html>

¹¹ Developer Survey Results 2018

Източник: <https://insights.stackoverflow.com/survey/2018/>

1.3.2 PyCharm

PyCharm, също както и *Visual Studio Code*, е интегрирана среда за разработка (*IDE*), използвана от програмистите специално за програмиране на езика *Python*.

Средата е разработена от чешката компания *JetBrains* и се предлага с безплатен лиценз за ученици и студенти. *IDE*-то осигурява анализ на кода, графичен дебъгер, интегриран unit тестер, интеграция със системи за контрол на версиите (като например *Git*) и поддържа уеб разработки с *Django*.

В същото проучване на *StackOverflow* за 2018 година 12% от разработчиците са казали, че използват точно тази интегрирана среда за разработка. Това е три пъти по-малко от *Visual Studio Code*, но е важно да вземем предвид, че *VS Code* поддържа много повече езици за програмиране.

1.3.3 Sublime

Sublime Text е текстови редактор за различни платформи. Вградено има поддръжка за писане на код на много езици за програмиране, а функциите могат да се добавят от потребители с разширения, обикновено изградени от общността и поддържани под лицензи за свободен софтуер.

Sublime Text, също като *Visual Studio Code* и *PyCharm*, може да бъде персонализиран. Също така, самата програма е доста по-олекотена откъм изисквания за машината, на която ще се изпълнява.

1.3.4 Notepad++

Един класически текстов редактор, използван не само за програмиране, е *Notepad++*. Той включва подчертаване на синтаксиса, сгъване на код и ограничено автоматично довършване на кода, но не и интелигентно завършване на кода или проверка на синтаксиса. Като такъв той може правилно да маркира код, написан в поддържана схема, но дали синтаксисът е верен - не може да бъде проверено.

Въпреки това, той е един от най-популярните избори за текстов редактор. Това ясно си личи и от проучването на *StackOverflow* за 2018, където *Notepad++* заема 3-то място. Пред него са единствено *Visual Studio* и *Visual Studio Code*.

1.4 Избор на средства за графичен интерфейс

Основна част от уеб приложенията, която потребителите забелязват, е именно графичния дизайн. Писането на стилизациите, които съставят дизайна на приложенията, от нулата, както и при правенето на уеб приложения, е трудоемка задача. Най-лесно проблемът, произлизащ от това, се решава чрез използване на

библиотеки и софтуерни рамки за дизайн като за тази цел при повечето проекти трябва да бъдат избрани средства за графичния интерфейс. Освен софтуерни рамки за уеб приложения, има и много софтуерни рамки за графичен интерфейс, позволяващи използването на готови стилови класове и темплейти.

Някои от по-известните софтуерни рамки за графичния интерфейс са *Bootstrap*, *Semantic-UI*, *Foundation*, *Materialize*, *Material UI*, *Pure*, *Skeleton*, *UIKit*, *Milligram*, *Susy* и други.¹²

От изброените софтуерни рамки за проектиране на уеб сайтове и уеб приложения, *Bootstrap* е една от най-популярните. *Bootstrap* е безплатна софтуерни рамка с отворен код, съдържаща шаблони за типография, формуляри, бутони, навигация и други компоненти на интерфейса, както и опционални разширения на *JavaScript*.

За разлика от повечето по-ранни софтуерни рамки, *Bootstrap* се занимава единствено с разработката на *front-end*-а на приложението(тоест там няма да намерите например логика за влизане на потребителите в системата, но можете да намерите как да стилизирате страницата за вход на потребителя).

Софтуерните рамки за графичен интерфейс са много и най-различни. Общото между почти всички е, че са респонсив

¹² **Top 10 Front-End Frameworks of 2018**

Източник: <https://www.keycdn.com/blog/front-end-frameworks>

(подходящи са за ползване и за десктоп версии на уеб приложения, и за мобилни версии) и имат готови стилови класове, от които програмиста може да се възползва. По този начин се спестява много време за написване на едни и същи стилове при започване на нов проект - например чрез използване на софтуерните рамки за графичен интерфейс обикновено типографията идва готова и не е нужно разработчика да задава размерите на шрифтовете на различните типове елементи, но винаги може да промени тези, които идват със софтуерната рамка ако поиска.

1.4 Преглед на подобни социални мрежи

Социалните мрежи са доста срещано явление - има социални мрежи за почти всичко - за споделяне на места, музика, клипове, снимки или просто чат с приятели. Също така има и сайтове за организиране на работа или задачи, има всякакви календари и т.н. Но това, което все още не съществува, е социална мрежа, която да обединява и двете - организиране на задачи и събития и посещаване на събития, организирани от други. Ето и няколко примера за сходни социални мрежи.

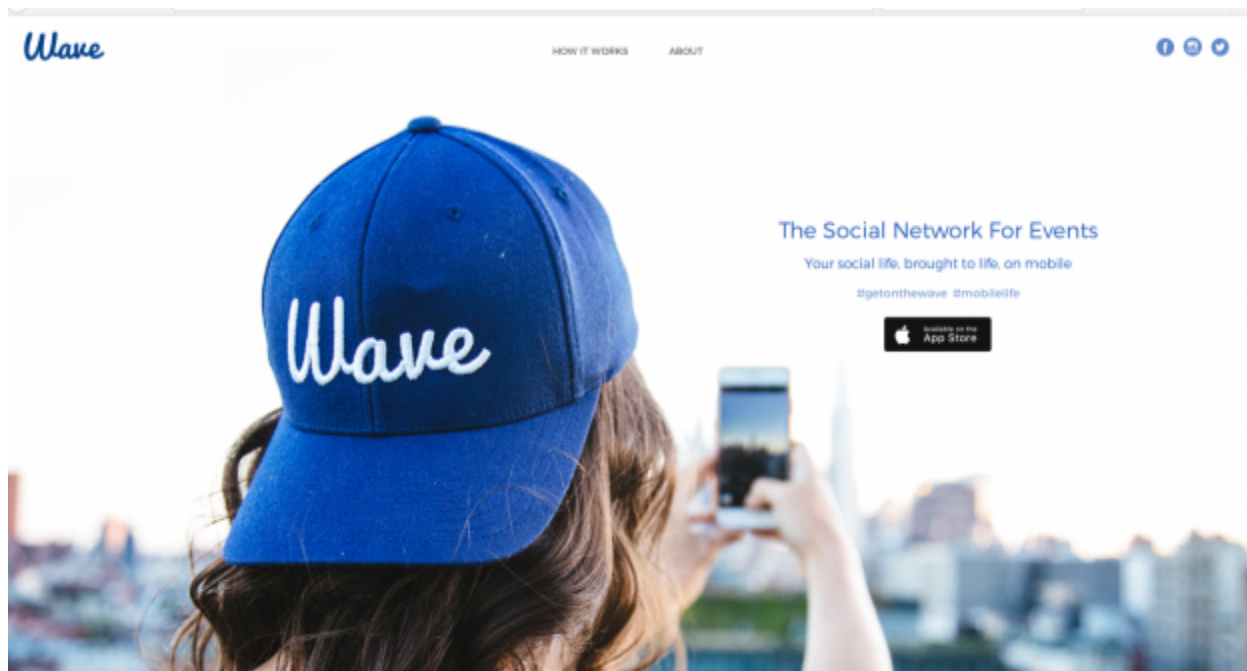
1.4.1 Wave¹³

Wave е приложение за *iPhone*, чрез което хората могат да създават мигновено събития от телефона си и да канят своите фейсбук

¹³ Wave

Източник: <https://www.getonthewave.com/>

приятели. За съжаление приложението е достъпно само за потребители, притежаващи *iPhone*, което е крайно ограничаващо понеже не всички потребители, които биха имали интерес към услугата, разполагат с конкретно това устройство.



Фиг. 1.4.1 Wave

1.4.2 Facebook Local¹⁴

Facebook Local е обновената версия на *Facebook events* - търсачка, която помага на потребителите да открият местни бизнеси и събития в близост до тях. Плюсът на тази социална мрежа е, че понеже е разработвана от *Facebook* има достъп до всички събития и бизнеси в тази социална мрежа, което много обогатява съдържанието.

¹⁴ Facebook Local

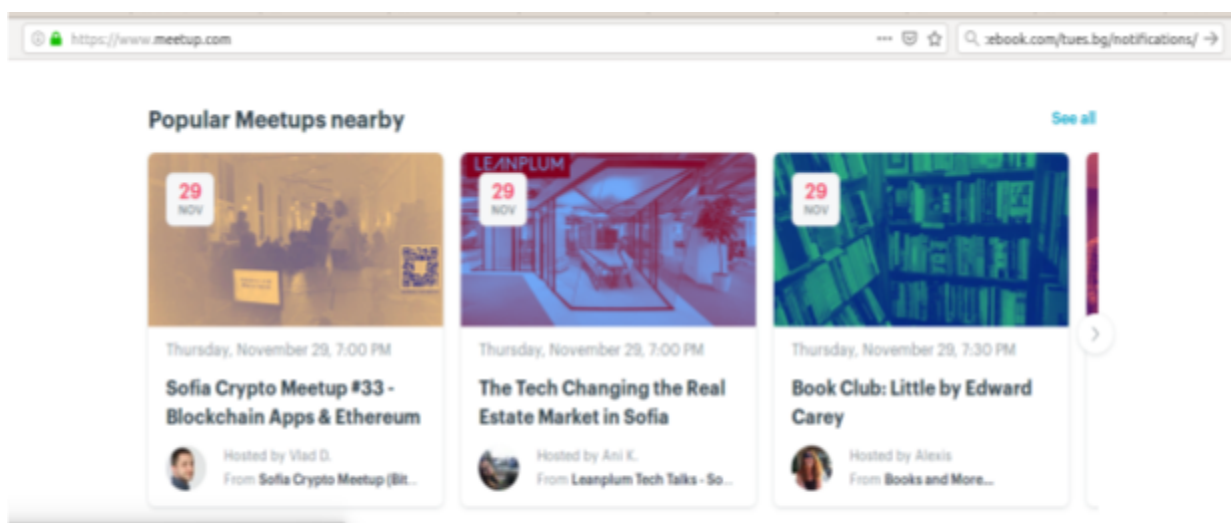
Източник: <https://www.facebook.com/local/>



Фиг. 1.4.2 Facebook local

1.4.3 Meet up¹⁵

Meetup е уеб приложение чрез което хората със сходни интереси могат да се организират в групи и да създават събития, чрез които да се срещат на живо.



Фиг. 1.4.3 Meet up

1.4.5 EventBrite¹⁶

Eventbrite е уеб платформа за управление на билети и събития, позволяваща на потребителите да разглеждат, създават и промотират локални събития. За събития, които не са безплатни, има такса към

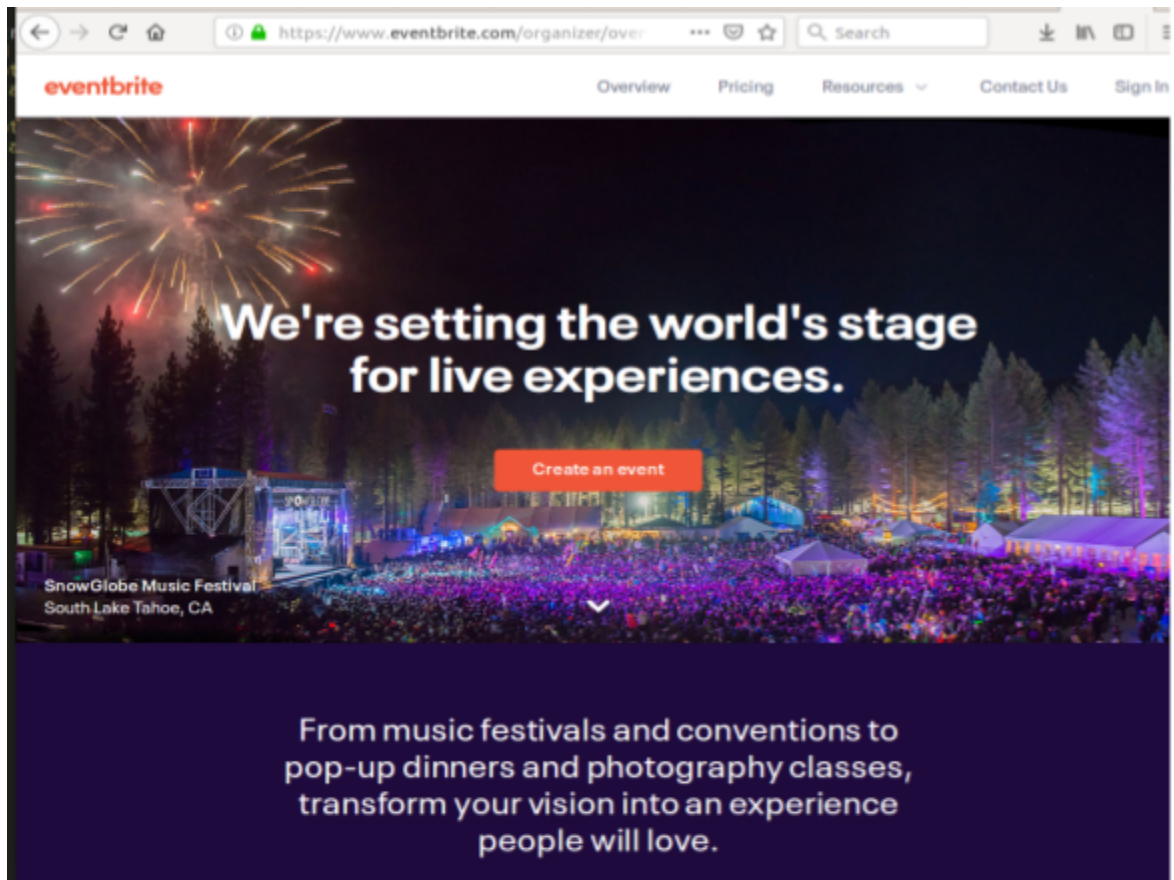
¹⁵ Meetup

Източник: <https://www.meetup.com/>

¹⁶Eventbrite

Източник: <https://www.eventbrite.com/>

самата платформа. Фокусът на тази платформа е върху продажбата на билети и разпространение на безплатни и платени събития.



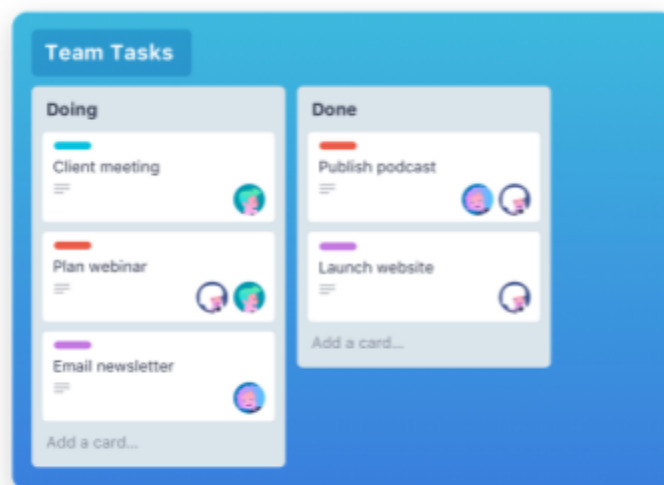
Фиг. 1.4.5 EventBrite

1.4.6 Trello¹⁷

Work with any team

Whether it's for work, a side project or even the next family vacation, Trello helps your team stay organized.

Start doing →



Фиг. 1.4.6 Trello

За разлика от предишните примери за подобни приложения, *Trello* дори не се доближава до това да бъде социална мрежа, но това не е и неговата цел. *Trello* е инструмент за сътрудничество, който дава на потребителите визуален преглед върху какво се работи по даден проект, кой работи върху него и докъде са стигнали.

Trello може да бъде изключително полезен, ако целим да създадем план на задачите, както и за добавяне и проследяване на задачите за вършене по даден проект.

¹⁷ Trello

Източник: <https://trello.com/>

Втора глава

Изисквания към приложението. Проектиране на структурата на уеб базирана социална мрежа за създаване и управляване на събития

2.1 Функционални изисквания към социална мрежа за създаване и управляване на събития

2.1.1 Акаунти

Всеки потребител може да се регистрира в уеб приложението (чрез използване на име, парола и имейл). След това има възможност да добави и детайли за себе си като дата на раждане, кратко описание, което другите потребители ще виждат, и профилна снимка.

Всеки потребител може да добавя друг като приятел. След регистрацията, потребителят може да променя данните, които е въвел за паролата и имейла. Освен да се регистрира, потребителят може и да влезе в системата чрез страницата за login. За всеки акаунт има уникална страница, на която се вижда информация за акаунта, а самият потребител може и да редактира въведените детайли.

Също така, потребителят може да изтрие акаунта си след повторна автентикация чрез парола.

2.1.2 Събития

Всеки потребител има правото да създава и разглежда събития, да се присъединява към събитие ако то е видимо за него и да кани свои приятели към събитието. Всяко събитие си има свои собствени параметри, като: заглавие, описание, снимка, дата и час на започване и завършване, и локация. Тези параметри се задават при самото създаване на събитието, но могат да бъдат редактирани впоследствие от потребителят, създал събитието, или от членовете на екипа.

2.1.3 Коментари

Всеки потребител може да коментира събитията (и тези, на които е създател, и тези, на които е поканен). Няма ограничение на броя коментари под едно събитие от един потребител. Под всяко събитие се виждат всички коментари. Коментарите могат да бъдат изтривани и редактирани само и единствено от потребителят, който го е създали. Потребители, които не са влезли в системата, могат да виждат съществуващите коментари, но не и да добавят нови.

2.1.4 Приятелства

Всеки потребител може да добавя или премахва друг потребител от своите приятели. Всеки потребител има страница, на която може да

види своите приятели, страница с всички потребители и страница, на която може да търси за даден потребител.

2.1.5 Категории

Всяко събитие може да бъде добавено в няколко категории. Категориите са изначално въведени в приложението и потребителите не могат да добавят нови категории. Потребителят може да филтрира събитията по категория. Има страница, на която се листват всички събития, отговарящи на дадения филтър. Към всяко от тези събития има линк, който да отваря страница само за това събитие. Освен това има и страница, на която се листват всички категории.

2.1.6 Страница за всяко събитие

Всяко събитие си има собствена страница на уникален *url*. На тази страница може да се види информация за събитието като локация, описание, визуализират се и всички коментари към даденото събитие, гостите и екипът на събитието. За потребителите, които са влезли в системата, има и възможност за канене на приятели и за добавяне на коментар.

Създателят на събитието и членовете на екипа виждат и бутони за редактиране на събитието, както и бутон за достъп до дъската на събитието, където могат да видят всички задачи към него без значение

дали са разпределени или не. Освен това те могат да добавят свои приятели в екипа на събитието. Също така има и бутон, чрез който потребителят може да отиде на страница, на която да добави свои приятели към екипа на събитието. Членовете на екипа имат достъп и до бутон за изтриване на събитието. Веднъж изтрито, събитието не може да бъде върнато обратно.

2.1.7 Покани

Всеки потребител може да кани друг потребител (стига да са приятели) за дадено събитие. Потребителите имат страница, на която могат да видят всички покани, които са получили и съответно имат възможност да приемат поканите за събитията, които искат, или да изтриват поканите, които не желаят да приемат. Ако поканеният потребител изтрие покана, той може отново да бъде поканен на събитието.

2.1.8 Добавяне на екип към събитието

Създателят на събитието може да добавя други потребители като организатори на събитието, тоест като част от екипа. Екипът на събитието има достъп до допълнителни страници като страницата, на която се листват задачите за вършене към събитието и където могат да добавят нови задачи, и настройките на събитието.

2.1.9 Добавяне на задачи от екипа

Потребителите, които са включени в организационния екип на събитието, могат да добавят задачи за вършене към самото събитие. Тези задачи са видими само за хора, добавени в екипа. Също така, могат да редактират и изтриват тези задачи.

2.1.10 Задачи

За всяка една задача в събитието могат да се добавят потребител, който се грижи за свършването на дадена задача и статус на задачата (to do, doing, done). В зависимост от статуса, задачите са разпределени на различни колони на дъската на събитието. Всеки потребител вижда на едно място всичките задачи, които има да свърши по всички събития, на които е член на организационния екип.

2.1.11 RSS feed

В приложението има и *RSS feed*-ове - един, на който се показват всички събития и друг, на който се виждат най-новите (последните 20 добавени) събития.

2.1.12 API

Към приложението имаме и *API*. Чрез използването на *API*-то могат да се изпълняват заявки за извличане, създаване, редактиране и изтриване върху следните обекти:

- Събития
- Коментари
- Покани
- Детайли на акаунтите
- Задачи

Така тези операции ще могат да бъдат изпълнявани не само през фронтенда на приложението, а и с помощта на други приложения, които директно да правят заявки към *API*-то.

2.1.13 Swagger

Към *API*-то има и документация, която се генерира автоматично чрез използването на *Swagger*. По този начин се гарантира, че ще използва възможно най-новото текстово описание към *API*-то.

2.2 Аргументация на избора на средите за разработка

2.2.1 Django

За софтуерна рамка за уеб приложението *EventManager* избрах *Django*.

Django е софтуерна рамка, базирана на скриптовия език за програмиране *Python*, безплатна е и с отворен код, следваща архитектурния шаблон Модел-Изглед-Шаблон (*Model-View-Template*).

Основното и предназначение е улесняване на създаването на сложни уеб-системи, почти винаги включващи бази данни. Джанго предлага гъвкав *ORM* (*Object-relational mapping*; механизъм, който дава възможност за адресиране, достъп до и манипулиране на обекти, без да се налага да се взима предвид как тези обекти са свързани с техните източници на данни.¹⁸) за моделиране и завършен административен интерфейс за управление на съдържанието. Накратко казано - това е уеб софтуерната рамка за перфекционисти с крайни срокове, каквото всъщност е и мотото на *Django* - “*The web framework for perfectionists with deadlines.*”¹⁹

Причините да избира точно тази софтуерна рамка са няколко:

¹⁸ **Object-relational mapping (ORM)**

Източник: <https://searchwindevelopment.techtarget.com/definition/object-relational-mapping>

¹⁹ **Django Overview**

Източник: <https://www.djangoproject.com/start/overview/>

- *Django* е създаден, за да помогне на разработчиците да преминат възможно най-бързо от идеята за продукта до готовия продукт или поне до работещ прототип
- Джанго включва десетки екстри, които могат да се използват за справяне с общите проблеми при разработване на уеб софтуерно приложение. Например, удостоверяване на потребителя, администриране на съдържанието, карти на сайта, *RSS* емисии и т.н.
- Джанго взема на сериозно сигурността и помага на разработчиците да избягват много често срещани грешки в сигурността, като например *SQL injection*, *cross-site scripting*, *cross-site request forgery* и *clickjacking*.
- Някои от най-използваните сайтове използват способността на *Django* за скалиране, за да поддържат дори и много тежък трафик.

2.2.1.1 Python

Django е софтуерна рамка, базирана на скриптовия език за програмиране *Python* и използваща синтаксиса на езика *Python*. *Python* е интерпретируем, интерактивен, обектно ориентиран език за програмиране от високо ниво, създаден от Гуидо ван Росум в началото на 90-те години.

Н

Тъй като *Python* е език, който се интерпретира, се спестява значително време за разработка, тъй като не са необходими компилиране и свързване (*linking*) за тестването на дадено приложение. *Python* притежава вградени сложни типове данни като масиви и речници, за които биха били необходими дни, за да се напишат ефикасно на *C*.

Python също така позволява разделянето на една програма на модули, които могат да се използват отново в други програми, и притежава голям набор от стандартни модули, които да се използват като основа на програмите.

В ежегодното проучване на *Stack Overflow*, насочено към програмисти от целия свят, за 2018 година, Пайтън²⁰ е в топ 10 на най-популярните технологии. Той заема 7-мо място с 38.8% от всички гласували. Също така заема трето място в най-обичаните от разработчиците езици с 68.0%. Пред него в тази категория са само *Rust* (78.9%) и *Kotlin*(75.1%).

2.2.2 Django Rest Framework (DRF)

Освен самото уеб приложение, имаме и *API*. За разработване на *API*-то към приложението е използвано *Django Rest Framework (DRF)*,

²⁰ Developer Survey Results 2018

Източник: <https://insights.stackoverflow.com/survey/2018/#technology>

защото олекотява създаването на *API* спрямо това ако *API*-то се пише на *Django*.

Django Rest Framework е софтуерна рамка за създаване на *REST API*-та, изискваща *Django* като зависимост. *DRF* генерира *HTML* страници за преглеждане и изпълнение на всички крайни точки (*endpoints*) на дадено *API*. С тази функция потребителите и/или разработчиците могат бързо и лесно да изпълняват *HTTP* методи в своя браузър.

2.2.3 Swagger

За описване на възможните действия през *API*-то или така нареченото изготвяне на автоматична документация използвам *Swagger*.

Swagger е софтуерна рамка с отворен код, която помага на разработчиците да проектират, изграждат, документират и консумират *RESTful* уеб услуги. При създаване на *API*, инструментите на *Swagger* могат да се използват за автоматично генериране на Open *API* документ, базиран на самия код.

Този проект позволява директно свързване към *API*-то чрез интерактивен, базиран на *HTML* потребителски интерфейс. Заявките могат да бъдат направени директно от потребителския интерфейс.

Swagger олекотява работата по генериране и поддържане на документацията на *API*-та и гарантира актуалността на генерираната документация.

2.2.4 JavaScript

JavaScript, често изписван съкратено като *JS*, е език за програмиране на високо ниво, който отговаря на спецификацията *ECMAScript*. Това е език, който се характеризира като динамичен, слабо типизиран и многопарадигмен.

Наред с *HTML* и *CSS*, *JavaScript* е една от трите основни технологии на *World Wide Web*. *JavaScript* позволява интерактивни уеб страници и по този начин е съществена част от уеб приложенията. По-голямата част от уебсайтовете го използват, а всички основни уеб браузъри имат специален *JavaScript* механизъм, който да го изпълни.

Много често *JavaScript* се използва за динамично модифициране на *HTML* и *CSS* за обновяване на потребителския интерфейс чрез използването на *Document Object Model API*.

2.2.4.1 jQuery

Освен *JavaScript*, в технологичния стек на *EventManager* присъства и *jQuery*. *jQuery* е безплатна *JavaScript* библиотека с отворен код, използваща разрешителния лиценз на *MIT*. Предназначена е да опрости обхождането и манипулирането на

HTML DOM дървета, както и обработката на събития и *CSS* анимациите. *jQuery* е библиотеката, която направи *JavaScript* по-достъпен и *DOM* манипулирането по-лесно от преди.

21

Целта на *jQuery* е да направи много по-лесно използването на програмния език *JavaScript* от разработчиците. *jQuery* взима много често срещани проблеми, за решаването на които се изискват много редове *JavaScript* код, и ги трансформира в методи, които можете да извикате с един ред код.²²

Също така, един от най-големите плюсове на тази *JavaScript* библиотека е, че работи по един и същ начин във всички основни браузъри, което е огромно предимство в света на уеб програмирането.

2.2.5 Bootstrap

За потребителския интерфейс на уеб приложението, сведох избора си на софтуерна рамка за графичен интерфейс до *Bootstrap*.

Както е описано в първа глава, *Bootstrap* е безплатна софтуерна рамка с отворен код, съдържаща шаблони за типография, формуляри, бутони, навигация и други основни компоненти на интерфейса. Тя е

²¹ **jQuery**

Източник: <https://jquery.com/>

²² **jQuery Introduction**

Източник: https://www.w3schools.com/jquery/jquery_intro.asp

съвместима с повечето съвременни браузъри, като *Google Chrome*, *Mozilla Firefox*, *Internet Explorer*, *Opera*, *Safari* и други и дава свобода на програмиста за интегриране на неговите идеи към *Bootstrap* темплейтите.

Също така идва и с готови стилове за повечето *HTML* елементи, което значително намалява работата на програмиста в сравнение с това ако му се налага да пише изцяло персонализирани стилове от нулата.

2.2.6 Библиотеки и зависимости

2.2.6.1 Версии на зависимости

За да се изпълнява правилно едно приложение, то се нуждае от определени външни библиотеки, често наричани зависимости. Както всяко нещо в ИТ сектора и външните библиотеки се развиват бързо и затова е важно да се следи коя конкретно версия на всяка библиотека се ползва в приложението, защото някои ъпдейти на външни библиотеки могат да “счупят” нашето приложение.

В света на *Python*, за инсталиране на софтуерни пакети се използва *pip*. Много пакети могат да бъдат намерени в източника по подразбиране за пакети и техните зависимости - *Python Package Index (PyPI)*.

Най-често, за да се следи версията на нужните зависимости към даден проект, се използва requirements файл, който се подава впоследствие като аргумент на *pip*. Това е най-обикновен текстов файл, в който просто са написани имената на пакетите, които *pip* трябва да инсталира, за да работи програмата ни коректно, както и тяхната версия.

Нужните външни библиотеки за *EventManager* се съхраняват във файла requirements.txt. Съдържанието на този файл може да бъде видяно на **Фиг. 2.2.6.1.1**.

```
django==2.1.5
mysqlclient==1.3.13
pycodestyle==2.4
django-mptt==0.8.6
Pillow==5.3.0
django-crispy-forms==1.7.2
django-notifications-hq==1.5.0
django-cleanup==3.0.1
djangorestframework==3.9.0
markdown==3.0.1
django-filter==2.0.0
django-rest-swagger==2.2.0
social-auth-app-django==3.1.0
python-social-auth[django]==0.3.6
django-countries==5.3.2
```

Фиг. 2.2.6.1.1 Requirements на EventManager

2.2.6.2 Използвани Библиотеки

2.2.6.2.1 JavaScript

Единствените използвани *JavaScript* библиотеки в проекта са *jQuery* и *Popper.js*. И двете са нужни за правилното функциониране на избраната софтуерна рамка за графичен интерфейс - *Bootstrap*.

2.2.6.2.2 Django

Всички използвани библиотеки (освен *pycodestyle*, която не е задължителна за работенето на приложението) за *Django* могат да се видят във файла *requirements.txt*.

2.2.6.2.2.1 *django-crispy-forms*

(<https://github.com/django-crispy-forms/django-crispy-forms>)

Според официалната документация на *django-crispy-forms*, това е приложение на *Django*, което позволява лесното изграждане, персонализиране и повторно използване на формуляри, използвайки любимата си *CSS* рамка²³, която в нашия случай е *Bootstrap*.

²³ **Django-crispy-forms**

Източник: <https://django-crispy-forms.readthedocs.io/en/latest/>

Чрез него лесно се спазва консистентност за дизайна на всички форми в приложението.

2.2.6.2.2.2 *django-cleanup*

(<https://github.com/un1t/django-cleanup>)

Django-cleanup автоматично изтрива *FileField* и *ImageField* файловете, когато на тяхно място бъде запазен нов файл, а при изтриване на модела изтрива и файловете, с които е бил попълнен.

Така се пести място, защото потребителят няма ограничение на броя пъти, в които може да променя например профилната си снимка, а старите профилни снимки не са ни нужни след като бъдат сменени от нова снимка.

2.2.6.2.2.3 *social-auth-app-django*

(<https://github.com/python-social-auth/social-app-django>)

Python Social Auth е механизъм за социална идентификация / регистрация, съвместим с някои социални мрежи като *Facebook*, *Google*, *GitHub* и т.н.

В *EventManager* се използва, за да може потребителите да се регистрират и да използват своя *GitHub* профил за аутентикиране.

2.2.6.2.2.4 *django-countries*

(<https://github.com/SmileyChris/django-countries>)

Django-countries е Джанго приложение, което предоставя полето *CountryField*, което представлява *CharField*, осигуряващ избор, съответстващ на официалния *ISO 3166-1* списък със съкращенията на имената на страните²⁴ (с подразбираща се максимална дължина от 2). Също така, свързва избраната държава със статичен файл, съдържащ флага ѝ.

В *EventManager* тази външна библиотека се използва за избиране на локация на събитието и за визуализиране на флага на държавата, в която ще се проведе събитието.

2.2.6.2.2.5 *pycodestyle*

(<https://github.com/PyCQA/pycodestyle>)

²⁴ *django-countries*

Източник: <https://github.com/SmileyChris/django-countries>

Pycodestyle е инструмент за проверка на стила на код, написан на *Python*, спрямо някои от стиловите конвенции в *PEP 8* за езика. Чрез него в проекта се следи качеството на кода - помага всичко да бъде по конвенциите, които са заложиени като стил в този език за програмиране.

2.2.7 База данни - MySQL

В *Django* света основно се използват две бази - *MySQL* и *PostgreSQL*. Основната разлика между тях е архитектурна - *MySQL* е релационна база данни за разлика от *PostgreSQL*, която е обектно-релационна.

Релационна база данни е тип база данни, която съхранява множество данни във вид на релации, съставени от записи и атрибути (полета), и възприемани от потребителите като таблици. Релационните бази данни понастоящем преобладават при избора на модел за съхранение на финансови, производствени, лични и други видове данни.

Софтуерът, който се използва за организиране и управление на този вид бази данни, се нарича най-общо система за управление на релационни бази данни (СУРБД).

Обектно-релационна база данни (ORD) е система за управление на база данни (СУБД), подобна на релационна база данни, но с обектно-ориентиран модел на базата: обекти, класове и наследяване директно се поддържат в схемите на базата данни и в езика за заявки.

За база данни в настоящия проект е избрана да се използва вградената база данни *MySQL*. *MySQL* е система за управление на релационни бази данни с отворен код (*RDBMS*).

Точно както всички останали релационни бази данни, *MySQL* използва таблици, ограничения, тригери, роли, съхранени процедури и изгледи като основни компоненти. Таблицата се състои от редове и всеки ред съдържа един и същ набор от колони.

2.3 Връзки с външни сървиси

2.3.1 Share this

За споделяне на събитията в социалните мрежи в проекта се използва *ShareThis*. *ShareThis* е приспособление "всичко в едно", което позволява на хората да споделят съдържание в мрежата. Поддържа се за *Firefox*, *Mozilla Application Suite* и *Internet Explorer*, а персонализирани джаджи са достъпни за водещи сайтове на издателски блог, включително *Blogger*, *TypePad* и *WordPress*.

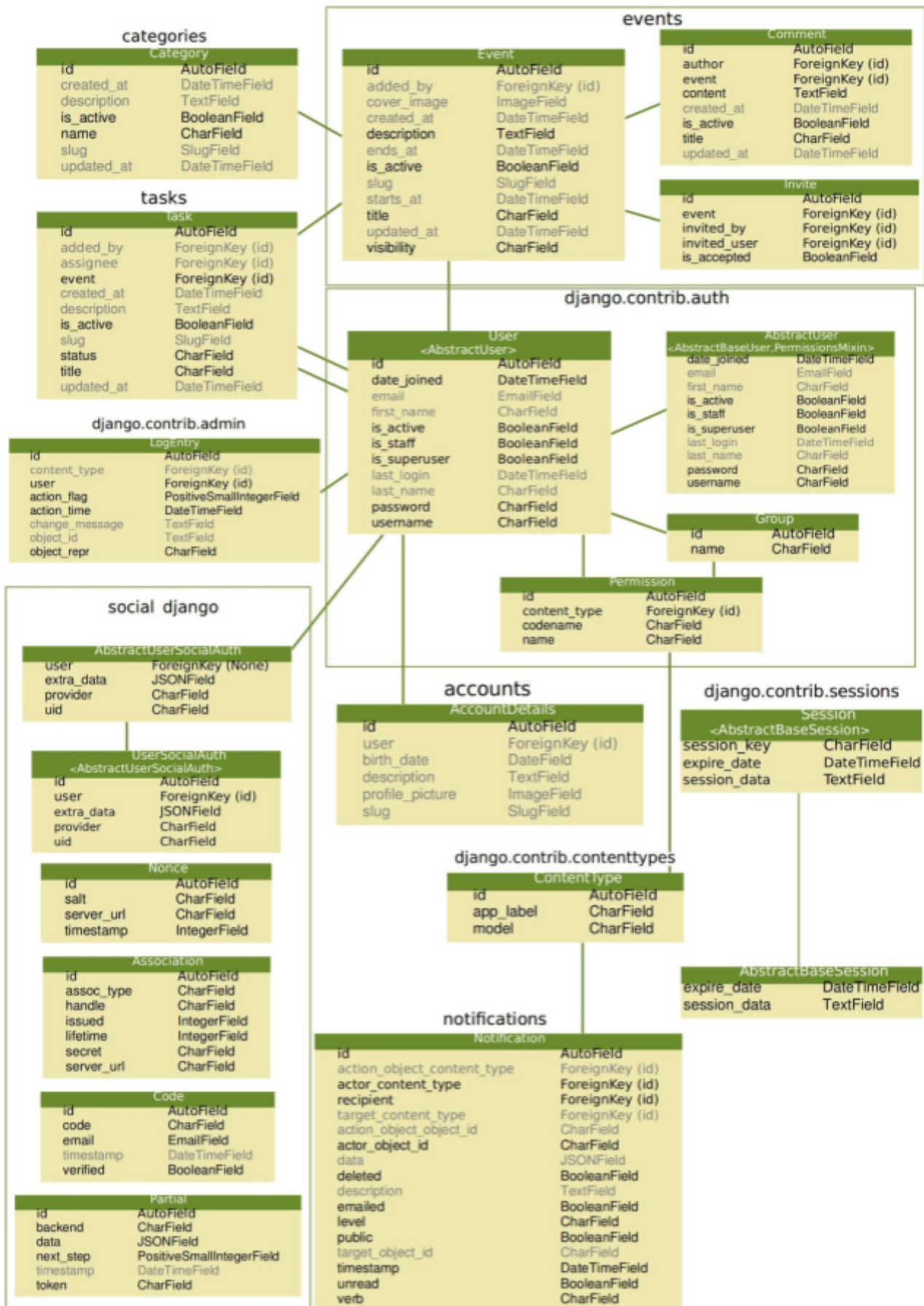
В проекта се използва функционалността за споделяне на съдържание, която в конкретния случай дава възможност за споделяне на събитие по различни популярни социални мрежи (*Facebook, Twitter, Email, LinkedIn, Messenger* и т.н)



2.4 Структура на базата данни

За база данни в приложението се използва *MySQL*. Базата данни на проекта се състои от 21 таблици. Схемата на базата данни е показана на **фиг.**

2.1



Фиг. 2.1 Структура на базата данни

Трета глава

Разработка на уеб приложението

3.1 Потребители

Потребителският модел е реализиран чрез използването на *Django authentication system*²⁵ и по-специално *User objects* (потребителски обекти). Потребителските обекти са в основата на системата за удостоверяване. Те обикновено представляват хората, които взаимодействат със сайта, и се използва за действия като ограничаване на достъпа, регистриране на потребителски профили, асоцииране на съдържание с автор и т.н.

Основните атрибути на потребителя по подразбиране са:

- потребителско име
- парола
- електронна поща
- първо име
- фамилия

²⁵Using the Django authentication system

Източник: <https://docs.djangoproject.com/en/2.1/topics/auth/default/>

Понеже в уеб приложението тези атрибути не са достатъчни изцяло, е избран по-различен похват. Аутентикацията е оставена изцяло на потребителските обекти, но след това на потребителите е дадена възможността да добавят допълнителна информация за себе си. Тази допълнителна информация е реализирана чрез използване на модел *AccountDetails* със следните полета:

- **User**

Foreign Key връзка с базовия модел (*User object-a*); свързва детайлите с акаунта, за който се отнасят

- **Slug**

част от *URL* адрес, който идентифицира страница с четими за хората ключови думи, потребителите не въвеждат slug, а се генерира автоматично, така че да е уникален.

- **Profile_picture**

ImageField, в който се запазва профилната снимка на потребителя

- **Friends**

ManyToManyField, чрез който се създава връзката между потребителите, когато се добавят за приятели

- **Birth_date**

дата на раждане на потребителя

- **Description**

текстово описание, в което потребителят може да включи информация за себе си

Още при регистриране в системата и създаване на потребителски обект, обектът се свързва с модела за детайлите като се попълват полетата user и slug. Самата структура на модела за потребителските детайли може да бъде видяна на **Фиг. 3.1.1.**

Чрез потребителските обекти и модела за потребителските детайли, потребителите на приложението могат да се регистрират, да се аутентикират в приложението със свои собствени потребителско име и парола, да излизат от приложението, да виждат събития, видими само за потребители, които са влезли успешно в системата, да добавят коментари към събитията, да пращат покани за приятелство на други регистрирани потребители, да се присъединяват и да създават събития, към които да добавят задачи.

```
class AccountDetails(models.Model):
    user = models.ForeignKey(
        User,
        null=True,
        blank=True,
        on_delete=models.CASCADE
    )

    slug = models.SlugField(
        unique=True,
        blank=True,
        null=True
    )

    profile_picture = models.ImageField(
        upload_to='profile_pictures',
        blank=True,
        null=True
    )

    friends = models.ManyToManyField(
        User,
```



```

        verbose_name=_("Friends"),
        blank=True,
        related_name="friends"
    )

    birth_date = models.DateField(
        ("birthdate"),
        blank=True,
        null=True
    )

    description = models.TextField(
        verbose_name=_("Description"),
        help_text=_("Plain text, any formatting or links will be removed"),
        unique=False,
        null=True,
        blank=True
    )

```

Фиг. 3.1.1 AccountDetails модел

3.2 Събития

Едно от ключовите неща в проекта са именно събитията. За тях имаме всички възможни *CRUD* (*create, retrieve, update, delete*) операции:

- **Листване на всички събития**

За улеснение на потребителите има страница, на която се виждат всички събития (до които конкретния потребител има достъп). Освен това, има страница, на която се виждат и всички събития, към които логнатият потребител се е присъединил. Така потребителят може да вижда и събитията, на които ще ходи, и да намира нови събития, към които да се присъедини.

И на двете страници се използва *Pagination* (разделяне на елементите по страници; това позволява по-бързо зареждане на страницата, защото не зарежда абсолютно всички събития и по-добро потребителско преживяване). Реализацията на листването на всички събития е показана на **фиг. 3.2.1**

```
def index(request):
    events_list = Event.objects.active()

    events = []
    for event in events_list:
        if Event.can_view_event(event.slug, request.user):
            events.append(event)

    events_list = events
    paginator = Paginator(events_list, number_of_items_per_page)

    page = request.GET.get('page', 1)

    categories = Category.objects.active()
    try:
        events = paginator.page(page)
    except PageNotAnInteger:
        events = paginator.page(1)
    except EmptyPage:
        events = paginator.page(paginator.num_pages)

    context = {'events': events, 'categories': categories}
    return render(request, 'events/list_events.html', context)
```

Фиг. 3.2.1 Листване на всички събития

- **Създаване на събития**

Всеки логнат потребител може да създава събитие.

Събитията имат следните полета:

- **Title** - заглавието на събитието, то се използва за образуване на slug
- **Slug** - изграден по същия начин както и при *AccountDetails*. Използва се за генериране на url-то, на което се вижда конкретното събитие.
- **Description** - описание на събитието
- **Category** - категория, към която принадлежи събитието. Категориите са фиксирани предварително - потребителите не могат да създават нова категория.
- **Visibility** - има различни стойности в зависимост от настройки по събитието(дали да се вижда от всички, само от логнати или само от поканените гости)
- **Attendees** - *ManyToManyField*, в това поле се съдържат всички потребители, които са се присъединили към събитието.
- **Team_members** - *ManyToManyField*, в това поле се съдържат всички потребители, които са добавени в екипа на събитието
- **Cover_image** - снимка, която се използва при визуализиране на събитието
- **Is_active** - поле, което показва дали събитието е активно.
- **Created_at** - *DateTimeField*, показва кога точно е добавено в системата самото събитие
- **Added_by** - *Foreign Key*, показва кой потребител е добавил събитието в системата

- **Updated_at** - *DateTimeField*, показва кога за последно е променяно събитието
 - **Starts_at** - начална дата и час на събитието
 - **Ends_at** - дата и час на края на събитието
- **Конкретно събитие**

За всяко събитие има страница, на която се визуализират детайли за събитието, може да се види кои са гости, кои са членове на екипа, информация за самото събитие и коментарите, направени от потребителите за това събитие. Имплементираното view за показване на конкретно събитие е показано на **Фиг. 3.2.2**

```
def show_events_by_slug(request, slug):
    event = Event.objects.active().get(slug=slug)
    if Event.can_view_event(slug, request.user) is True:
        comments = Comment.objects.active().sort()
        comments = comments.filter(event=event).order_by('-created_at')
        form = CommentForm(request.POST or None)
        username_form = UserForm(request.POST or None)

        storage = messages.get_messages(request)
        for message in storage:
            pass

        final_users = []
        final_team = []

        if request.user.is_authenticated:
            if AccountDetails.objects.filter(user=request.user).exists():
                users = AccountDetails.objects.get(
                    user=request.user).friends.all()
                for user in users:
                    if AccountDetails.objects.filter(user=user):
                        details = AccountDetails.objects.get(
```

```

        user=user
    )
    user.details = details
    final_users.append(user)

    if request.method == 'POST':
        if form.is_valid():
            comment = form.save(commit=False)
            comment.event = event
            comment.author = request.user
            comment.save()

has_joined = Event.has_joined_event(request.user, slug)

team = event.team_members.all()
for user in team:
    if AccountDetails.objects.filter(user=user):
        details = AccountDetails.objects.get(user=user)
        user.details = details
        final_team.append(user)

context = {
    'event': event,
    'comments': comments,
    'form': form,
    'has_joined': has_joined,
    'guests': Event.get_guests(slug),
    'users': final_users,
    'team': final_team,
    'is_team_member': Event.is_team_member(request.user, slug)
}
return render(request, 'events/event.html', context)

error_message = "Event is not available yet\
or you don't have permission to view it."
context = {
    'error_message': error_message
}
return render(request, 'CRUDops/error.html', context)

```

Фиг. 3.2.2 View за показване на конкретно събитие

- **Филтриране по категория**

При листинга на събитията вляво на листинга има панел, чрез който може да се избере категория. Когато се избере тази категория, на страницата се виждат само събитията, които принадлежат към нея. Имплементираното view за филтриране на събития по категория е показано на **Фиг. 3.2.3**

```
def all_from_category(request, slug):
    category = Category.objects.get(slug=slug)
    events_list = Event.objects.filter(category=category)
    number_of_items_per_page = 3
    paginator = Paginator(events_list, number_of_items_per_page)

    page = request.GET.get('page', 1)

    try:
        events = paginator.page(page)
    except PageNotAnInteger:
        events = paginator.page(1)
    except EmptyPage:
        events = paginator.page(paginator.num_pages)

    categories = Category.objects.sort().active()
    context = {'events': events, 'categories': categories}
    return render(request, 'events/list_events.html', context)
```

Фиг. 3.2.3 Филтриране на събития по категория

- **Променяне на събитието**

При променяне на събитието (което е възможно да бъде направено само от създателя на събитието и от хората, които са в екипа на съответното събитие) се визуализира същата форма както и при създаване на събитието, но вече във формата изначално са визуализирани данните на конкретното събитие, което се редактира. След като се променят данните във формата и се натисне бутона за запазване на промените, събитието се променя в базата и вече се вижда новата информация за събитието.

- **Изтриване на събитието**

Събитието може да бъде изтрито само от потребителя, който го е добавил, или от членовете на екипа. Когато бъде изтрито, то не може да бъде върнато обратно.

3.3 Коментари

Към всяко събитие могат да бъдат създавани, гледани, променяни и изтривани коментари. Всеки потребител може да създава коментари само от негово име и да изтрива и променя коментарите, които е създал. Моделът на коментар е показан на **Фиг.3.2.1**.

```
class Comment(models.Model):
    author = models.ForeignKey(
        User,
        on_delete=models.CASCADE
    )
```

```

event = models.ForeignKey(
    'events.Event',
    on_delete=models.CASCADE,
    related_name='comments'
)

title = models.CharField(
    verbose_name=_("Title"),
    max_length=120,
    unique=False,
    null=False,
    blank=False
)

content = models.TextField(
    verbose_name=_("Content"),
    unique=False,
    null=False,
    max_length=500
)

is_active = models.BooleanField(
    verbose_name=_("Is active"),
    default=True
)

created_at = models.DateTimeField(
    verbose_name=_("Created at"),
    auto_now_add=True
)

updated_at = models.DateTimeField(
    verbose_name=_("Updated at"),
    auto_now_add=True
)

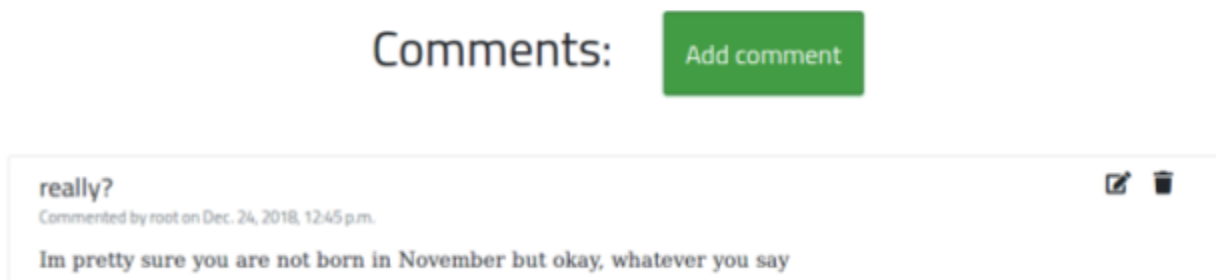
```

Фиг. 3.2.1 Comment модел

Създаването на коментар става по стандартен начин - взима се информацията от формата, превръща се в обект и се съхранява в базата. Аналогично е и редактирането, само че при него директно се редактира върху

конкретна инстанция на обект от базата. При изтриването не се използва форма, а директно се прави заявка към базата за изтриване на съответния обект.

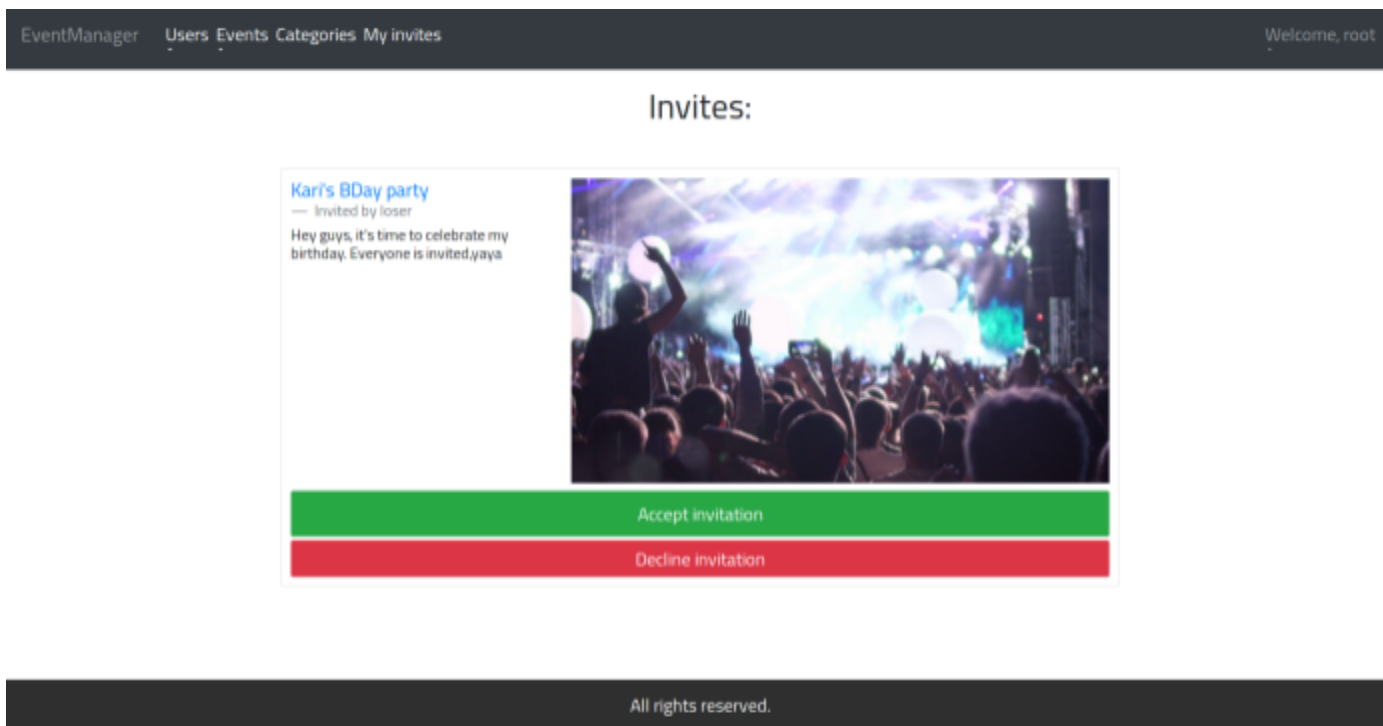
Коментарите се визуализират под самото събитие (**Фиг. 3.2.2**)



Фиг. 3.2.2 Визуализиране на коментар

3.4 Покани

Всеки потребител може да кани друг потребител, стига да са приятели, да се присъедини към събитие. Също така, всеки логнат потребител има страница, на която може да види всички покани, които е получил (**Фиг. 3.4.1**). На въпросната страница може да бъде избрано дали поканата да бъде приета, или да бъде изтрита. По всяко време потребител, който се е присъединил към събитие, може да се откаже и да махне присъединяването си от страницата на самото събитие чрез бутон в съответното събитие, за което иска да направи промяната.



Фиг. 3.4.1 Екран за поканите

Имплементацията на показването на всички покани, каненето на потребител, потвърждаването на покана и отхвърлянето на покана може да се види на **Фиг 3.4.2**.

```
@login_required(login_url='/login')
def invite(request, slug, event):
    logged_in_user = request.user
    logged_in_user_details = AccountDetails.objects.get(user=logged_in_user)
    other_user_details = AccountDetails.objects.get(slug=slug)
    invited_user = other_user_details.user

    event = Event.objects.get(slug=event)
    invite = Invite.objects.get_or_create(
        invited_user=invited_user,
        invited_by=logged_in_user,
        event=event)

    success_message = 'You have invited the user successfully'
```

```

        messages.success(request, success_message)
        return show_events_by_slug(request, event.slug)

@login_required(login_url='/login')
def invites(request):
    user = request.user
    invites = Invite.objects.filter(invited_user=user, is_accepted=False)

    context = {'events': invites}
    return render(request, 'events/invites.html', context)

@login_required(login_url='/login')
def confirm_invite(request, slug):
    event = Event.objects.get(slug=slug)
    event.attendees.add(request.user)
    Invite.objects.filter(
        invited_user=request.user,
        event=event).update(
        is_accepted=True)
    return invites(request)

@login_required(login_url='/login')
def decline_invite(request, slug):
    event = Event.objects.get(slug=slug)
    Invite.objects.filter(
        invited_user=request.user,
        event=event).delete()
    return invites(request)

```

Фиг 3.4.2 Показването на всички покани, канене на потребител, потвърждаване и отхвърляне на покана

3.5 Присъединяване към събитие и напускане на събитие

Всеки потребител, който успешно е влязъл в системата, може да се присъединява и да напуска събитията, които са видими за него. Когато се присъедини, той се добавя към списъка с гости на събитието. Когато напусне събитието, потребителят се премахва от списъка с гости на събитието. Имплементираните *view*-та за присъединяване към събитие и за изтриване на присъединяването са показани на **Фиг. 3.5**

```
@login_required(login_url='/login')
def join_event(request, slug):
    event = Event.objects.get(slug=slug)
    event.attendees.add(request.user)
    event.save()
    context = {'success_message': "joined event " + event.title}
    return render(request, 'CRUDops/successfully.html', context)

@login_required(login_url='/login')
def cancel_join(request, slug):
    event = Event.objects.get(slug=slug)
    event.attendees.remove(request.user)
    event.save()
    context = {'success_message': "removed going status from " + event.title}
    return render(request, 'CRUDops/successfully.html', context)
```

Фиг. 3.5 Присъединяване към събитие и изтриване на присъединяването

3.6 Приятелство

Всеки потребител, който е влязъл в системата, може да изпраща покани за приятелство на други потребители. При приемане на приятелството

поканите се изтриват и всеки потребител се добавя в приятелите на другия потребител. При отказване на поканата за приятелство, поканите се изтриват и е възможно повторно изпращане на покана.

Има екран, на който всеки потребител вижда поканите за приятелство, които има, и съответно може да ги отхвърля или приема. Моделът на поканите за приятелство може да бъде видян на **Фиг. 3.6**. В него има само две полета - едното показва кой потребител изпраща покана, а другото - на кого я изпраща.

```
class FriendRequest(models.Model):
    sent_by = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name='sent_by'
    )

    sent_to = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name='sent_to'
    )
```

Фиг. 3.6 Покани за приятелство

3.7 Тестове на приложението

За тестване на приложението се използват unit тестове. *Unit* тестването е процес на разработка на софтуер, при който най-малките части на

приложение, които могат да се проверяват, наречени единици (*unit-u*), се разглеждат индивидуално и независимо за правилна работа.²⁶

3.7.1 Стандартната библиотека unittest

За тестване в *Django* проекта се използва дефолтната конфигурация - стандартната питонска библиотека *unittest*, която позволява тестване чрез класово базирани методи.

3.7.2 Пример от тест

```
class EventsTestCase(TestCase):
    def setUp(self):
        self.total_number_of_events = 25
        self.client = Client()

        self.client.login(username='john', password='johnpassword')
        category = Category.objects.create(
            name='test event category',
            description='cool description',
            slug="test",
        )

        for event_id in range(self.total_number_of_events):
            eventstring = "test" + str(event_id)
            self.event = Event.objects.create(
                title=eventstring,
                description=eventstring,
            )
            self.event.save()
            self.event.category.add(category)
            self.event.save()

    def test_list_events_lists_event(self):
        url = reverse('events.list')
        resp = self.client.get(url)
```

²⁶ What is unit testing?

Източник: <https://searchsoftwarequality.techtarget.com/definition/unit-testing>

```
self.assertEqual(resp.status_code, 200)
self.assertIn(b"test1", resp.content)
```

Фиг. 3.4 Примерен test case

На **Фиг. 3.4** виждаме примерен *test case*, използван за тестване на листването на събития. *setUp* метода се изпълнява в началото, преди да се изпълнят другите функции на класа, започващи с *test_*. В конкретния пример се създават 25 събития и се логва потребител в системата. След това във функцията за тестване се извиква *url-to*, на което трябва да се листват събитията и се проверява дали *response status code*-а е 200 (тоест, че всичко е наред в заявката) и че се съдържа “*test1*”, което е едно от създадените събития, в съдържанието на *response-a*.

3.7.3 Code Coverage

Code Coverage е процентът на кода, който се покрива от автоматизирани тестове. Измерването му просто показва кои части от кода са били изпълнени по време на тестването и кои не са.

Като цяло, системата за покритие на код събира информация за изпълняваните редове код по време на тестването и след това комбинира това с информация за самия код, за да генерира отчет за покритието на кода.

С разработването на тестове, покритието на кода подчертава аспектите на кода, които може да не са адекватно тествани и които изискват допълнително тестване.²⁷

За отчитане на покритието на кода в настоящия проект се използва *Coverage.py*²⁸, което дава възможност за преглеждане кои редове от кои файлове са тествани и кои не са. В уеб приложението *EventManager* е достигнато покритие на кода от 85%, което се вижда на **фиг. 3.5.**

²⁷ **About Code Coverage**

Източник: <https://confluence.atlassian.com/clover/about-code-coverage-71599496.html>

²⁸ **Coverage.py Documentation**

Източник: <https://coverage.readthedocs.io>

Coverage report: 85%

<i>Module</i>	<i>statements</i>	<i>missing</i>	<i>excluded</i>	<i>coverage</i> %
accounts/views.py	248	86	0	65%
events/views.py	300	93	0	69%
accounts/models.py	67	20	0	70%
eventmanager/slugify.py	37	11	0	70%
tasks/models.py	39	9	0	77%
events/models.py	148	31	0	79%
tasks/views.py	34	6	0	82%
categories/views.py	26	4	0	85%
eventmanager/settings.py	36	2	0	94%
categories/models.py	36	1	0	97%
events/tests.py	197	1	0	99%
accounts/__init__.py	0	0	0	100%
accounts/admin.py	5	0	0	100%
accounts/forms.py	35	0	0	100%
accounts/migrations/0001_initial.py	7	0	0	100%
accounts/migrations/0002_friendrequest.py	6	0	0	100%
accounts/migrations/0003_friendrequest_slug.py	4	0	0	100%
accounts/migrations/0004_remove_friendrequest_slug.py	4	0	0	100%
accounts/migrations/__init__.py	0	0	0	100%
accounts/tests.py	222	0	9	100%
accounts/urls.py	4	0	0	100%
api/__init__.py	0	0	0	100%
api/admin.py	1	0	0	100%
api/migrations/__init__.py	0	0	0	100%
api/models.py	1	0	0	100%
api/serializers.py	31	0	0	100%
api/tests.py	1	0	0	100%
api/urls.py	15	0	0	100%
api/views.py	35	0	0	100%
categories/__init__.py	0	0	0	100%
categories/admin.py	3	0	0	100%

фиг. 3.5 Code Coverage

Създаден е и скрипт, чрез който по-лесно да бъде отчитането на покритието на кода. За да бъде стартиран въпросния скрипт, трябва да се логнем в контейнера на проекта (как да създадете контейнер може да видите в четвърта глава в точка 4.1 Инсталация) чрез викането на следната команда в терминала:

- **`sudo docker exec -it event-manager.container /bin/bash`**

След това трябва да стартираме bash скрипт-а:

- **`bash docker/helpers/coverage.sh`**

Последната команда изпълнява всички миграции и тестове като накрая в терминала показва всички файлове от проекта, редовете, които са били тествани, тези, които не са били тествани, процента на покритие за всеки един файл и редовете, които не са били покрити, както и покритие за целия проект.

След това ако искаме да видим резултатите по по-структуриран и оформен начин в брауъра, можем да изпълним командата:

- **`coverage html`**

Тази команда ще създаде в папката `eventmanager` папка *htmlcov*, от която трябва да отворим през някой уеб браузър файлът *index.html*, през който ще можем да навигираме по файловете, за да се визуализира кои редове не са покрити от автоматизираните тестове.

Четвърта глава

Ръководство на потребителя

4.1 Инсталация

Преди да бъде стартирано уеб приложението на *Ubuntu*, трябва да бъдат инсталирани няколко неща. Първото нещо, което трябва да бъде инсталирано е *Docker*.

Docker е инструмент, създаден, за да улесни създаването и стартирането на приложения с помощта на контейнери. Контейнерите позволяват на разработчика да пакетира приложението с всичките му необходими части (като библиотеки и други зависимости), и да го изпрати като един пакет. По този начин, благодарение на контейнера, разработчикът може да бъде сигурен, че приложението ще работи на всяка друга машина, независимо от персонализираните настройки, които машината може да има, които могат да се различават от машината, използвана за писане и тестване на кода, стига да може на операционната система да се инсталира *docker*.

За да бъде инсталиран докерът обаче, трябва на машината да се инсталират някои пакети. Това става чрез следните команди:

- **sudo apt-get install curl apt-transport-https ca-certificates software-properties-common**
- **curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -**

След това вече може да се инсталира и самия докер:

- **sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu xenial stable" | tee /etc/apt/sources.list.d/docker.list**
- **sudo apt-get update**
- **sudo apt-get install -y docker-ce**

За да проверим дали докерът е стартиран успешно, изпълняваме следната команда:

- **sudo systemctl status docker**

Ако докерът не е стартиран, изпълняваме следните две команди:

- **sudo systemctl start docker.service**
- **sudo systemctl enable docker.service**

Освен самия докер, трябва да инсталираме и *Docker Compose*. За да свалим най-новата версия на *Docker Compose*, изпълняваме следната команда:

- **sudo curl -L https://github.com/docker/compose/releases/download/1.18.0/dock**

```
er-compose-`uname -s`-`uname -m` -o  
/usr/local/bin/docker-compose
```

След това изпълняваме следната команда, която ще направи файлът `/usr/local/bin/docker-compose` изпълним:

- **`sudo chmod +x /usr/local/bin/docker-compose`**

За проверка на версията, изпълняваме следната команда:

- **`sudo docker-compose --version`**

След като имаме инсталирани *docker* и *docker-compose*, можем да стартираме приложението. Преди това обаче трябва да клонираме проекта с помощта на *Git*.

За инсталиране и конфигуриране на *Git* трябва да се изпълнят следните команди:

- **`sudo apt-get update`**
- **`sudo apt-get install git`**
- **`git config --global user.name "Your Name"`**
- **`git config --global user.email "youremail@domain.com"`**

На мястото на `Your Name` и youremail@domain.com сложете вашите потребителски данни за *GitHub*. Ако нямате акаунт, може да се регистрирате на <https://github.com/join>.

След това трябва да се изпълни командата:

- **git --version**

за да се провери дали *git* е инсталиран успешно.

След инсталирането на *git* вече може да бъде клонирано хранилището, в което се намира приложението, чрез изпълняването на:

- **git clone https://github.com/EventManagerTeam/EventManager**

По този начин в текущата директория вече имаме копие на най-новата версия на проекта. Сега се премества в самия проект с командата:

- **cd EventManager**

Ако искате да можете да използвате функционалността за влизане в системата чрез *GitHub* профил, трябва да си създадете файл **local_settings.py**, намиращ се в **eventmanager/eventmanager**. В този файл трябва да попълните следните полета:

- ***SOCIAL_AUTH_GITHUB_KEY* =**
- ***SOCIAL_AUTH_GITHUB_SECRET* =**

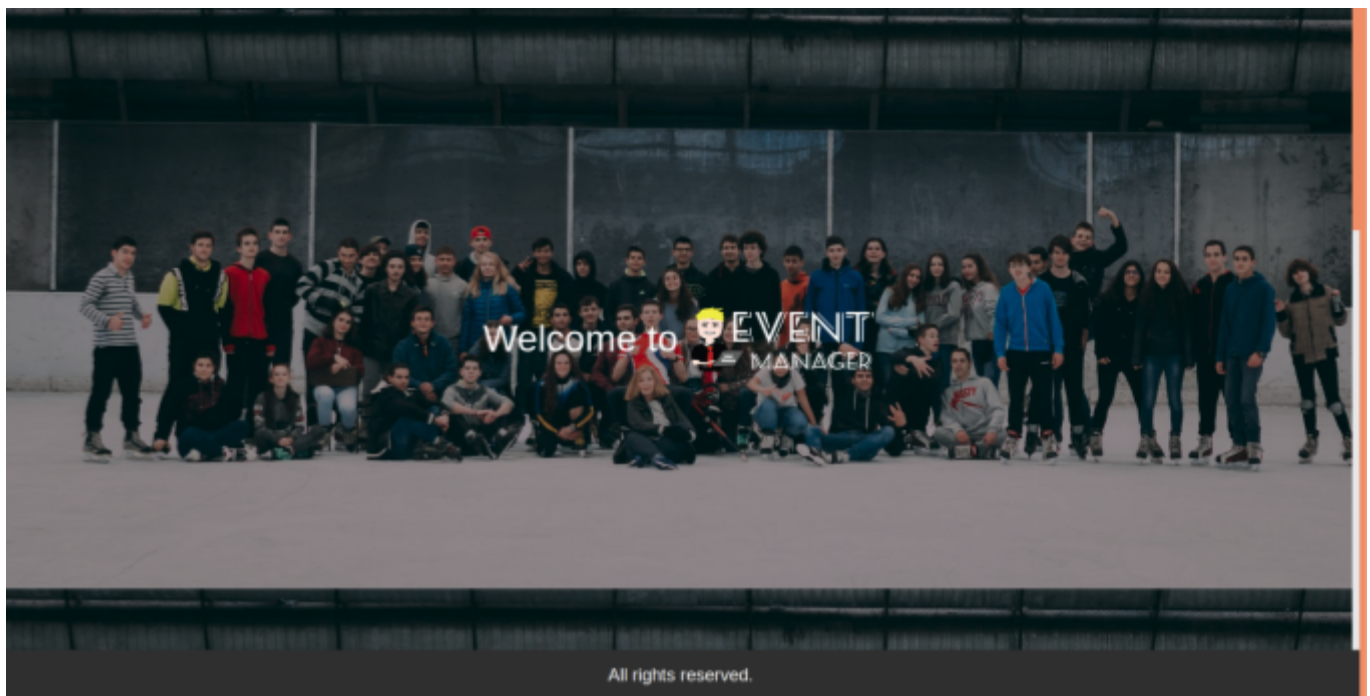
След равното на двете промени трябва да оградите в кавички хеша, който сте получили след като сте регистрирали приложението в *GitHub* на **https://github.com/settings/applications/new**.

Сега остава да създадем и да стартираме контейнера на проекта. Това става чрез следната команда:

- **sudo docker-compose up**

При пускането на последната команда се стартира сървър на 0.0.0.0:8000 следователно на машината, на която се стартира, въпросният адрес трябва да е свободен.

След това, за да бъде проверено дали всичко е наред, трябва да се отиде в браузър на **0.0.0.0:8000**, където трябва да се покаже началната страница на приложението.

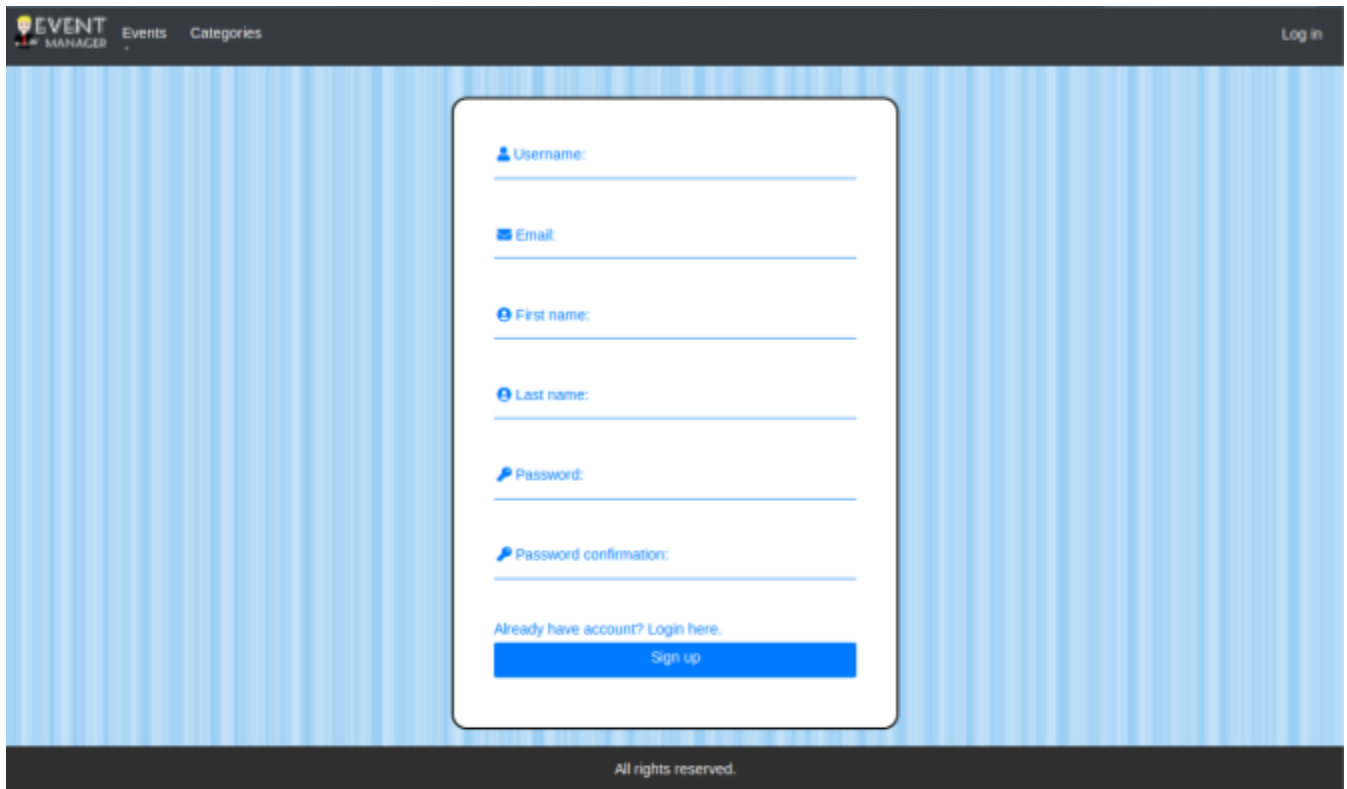


Фиг. 4.1.1 Начална страница

4.2 Използване на уеб приложението

От началната страница(показана на **Фиг. 4.1.1**) потребителят има два избора - да се регистрира(**Фиг. 4.2.1**) или да прегледа вече въведени в системата събития.

За да се използват всички функционалности на приложението, потребителят трябва да се е регистрирал и да е влязъл в системата. Ако вече има акаунт, потребителят може да отиде на страницата за вход, от където да се логне чрез профила си в *EventManager* или *GitHub* акаунт.



EVENT MANAGER Events Categories Log in

Username:

Email:

First name:

Last name:

Password:

Password confirmation:

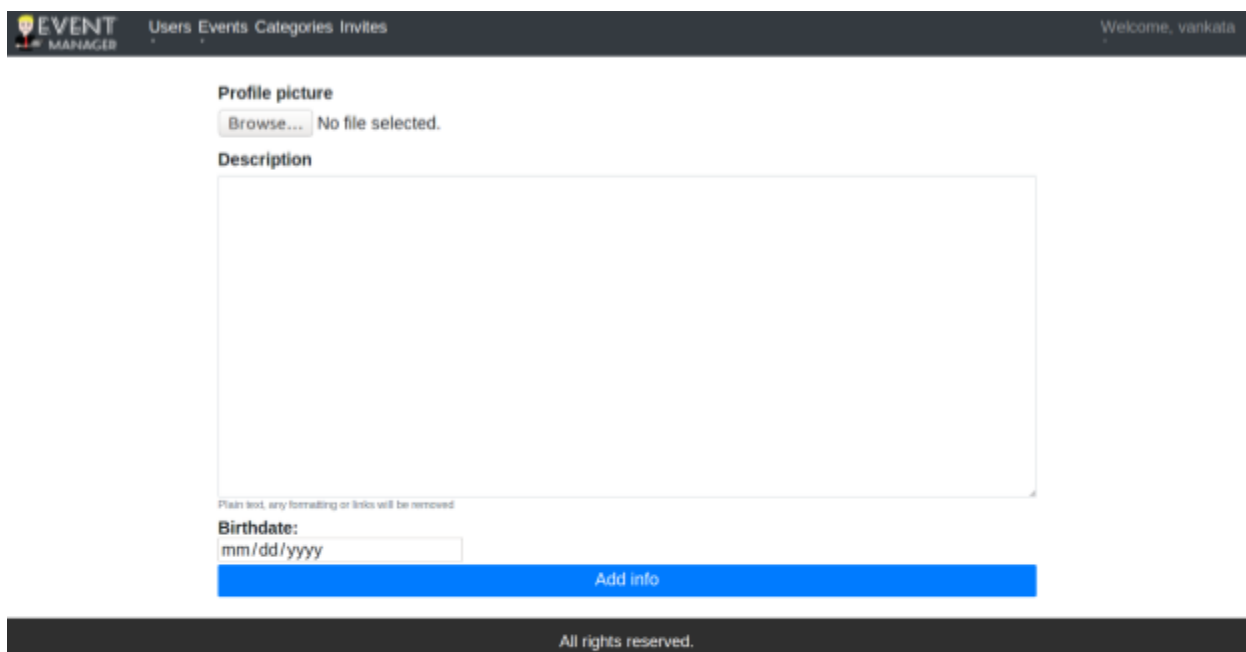
Already have account? Login here.

Sign up

All rights reserved.

Фиг. 4.2.1 Регистрация

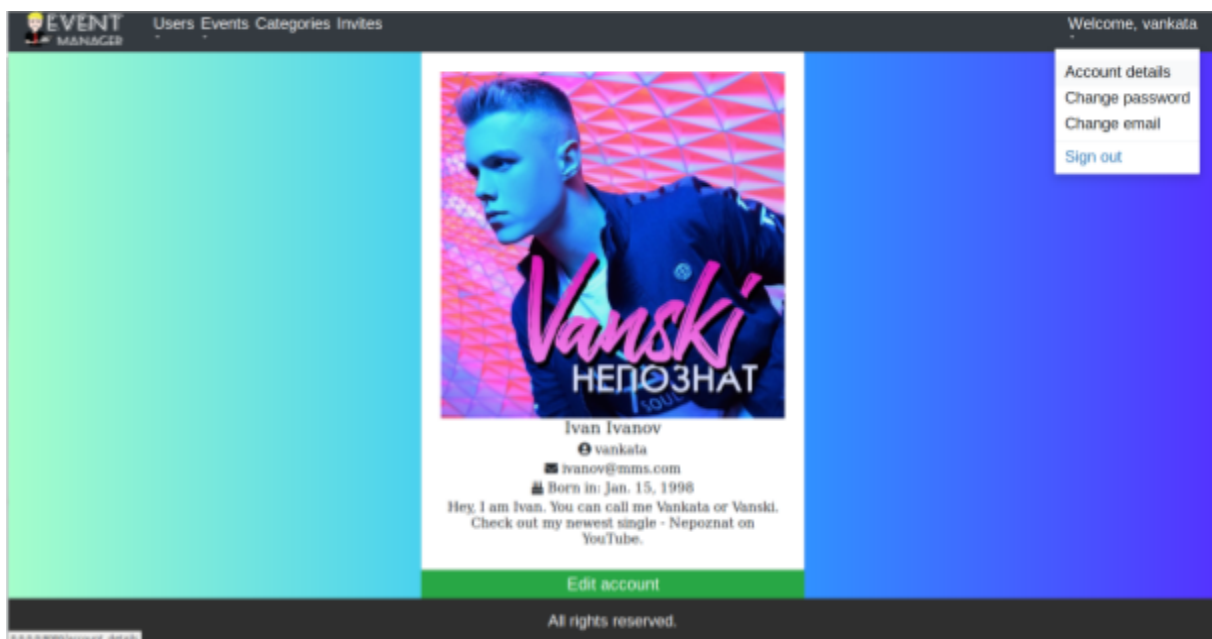
След регистрацията потребителят автоматично се логва в системата с новосъздадения акаунт и му се показва екрана за въвеждане на детайли за профила си (Фиг. 4.2.2).



The screenshot shows a web application interface for 'EVENT MANAGER'. At the top, there is a dark navigation bar with the logo on the left and links for 'Users', 'Events', 'Categories', and 'Invites' in the center. On the right of the bar, it says 'Welcome, vankata'. Below the navigation bar, the main content area contains a profile editing form. The form has three sections: 'Profile picture' with a 'Browse...' button and the text 'No file selected.'; 'Description' with a large text area; and 'Birthdate:' with a date input field showing 'mm/dd/yyyy'. Below the birthdate field is a blue button labeled 'Add info'. At the bottom of the form, there is a small note: 'Plain text, any formatting or links will be removed'. The entire form is set against a light gray background.

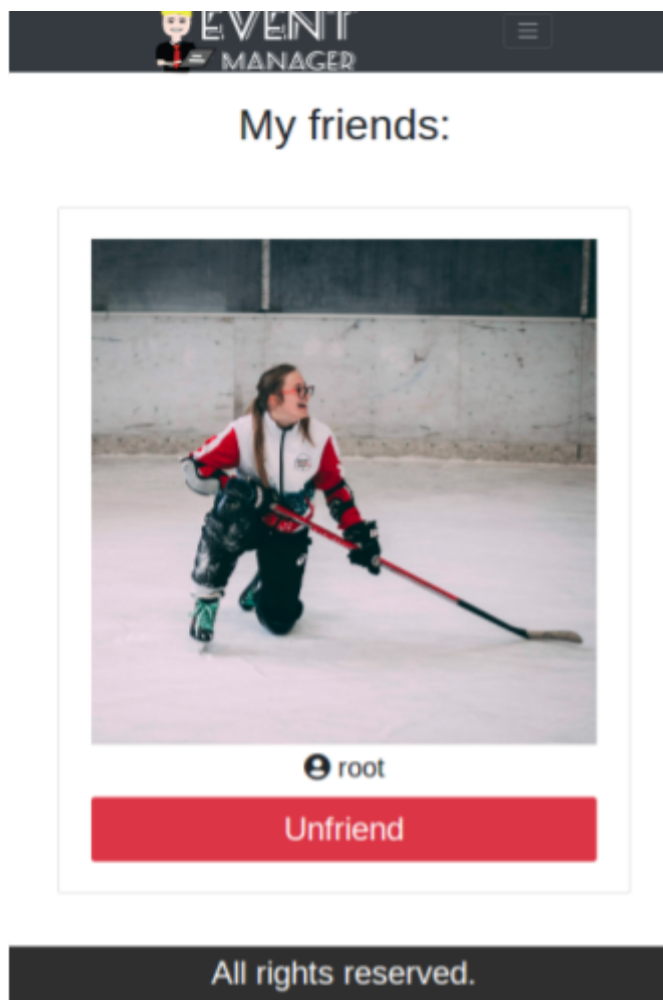
Фиг 4.2.2. Детайли за профила

След като е въвел информация за профила си, потребителят може да я разгледа като отиде на страницата за *Account Details* чрез падащото меню вдясно. (Фиг. 4.2.3). Също така, от менюто може да отвори и страници, чрез които да промени и паролата, и имейла си, а на страницата на детайлите за профила може да промени въведената информация (профилна снимка, дата на раждане и описание на профила).



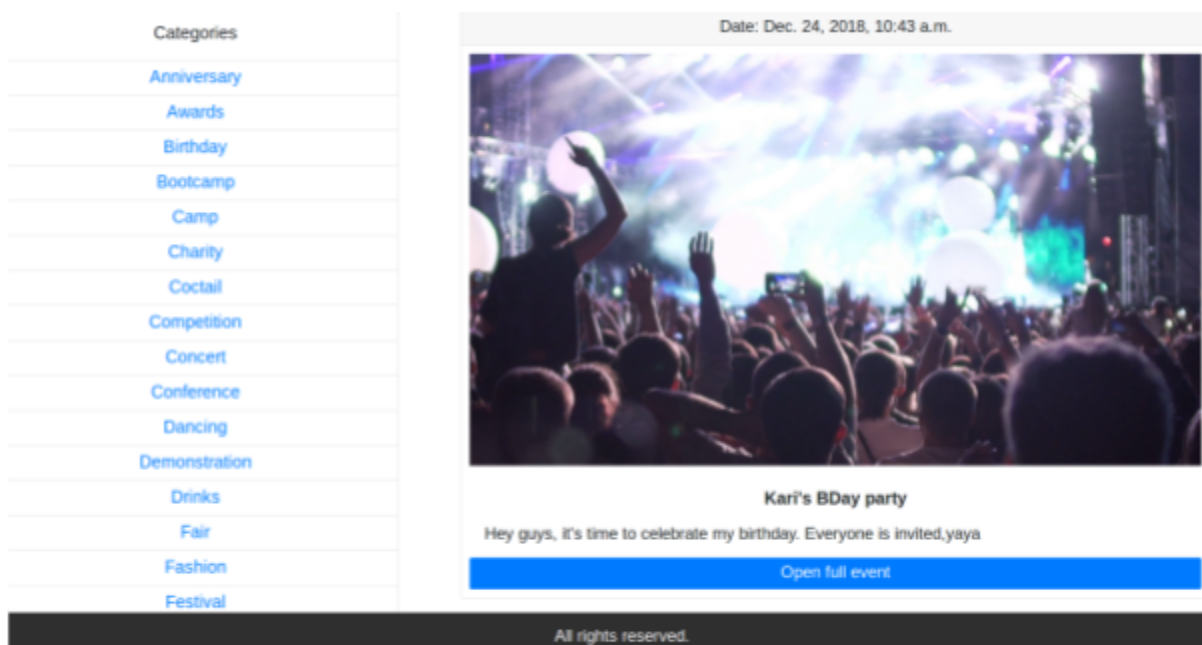
Фиг. 4.2.3 Профил

Освен това потребителят може да разглежда всички потребители в системата и да ги добавя към своите приятели. Има и страница, на която да вижда своите приятели.



Фиг. 4.2.4 Моите приятели

След като е влязъл в профила си, потребителят освен да вижда събития, отворени за всички, може да вижда и събития само за логнати потребители (Фиг.4.2.5) и да създава събития. Също така има и страница, на която може да види всички събития, към които се е присъединил. При преглеждане на дадено събитие потребителят може да се присъедини към него, да го коментира, да вижда информация относно него и да го сподели в други социални мрежи (Фиг. 4.2.6).

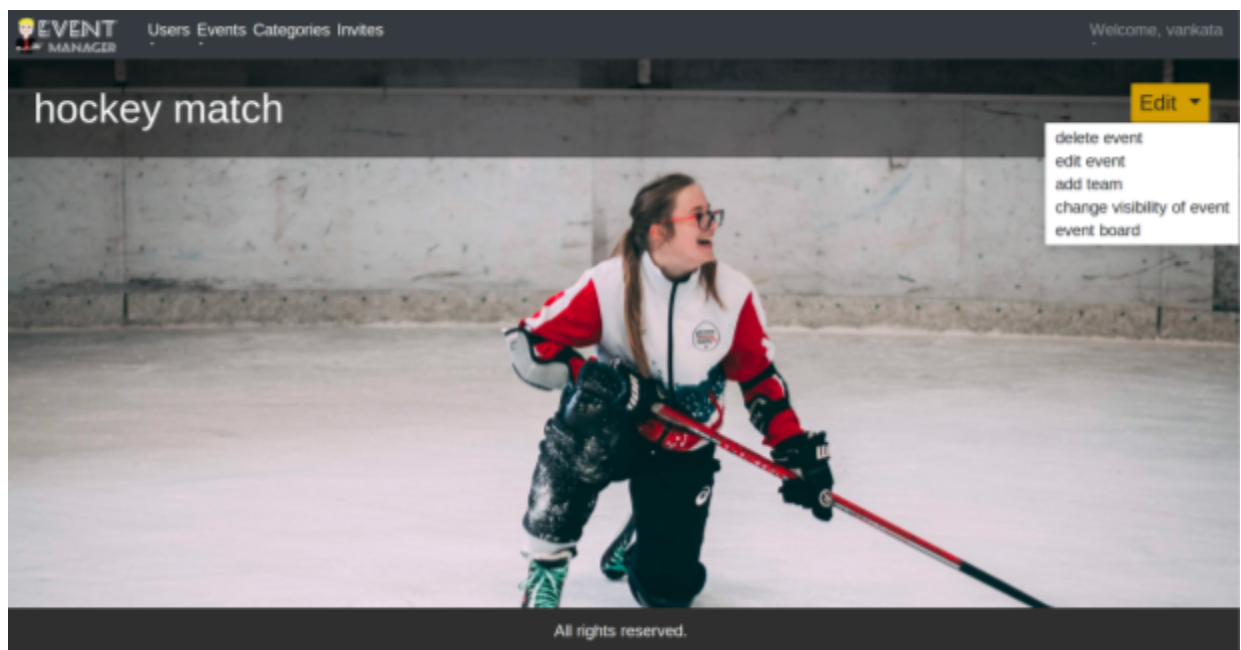


Фиг. 4.2.5. Листване на събития с категории



Фиг. 4.2.6 Част от страница на събитие

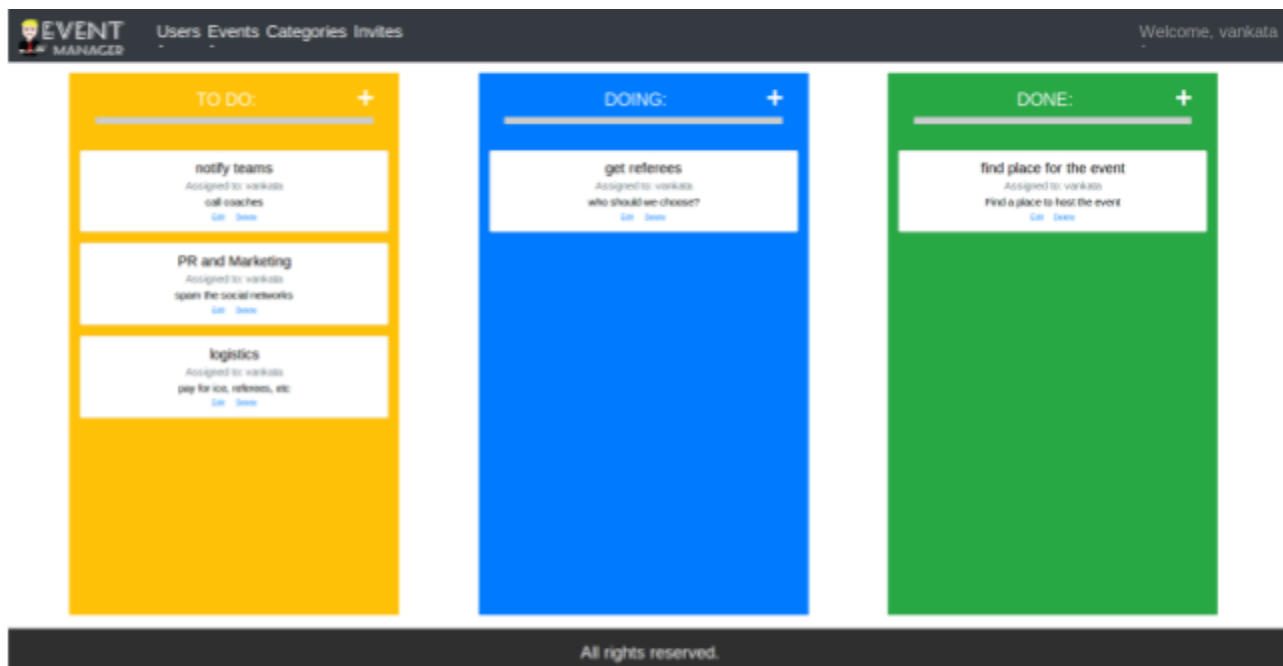
Ако потребителят е създател на събитието или е част от екипа на събитието, той има достъп и до настройките на събитието (Фиг. 4.2.7) през които може да изтрие събитието, да го промени, да добавя приятели към екипа на събитието, да промени настройките за видимост на събитието (дали да се вижда от всички, само от логнатите в системата потребители или само от поканените) (Фиг. 4.2.8) и дъската на събитието (Фиг. 4.2.9).



Фиг. 4.2.7 Настройки на мое събитие



Фиг. 4.2.8. Настройки на видимостта на събитието



Фиг. 4.2.9 Дъска на събитието

На страницата на дъската на събитието (Фиг. 4.2.9) се визуализират всички задачи, добавени от екипа, разделени по статуса им - за правене, правят се в момента или са направени. Всеки, който има достъп до дъската, може да редактира и трие задачи от всеки един борд (*TODO*, *DOING*, *DONE*).

Заклучение

В настоящата дипломна работа успешно бе разработено уеб приложение, което предоставя възможност на потребителите да създават и управляват събития. Приложението коректно покрива всички заложиени функционални изисквания, следвайки стандартите за качество на кода и използвайки автоматизирани тестове, чрез които да се следи коректното изпълнение на бизнес логиката.

Чрез *EventManager* потребителите могат освен да създават и да се присъединяват към различни събития, да следят прогреса си по задачите за събитията, които организират. Така на едно място те могат да имат едновременно и социална мрежа, чрез която да популяризират събитията си, и инструмент за подобряване на организацията им и разпределяне на задачите за вършене.

Също така, приложението *EventManager* има и широки възможности за бъдещо развитие. Подобрения могат да бъдат направени по вече съществуващите функционалности, както и може да бъдат добавени нови функционалности като например да бъде създадено приложение за мобилни телефони, за да бъде по-бързо и лесно за потребителите използването на *EventManager* от мобилни устройства. Възможно е и добавяне на още социални функционалности като съобщения между потребителите или добавяне на галерия за снимки от събитията.

Исползвана литература и ресурси

- Bootstrap - <http://getbootstrap.com/getting-started/>
- Coverage.py - <https://coverage.readthedocs.io>
- Django - <https://docs.djangoproject.com/en/2.1/>
- Django Cleanup - <https://github.com/un1t/django-cleanup>
- Django countries - <https://github.com/SmileyChris/django-countries>
- Django Crispy Forms -
<https://github.com/django-crispy-forms/django-crispy-forms>
- Django Rest Framework - <https://www.django-rest-framework.org/>
- Django REST Swagger -
<https://github.com/marcgibbons/django-rest-swagger>
- Github - <https://guides.github.com/>
- JavaScript - <https://developer.mozilla.org/bm/docs/Web/JavaScript>
- jQuery - <https://jquery.com/>
- Pep8 style guide - <https://www.python.org/dev/peps/pep-0008/>
- Pillow - <https://github.com/python-pillow/Pillow>
- pip - <https://pip.pypa.io/en/stable>
- Pycodestyle - <http://pycodestyle.pycqa.org/en/latest/>
- Swagger - <https://swagger.io/>
- Travis - <https://docs.travis-ci.com/>

Съдържание

Увод	4
Методи и технологии за реализиране на уеб приложения	6
1.1 Уеб приложение, уеб сайт, уеб страница	6
1.1.1 Уеб приложение	6
1.1.2 Уеб сайт	8
1.1.3 Уеб страница	8
1.2 Начини за разработка на уеб приложение, frameworks, MVC	9
1.2.1 Софтуерна рамка	10
1.2.2 Model-View-Controller	10
1.2.2.1 MVT	12
1.2.3 API	13
1.3 Текстови редактори. Интегрирани среди за програмиране.	14
1.3.1 Visual Studio Code	15
1.3.2 PyCharm	16
1.3.3 Sublime	16
1.3.4 Notepad++	17
1.4 Избор на средства за графичен интерфейс	17
1.4 Преглед на подобни социални мрежи	19
1.4.1 Wave	20
1.4.2 Facebook Local	21
1.4.3 Meet up	22
1.4.5 EventBrite	22
1.4.6 Trello	24
Изисквания към приложението. Проектиране на структурата на уеб базирана социална мрежа за създаване и управление на събития	25
2.1 Функционални изисквания към социална мрежа за създаване и управление на събития	25
2.1.1 Акаунти	25

2.1.2 Събития	26
2.1.3 Коментари	26
2.1.4 Приятелства	27
2.1.5 Категории	27
2.1.6 Страница за всяко събитие	27
2.1.7 Покани	28
2.1.8 Добавяне на екип към събитието	28
2.1.9 Добавяне на задачи от екипа	29
2.1.10 Задачи	29
2.1.11 RSS feed	29
2.1.12 API	30
2.1.13 Swagger	30
2.2 Аргументация на избора на средите за разработка	31
2.2.1 Django	31
2.2.1.1 Python	32
2.2.2 Django Rest Framework (DRF)	33
2.2.3 Swagger	34
2.2.4 JavaScript	35
2.2.4.1 jQuery	35
2.2.5 Bootstrap	36
2.2.6 Библиотеки и зависимости	37
2.2.6.1 Версии на зависимости	37
2.2.6.2 Използвани Библиотеки	39
2.2.6.2.1 JavaScript	39
2.2.6.2.2 Django	39
2.2.6.2.2.1 django-crispy-forms	39
2.2.6.2.2.2 django-cleanup	40
2.2.6.2.2.3 social-auth-app-django	40
2.2.6.2.2.4 django-countries	41
2.2.6.2.2.5 pycodestyle	41
2.2.7 База данни - MySQL	42
2.3 Връзки с външни сървиси	43
2.3.1 Share this	43

2.4 Структура на базата данни	44
Разработка на уеб приложението	46
3.1 Потребители	46
3.2 Събития	49
3.3 Коментари	55
3.4 Покани	57
3.5 Присъединяване към събитие и напускане на събитие	60
3.6 Приятелство	60
3.7 Тестове на приложението	61
3.7.1 Стандартната библиотека unittest	62
3.7.2 Пример от тест	62
3.7.3 Code Coverage	63
Ръководство на потребителя	67
4.1 Инсталация	67
4.2 Използване на уеб приложението	71
Заклучение	79
Използвана литература и ресурси	80
Съдържание	81