|  |
| --- |
| SwordsAndBooks Архитектурен дизайн |
|
| *Проект по „Софтуерни технологии“, 2015 г.* |
| Факултет математика и информатика, Софийски университет |

|  |
| --- |
| Изготвили: |
| *80662*, *Михаил Костадинов Господинов*, *bladespinner@gmail.com*  *Компютърни Науки, кур 4с, поток 1, група 3* |
| *80703*, *Христо Атанасов Аврамов*, *hristoavr@gmail.com*  *Компютърни Науки, кур 4с, поток 1, група 3* |
| *80654*, *Калоян Христов Калудов*, *unsigned92@gmail.com*  *Компютърни Науки, кур 4с, поток 1, група 3* |
|  |

Ръководител: доц. Димитър Биров

2015 г.

Съдържание

[1 Увод 5](#_Toc410494249)

[2 Описание и изисквания 5](#_Toc410494250)

[2.1 Цел и обхват 5](#_Toc410494251)

[2.2 Базова функционалност 5](#_Toc410494252)

[2.3 Скорост, използваемост и поддръжка 8](#_Toc410494253)

[2.4 Надежност и сигурност на системата 9](#_Toc410494254)

[2.5 Конкурентност, модифицируемост и тестируемост на системата. Стратегии за разширяване (scaling) 9](#_Toc410494255)

[2.6 Оптимизация за търсачки, семантичност на съдържанието 10](#_Toc410494256)

[2.7 Анализ на поведението на потребителите, AB тестване, профайлинг, Баунс рейтове и статистики 14](#_Toc410494257)

[3 Качествени атрибути 15](#_Toc410494258)

[3.1 Достъпност (availability) 15](#_Toc410494259)

[3.1.1 Дефиниция 15](#_Toc410494260)

[3.1.2 Сценарии 15](#_Toc410494261)

[3.1.3 Решения на системата 16](#_Toc410494262)

[3.2 Бързодействие (performance) 17](#_Toc410494263)

[3.2.1 Дефиниция 17](#_Toc410494264)

[3.2.2 Сценарии 17](#_Toc410494265)

[3.2.3 Решения на системата. 18](#_Toc410494266)

[3.3 Сигурност (security) 18](#_Toc410494267)

[3.3.1 Дефиниция 18](#_Toc410494268)

[3.3.2 Сценарии 19](#_Toc410494269)

[3.3.3 Решения на системата. 20](#_Toc410494270)

[3.4 Модифицируемост (modifiability) 20](#_Toc410494271)

[3.4.1 Дефиниция 20](#_Toc410494272)

[3.4.2 Сценарии 21](#_Toc410494273)

[3.4.3 Решения на системата. 22](#_Toc410494274)

[3.5 Възможост за тестване(testability) 22](#_Toc410494275)

[3.5.1 Дефиниция 22](#_Toc410494276)

[3.5.2 Сценарий 22](#_Toc410494277)

[3.5.3 Решения на системата. 23](#_Toc410494278)

[3.6 Използваемост и удобство на дизайна (usability) 23](#_Toc410494279)

[3.6.1 Дефиниция 23](#_Toc410494280)

[3.6.2 Сценарий 23](#_Toc410494281)

[3.6.3 Решения на системата. 24](#_Toc410494282)

[4 Архитектура 24](#_Toc410494283)

[4.1 Софтуерна Архитектура 25](#_Toc410494284)

[4.1.1 Controller layer 26](#_Toc410494285)

[4.1.2 VIEW 29](#_Toc410494286)

[4.1.3 MODEL 31](#_Toc410494287)

[4.1.4 DATA 33](#_Toc410494288)

[4.1.5 SERVICE Layer 34](#_Toc410494289)

[4.1.6 ANALYTICS 36](#_Toc410494290)

[4.1.7 DATA MINING 36](#_Toc410494291)

[4.2 Физическа Архитектура 37](#_Toc410494292)

[4.3 Архитектура на фронтенда 39](#_Toc410494293)

[4.3.1 Require JS 39](#_Toc410494294)

[4.3.2 Асинхронно зареждане на епизоди 39](#_Toc410494295)

[4.4 Архитектура на процеса по разработка. 40](#_Toc410494296)

[4.4.1 npm 40](#_Toc410494297)

[4.4.2 Grunt 40](#_Toc410494298)

[4.4.3 Bower 41](#_Toc410494299)

[4.4.4 Git и Github 42](#_Toc410494300)

[4.4.5 Аgile and Kanban 42](#_Toc410494301)

[4.4.6 Физически характеристики на елементите на системата. 43](#_Toc410494302)

[4.5 Мрежова Архитектура 43](#_Toc410494303)

[4.5.1 Външни интерфейси 43](#_Toc410494304)

[4.5.2 Вътрешни интерфейси и ноудове, събнети. 43](#_Toc410494305)

[5 Алтернативни архитектури (2 бр.) 44](#_Toc410494306)

[5.1 Алтернативна архитектура 1 44](#_Toc410494307)

[5.2 Алтернативна архитектура 2 47](#_Toc410494308)

[6 Заключение 48](#_Toc410494309)

[7 Ресурси 49](#_Toc410494310)

# Увод

*SwordsAndBooks* е платформа за създаване и четене на книги игри, книга игра е книга, в която на читателя се дава право да решава как да се развива действието , тоест читателя има избор в самата книга. Основният интерфейс на приложението представлява платофрми за намиране, четене и писане на книги игри.

Системата е поддържана от многослойна софтуерна архитектура, изпълнява най-новите препоръки за бързодействие, надежност, конкурентност, сигурност според топ-10 на OWASP[1], възможност за поддръжка.Системата също е създадена да е разшируема, лесно модифицируема използваема както и е направена достъпна за потребители с увредено или намалено зрение.

# Описание и изисквания

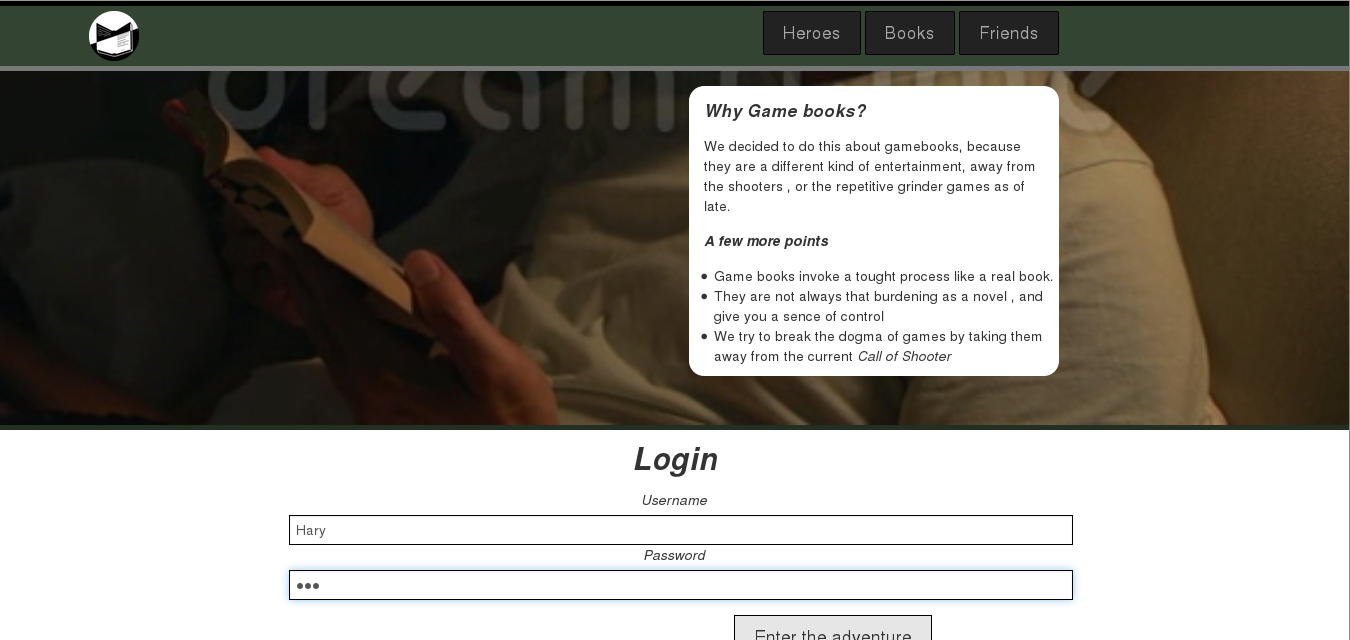
## Цел и обхват

Books and Swords е платформа за създаване и четене на книги игри (книга игра е книга, в която на читателя се дава право да решава как да се развива действието , тоест читателя има избор в самата книга). Основният интерфейс на приложението ще представлява платофрми за намиране, четене и писане на книги игри.Проекта е обмислян като система предоставяща достъп на хора по целия свят.

## Базова функционалност

Платформата за четене ще представлява удобен за потребителя интерфейс, с възможност за запазване на предпочитания за фон, шрифт и други. Тук ще се включват възможности за аудио прочитане на книгата за потребителя от приложението, както и възможност за навигиране на целия интерфейс за платформата за четене, чрез гласови команди от потребителя. В платформите за четене ще има още инртефейси за провеждане на битки,игра на мини игри,  разрещаване на логически загадки и други.

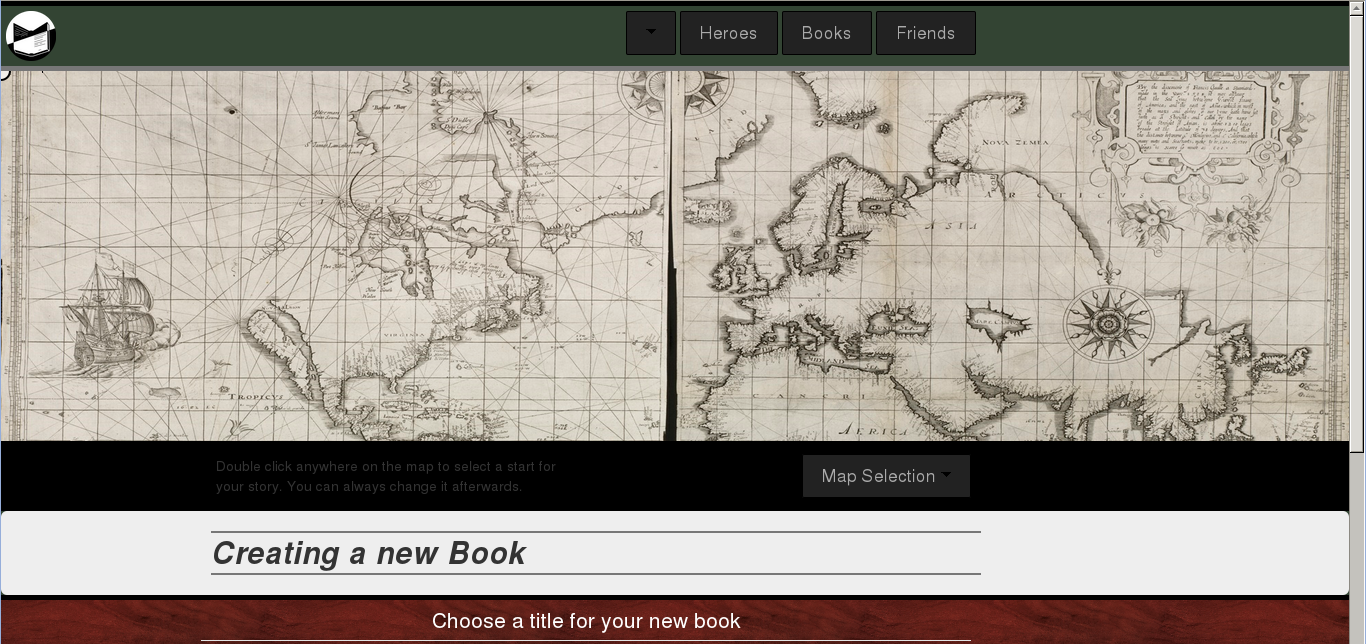
Платформата за писане на книги , ще включва модерен текстов редактор, удобен интерфейс за построяване на графа на книгата игра, допълнителни екстри за добавяне към всеки епизод, представени като widget система(логически загадки , мини игри , визуални ефекти и други). Ще бъде имплементирана и възможност за писане на текст посредством диктуване от потребителя.



**Фиг. 1**.*Логин скрин на приложението.*

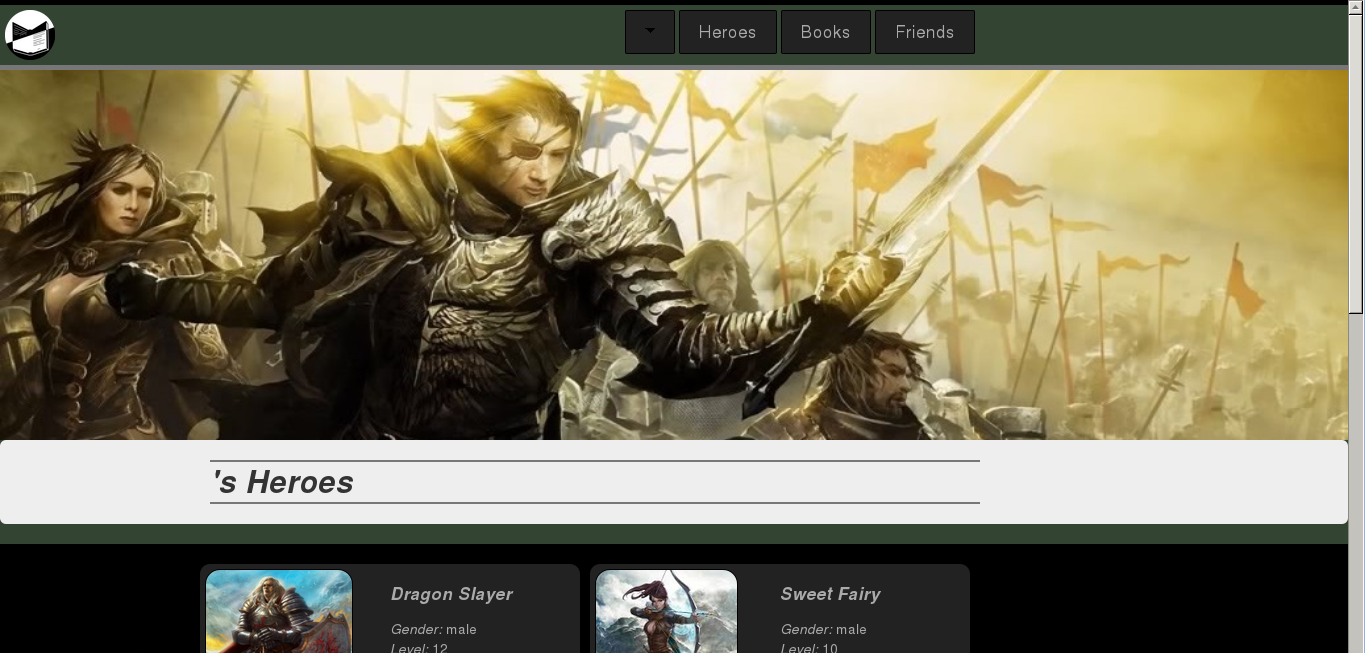
Допълнителните функционалности на приложението ще включват система  за развиване на герой, множество  начини за комуникация между потребителите, възможности за колаборативно писане и играене на книги игри , система за оценяване на книги в системата , интеграция с по- големите социални мрежи, както и функционалности за експортиране на книги игри в удобни формати, така че след малка или никаква реконстукция да бъдат създадени и хартиени книги от същия жанр.

Системата за развиване на герой, ще бъде стандартна и ще бъде силно геймифицирана с вдигане на нива , изпълняване на предизвикателства и възможност за намиране и търгувне с предмети. Всеки герой ще печели награди и пари, след като  с него бъде изиграна до край някоя книга. Така потребителите ще бъдат стимулирани да развиват геройте си,  за да могат да преминават все по-трудни книги игри.



**Фиг. 1**.*Създаване на книга*

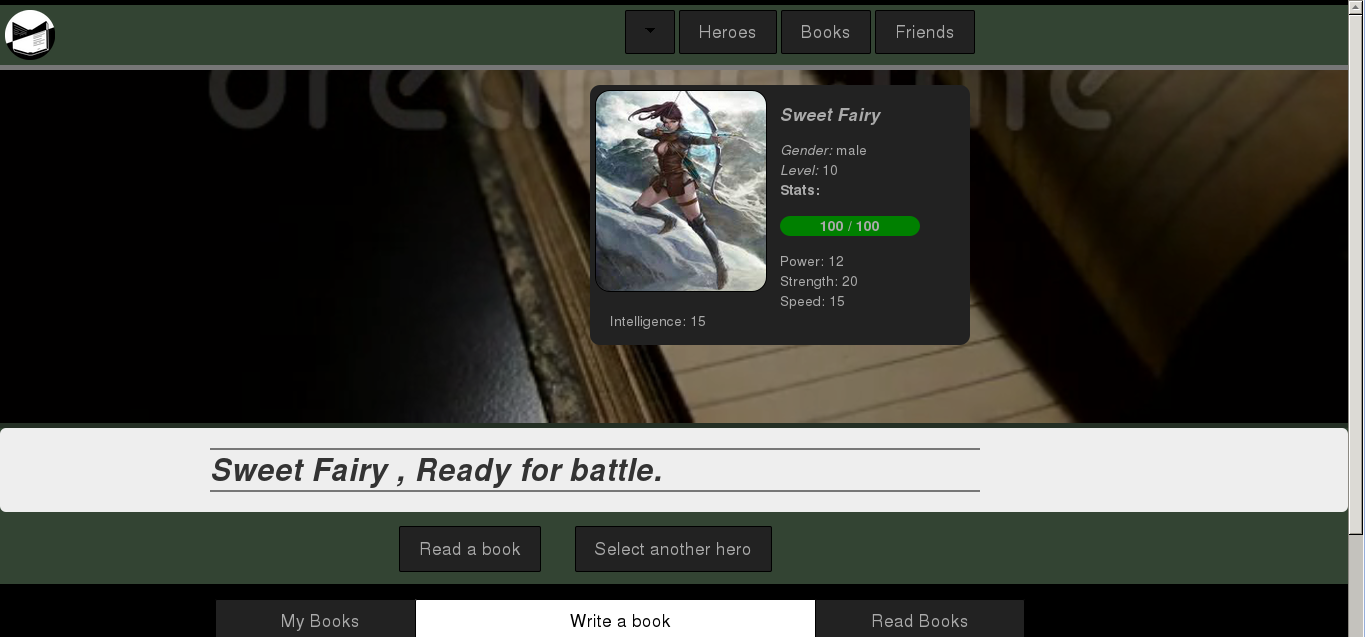
Функционалностите за комуникация  ще представляват чатове на живо, във всеки епизод от книга или в стая с потребители. Ще бъдат също имплементирани видео и аудио разговори , както и форум система, в която потребителите ще могат да обсъждат дадена книга. Ще има и възможност за оценяване на книга ( с оценки от  1 до 10).



**Фиг. 1**.*Хоум скрийн на потребитела показващ героите му.*

Възможностите за писане и четене на книги игри ще са услуги в реално време и потребителя ще може да вижда как играят или пишат другите. При писането ще има система за ограничаване на правата на пишещите от създателя на книгата.

Интеграциите със социалните мрежи ще дават възможност на потребителите за лесен достъп до системата, чрез вече съществуващи акаунти, за споделяне, харесване и други на книги от системата.



**Фиг. 1**.*Юзер скрийн за герой.*

Функционалностите за експортване на книги от системата ще позволява да бъдат създавани хартиени книги игри посредством системата , без почти никакво допълнително усилие от страна на автора. Благодарение на лесната система за създаване на граф на книгата, авторите ще бъдат силно улеснени в създаването на такава от жанра.

## Скорост, използваемост и поддръжка

Публичният уебсайт трябва да отговаря за под 500 милисекунди, и да изпълнява pageload най-късно след 1000 милисекунди от началото на заявката, като блокинг ресурсите трябва да са заредени до 700мс от започването на рекуеста. Ненужните в началото скриптове трябва да бъдат заредени асинхронно, както и възможно най-голяма част от съдържанието, особенно при четенето на книга трябва да се зарежда асинхронно.

Системата трябва да изпълнява “Златните правила на Schinderman”[2] за интеракшън на потребителя със ситемата. Уебсайтът трябва да е съобразен за хора със увредено зрение, да може да работи със скрийн ридъри и да спазва ARIA accessibility схемата[3].

Код-базата на системата се гарантира като лесна и удобна за поддържане като се спазват най-добрите практики и методологии при имплементация. Също върху кода се пускат автоматични тулове за определяне и при проблем справяне със цикломатичната сложност на кода, мейнтейнабилити индексът , както и халстидовата сложност.

## Надежност и сигурност на системата

Системата и обграждащата я архитектура трябва да поддържат минимален ъптайм от 99.97% , и да са съобразени със периодите на висока натовареност във пикови часове на ползваемост. За борба със DDOS атаки които се опитват да намалят този процент, **DDoS mitigation** система трябва да бъде ползвана или наета, поддържайки поне 1 ТB/s натоварване.

Изискванията към сигурността на системата включват освен усточивост на Denial of Service атаки , също и подсигуряване на това че системата не е уязвима на познати или непознати атаки, подсигурено със пенетрейшън тестове от поне един пенетрейшън туул, както и оглед от външен одитор по сигурността.

## Конкурентност, модифицируемост и тестируемост на системата. Стратегии за разширяване (scaling)

Уебсайтът и поддържащите го системи трябва да могат да бъдат дистрибутирани между множество ядра на физическите или логическите машини на които са сложени, както и самия уебсайт да може да бъде размножен на множество машини

## Оптимизация за търсачки, семантичност на съдържанието

Приложението още по време на своето проектиране трябва да бъде замислен с идеята да се намира по-бързо и по-лесно във всички известни търсачки. Search Engine Optimization (SEO) трябва да се постигне чрез добра оптимизация на HTML кода на страниците от сайта, както и чрез рекламирането и непрекъснатото увеличаване на броя на вътрешните връзки, както и на връзките, които водят обратно към сайта (backlinks). Понеже това е много важно за успеха на платформата, като такава, всичко това трябва да бъде включено в маркетинговата стратегия за развитие на сайта още от самото начало на процеса по разработката му. За целта се предвижда намесата и на външна помощ в лицето на хора, които спациално ще се занимават с този аспект от разработката на проекта, така наречените “SEO оптимизатори”.

За да се постигне всичко изброено до момента, доста голям набор от технологии и стратегии, трябва да бъде разгледан. За да е възможно най-лесен за откриване, нашият сайт трябва да използва редица стратегии, които да го издигнат по-нагоре в резултатите от търсенията, които евентуалните клиенти на системата биха използвали.

Една от основните техники ще бъде Sitemaps протоколът. Това е широко използван протокол, който всички по-големи Search Engine-ни използват за да разбират по-добре съдържанието на даден уебсайт. Този протокол работи, като използва специално създаден файл, който обикновено се нарича Sitemap.xml. Този файл трябва да съдържа в себе си всички URL-ли свързани със конкретния уебсайт. Освен това, Webmaster-ът може да добавя допълнителна информация към всеки един от тези URL-ли. Тази информация може да се използва от Search Engine-ите по време на процеса на web crawling. Тази добълнителна информация може да включва много неща, но сред основните характеристики, е например, кога съответната страница и била обновена за последен път, колко често тази страница се променя, колко е важна точно тази страница във връзка със останалите страници в рамките на уебсайта. Тези допълнителни насоки за search engine-ите са от първостепенна важност за нашето приложение, като мярка за по-лесното му достигане до хора, които евентуално биха могли да станат наши клиенти.

Друг много важен механизъм, който нашият сайт трябва да поддържа е така нареченията Robots.txt протокол. Става дума за протокол измислен още в зората на интернет през далечната 1994та година. Това е механизъм, който отново помага на search engine-ите ефективно да претърсват уебсайта ни. Основната му функция е да препоръчва определени директории от структурата на нашето приложение да не бъдат претърсвани. Това разбира се е само препоръка и някои engine-ни може и да не се съобразят с нея, но всички по-големи се съобразяват. По този начин, части от сайта, които нямат релевантна информация по отношение на търсенето ще бъдат прескочени и това ще помогне на търсачките да работят само с най-важните аспекти на сайта. Пример за директории, които трябва да бъдат включени в този списък са:

/admin/

/cms/

Подобно на Sitemap.xml, съществува и Robots.txt във root директорията на нашия проект. Въпросните пътища ще трябва да бъдат добавени в него. Освен това този файл ще оказва и местонахождението на нашия Sitemap.xml файл и така този механизъм ще помага експлицитно и на вече споменатия механизъм със Sitemap-а.

Освен това нашия сайт е доста динамично web-приложение. Това означава, че доста голяма част от функционалността, няма да зависи само от суровия HTML. Понеже книгите игри са доста динамични по своето естество, то сайтът който ги възпроизвежда също би следвало да съдържа доста динамично съдържание. За да може това съдържание да стане по достъпно, както за нашите клиенти, така и за всякакви search engine-ни, които изследват web пространството, ще трябва да се използва широко WAI-ARIA спецификацията.

Тази спецификация е сравнително нова. Официално е била публикувана от World Wide Web Consortium-а през март 2014та, като официална препоръка. Абревиатурата означава Web Accessibility Initiative - Accessible Rich Internet Applications. На практика става дума за начин за описване на значението на съдържанието на дадена web страница, както и на друга metadata. По този начин страници съдържащи динамично генерирано съдържание, стават по достъпни за потребителите, особено тези които имат проблеми като липса на зрение например. Използвайки тази спецификация е възможно да се опише един списък от линкове така, че лесно да се разпознава като навигационно меню. Освен това е възможно да се добави информация чрез която да се знае дали това меню е било отворено или останало скрито. Тази сематична информация е възможно да се добави директно като част от синтаксиса на езика, в който се вгражда. Такъв език е естествено HTML, но освен него също така и SVG. Възможно е също е и директното им вграждане в DOM дървото. По тази причина, тази спецификация трябва да е интегрална част от архитектурния дизайн на всяка една страница от проекта.

Друг начин за да се направи съдържанието на една страница по-достъпно за search engine-ни е използването на meta keywords. Тези тагове ще помогнат на отделните страници от сайта по-лесно да бъдат разпознавани от търсещите алгоритми. В последни години значението на тези ключови думи все повече намалява. Google вече спира да ги използва широко, понеже често се използват от недобронамерени сайтове като средство за измамно рекламиране на дадена страница като нещо, което не е. Въпреки това те все пак имат някакво значение, което не бива да се пропуска на днешния пазар, където конкуренцията никога не спи.

Освен keywords meta tag-ът, друг meta tag, който помага в тази сфера е description meta tag-ът. Този метатаг е много подобен на ключовите думи, с тази разлика, че тук трябва да имаме кратък текст, а не просто поредица несвързани думи. В тези тагове страниците от нашето приложение, ще трябва да опишат какво точно е тяхното предназначение и на кратко какво всъщност съдържат. Реално текстчето, записано в description meta tag-а потенциално може да се превърне в краткото описание, което се изписва под резултатът намерен от search engine-а. Това се случва, когато търсачката, не успее да създаде свое собтвено такова текстче, на базата на съдържанието на страницата. От това произлиза важността на текста написан в този таг. Много често евентуални клиенти биха отворили даден резултат от търсачката, защото са прочели именно това описание и то ги е заинтригувало.

Докато сме на тема мета тагове и много важно да споменем нуждата от използването на мета тагове за роботи в нашето приложение. Става дума за мета тагове, които идват като допълнение на Robots.txt протокола. С тяхна помощ може да се разшири още повече функционалността предоставена от Robots.txt файла. С помощта на тези тагове можем да специализираме по-добре поведението на роботите, които индексират нашия сайт, но вече на ниво отделна страница. Понеже нашия проект е свързан с книги игри, а основният начин за боравенето с книги игри, е чрез следването на линкове от една страница на друга, то тогава следва, че тази възможност със сигурност трябва да бъде използвана добре, за да се адаптират отделните компоненти на приложението по-правилно, към онова, което големите search engine-ни биха очаквали от нас.

Всички тези мета тагове са важна част от всяка една страница от сайта, както и от всяка една страница от книгите-игри, които ще се съдържат в него. Поради огромния обем на тези страници и възможността те да се създават динамично ще бъде добро решение да се създаде система, която автоматично да генерира подходящи мета тагове от тези категории, които правилно да отразяват нуждите на всяка една страница.

Разбира се е важно страниците да следват определени правила, когато става дума за динамично генерирано съдържание. Важно е да не се повтаря съдържанието на отделните страници. Това, което е част от една секция от книгата игра трябва по възможност да се преизползва, като се направят линковете да сочат динамично към него. Това би подобрило и търсенето, понеже би намалило дублирането на дадени секции, което от своя страна би направило търсенето по-еднозначно.

Въпреки това отделните страници трябва все пак да съдържат повече текст от колкото HTML тагове. Всичките тези мета настройки и организации не трябва да превишават самото действително съдържание на страницата, което реално ще бъде основния обект на изучаване от страна на search engine-ните. По-големите търсачки все пак разглеждат от първостепенно значение самото съдържание на дадена страница.

Понеже голяма част от съдържанието на нашия сайт ще бъде генерирано от потребителите му, е много важно да се използва специалния таг за авторство използван от Google. Става дума за “ rel=”author” “ атрибутът, който можем да закачим за даден href. Сайтът ще предоставя възможността генерираните страници да могат сами да идентифицират потребителите, които са ги създали. По този начин, ако някой клиент по някаква причина е известен, всички търсения за неговото име в мрежата евентуално биха довели търсещия до негоната творба, която се помещава на нашия сайт. С други думи освен, че това е добър начин за потребителите ни да стават известни с творбите си, но освен това и сайтът ни да става известен чрез известните си потребители. Разбира се тази възможнст трябва да остане опционална, за да се даде възможност на клиентите да избират какво да споделят за себе си и да запазят правото да третират творбите си като конфиденциална информация.

Освен Google, друга голяма компания, която предлага поддръжка за специалицирани HTML тагове е Facebook. Те предлагат така нареченият Open Graph протокол, за свързване на отделни страници в социалния граф на своите протребители. Това се реализира с помощта на поредица нововъведени meta тагове, които определят даден документ като част от социалният граф, който се поддържа от Facebook. По този начин отделните страници от творенията на нашите клиенти, могат динамично да се свържат и с техните Facebook профили, което от една страна облагодетелства клиента, но от друга, също както беше примера със authorship тага на Google, облагодетелства и нашия проект понеже разширява начините, който могат да бъдат използвани за да се достигне до нашия web сайт.

## Анализ на поведението на потребителите, AB тестване, профайлинг, Баунс рейтове и статистики

Важна част от нашата бизнес стратегия е анализът на поведението на нашите потребители. Както всички по-големи играчи на пазара ние сме в състояние да наблюдаваме как нашите клиенти използват сайта ни, какво правят в него, какво създават, и от какви теми се интересуват. Всичката тази събрана информация ни помага да определяме динамично нашата бизнес стратегия, така че тя да отговаря максимално на нуждите на нашите клиенти. Тази способност да се нагаждаме, към ситуацията според текущите данни, които получаваме е ключова за успеха на нашата платформа.

Освен това друго преимущество, което системите ни за анализ на поведението на потребителите ни предоставят, е фактът че разполагаме с голяма база данни за нашите клиенти. Тази голяма база данни от систематизирана информация, може да служи както на нас, така и на евентуални наши партньори, които желаят да се възползват от нашите познания за нуждите на нашите клиенти. Разбира се това е и един допълнителен източник на приходи, който може да помогне за бъдещото разрастване и развитие на нашата онлайн платформа за създаване на книги игри.

Реално погледнато вместо да използваме готово решение за анализ, ние имплементирахме сами такова. Събираме информация от нашите клиенти като прекарване всеки request от тяхна страна през специализилана система, за събъране на информацията. По-късно тази информация се използва за анализ.

Прави се AB тестване, профилиране, bounce rate-ове и се изготвят статистики на базата на получената информация.

# Качествени атрибути

Това са атрибутите, по-който е притео всяка архитектура да бъде оценявана и който са смятани за основни свойства на една такава. В този раздел ще разгледаме кракто изяснене на същноста на всеки от тях, както и сценарии при който те са от значение. Ще бъде разгледано още какв механизми се използават от разработената до тук архитектура за подобряване на всеки от атрибутите

## Достъпност (availability)

### Дефиниция

Достъпността представлява способноста на системата да предоставя услугата, за която е  създадена. Или с други думи  достъпността се свързва с вероятността, когато даден потребител се опита да достъпи системата, тя да не е на разположение. В теорията тази вероятност се пресмята като  вероятноста за спиране на системата разделена на вероятноста за спиране на системата + времето за връщане на системата в режим на нормално фунциониране.

### Сценарии

#### Сценарий 1

##### Artifact - възможността за системен рестарт и за нотивикация на админи

##### Source - вътрешен за системата

##### Stimulus - срив

##### Environment - при нормална операция

##### Response - системата ще опита да се рестарира, освен това ще изпрати емайли до всички отговорни за подръжката ти. В случай на невъзможност за рестартиране ще подаде сигнал към сървърите за разпределение на трафика , за да се пренасочат всички потребители към друга работеща инстаниця.

**Measure** -  Времето, системата ще бъде недостъпна ще е в рамките на няколко секунди ( под минута)

**Цялостен сценарии:**

    Потребител влиза в системата, която крашва поради вътрешни за системата причини. Потребителя получава съобщение за грешка и  при следващия си опит бива пренасочен към друга инстанция на системата,докато крашналата инстанция се рестаритра, или проблема в нея бъде отстранен от известените с емайл админи. Времето, в което системата е неактивна за потребителя е в рамките на една заявка.

#### Сценарий 2

**Artifact** - възможноста на системата при реакция ако дадена инстанция на приложението не отговаря

**Source** - вътрешен за системата

**Stimulus** - сървър не отговаря

**Environment**  - при нормална операция

**Response** - Ще подаде сигнал към сървърите за разпределение на трафика, за да се пренасочат всички потребители към друга работеща инстаниця.Ще се известят всички администратори за проблема

**Measure** -  Времето, системата ще бъде недостъпна ще е в рамките на няколко минути ( под 10)

**Цялостен сценарии:**

    Потребител опитва да достъпи системата, която не отговаря. Потребителя получава съобщение за грешка и след минути когато сървъра следящ чрез chrontab на всяко 10 минути изправността на сървърите, установи проблема, потребителя ще бъде пренасочен към друга инстанция на системата,докато проблема в нефунцкиониращата инстанция бъде отстранен от известените с емайл админи. Времето, в което системата е неактивна за потребителя е в рамките на няколко минути ( под 10)

### Решения на системата

За да се повиши достъпността до системата се използва силно развита физическа архгитектура, която разпределя потребителите към подходящи и работещи инстанции на приложението, и в случаи на проблем реагира максимално бързо пренасочвайки потребителите към работеща инстанция.Вероятноста за едновременен провал на няколко инстанции и съответно провал на цялата система е считан за практически невъзможен.

Системата също подържа машини за следене на изправносттта и достъпността на дадени сървари, такаче максимално бързода бъдат остановени евентуални проблеми, преди да бъдат усетени от потребителите.Локално всяка инстанция разчита на модула forever,  която при евентуални крашове в системата да я рестартира и да локализира проблемите , изпращайки съответните нотификации.

## Бързодействие (performance)

### Дефиниция

Бързодействието на системтата представлява бързината, с която тя работи и се характеризира с времето което даден потребител чака за да използва различни функционалности на програмата.

### Сценарии

#### Сценарий 1

**Artifact** - тестване за времето на отговор на системата

**Source** - външен за системата

**Stimulus** - произволни евенти създадени от клиента

**Environment**  - при нормална операция

**Response** - Системата ще използва кешираните ресурсни и ще отговори със максимална скорост изпълняваща изискванията

**Measure** -  Времето, в най-лошия сценарии , което ще е времето за което системата зарежда основните ресуриси за показване на страница.

**Цялостен сценарии:**

Потребител влиза в системата, изискваики произволни ресурси. Системата използва функционалностите си за кеширане и доставя голяма част от информацията бъро на клиента.В случаи че тези системи се окажат непотребни потребителя изчаква зареждането само на основните ресурси за страницата, докато останалите се получават в последсвтие благодарение на асинхронния модел на работа на приложението.

#### Сценарий 2

**Artifact** - възможността на системата за работа с бавни алгоритми и предоставянето на ифнормация на клиентите.

Source - вътрешен за системата

**Stimulus** - еденичен или повтарящ се евент изискващ голямо количество изчислителна мощ

**Environment**  - при претоварващо операции

**Response** - Системата ще извика машинен модул на по ниско ниво за да подобри представянето на алгоритъма. Този модул ще работи асинхронно от системата и няма да влияна работата и, а когато проключи резултата от него ще бъде отново зареден асинхронно

**Measure** - тук времето, за изчакване ще бъде сведено единсвено до небходимиото извършване на операцията

**Цялостен сценарии**:

Системата извършва дейстиве, което е е време консумиращо. Модулите за извършване на такива операции, създават свой, процес независим от останалата логика на приложението, който създава евенет и връща информация когато приключи. По този начин работата на системата не се влияе от предпиремането на операции нуждаещи се от голямо количество време.

### Решения на системата.

За да се сведе до максимум бързодействието на системата са предприети множество стъпки. Първата такава е осигуряването на достатъчно добра физическа архитектура. Втората е използването на асинхронен модел за приемане на зявките и обработването им без значение от броя и сложноста им. Това става чрез пускането на различни процеси когато е необходимо, и чрез употеребата на езици и фунцкионалности от ниско ниво за постигане на желания ефект.

## Сигурност (security)

### Дефиниция

Сигурността е свързана с качествта на системата да предовратява неоторизиран достъп и да издържа на злонамерени вмешателства и атаки.Сигурноста се разделя на две нива - предодрватяване на нанасяне на щети и възтановяването на системата след нанесени такива.

### Сценарии

#### Сценарий 1

**Artifact** - тестване за системите за сигурност на системата.

**Source** - външен за системата хакер

**Stimulus** - Опит за намаляне на достъпноста на системата

**Environment**  - при нормално фукнционираща система.

**Response** - Системата ще използва механизмите си на защита и ще ограничи възможностите на атакуващия, правейки опитите му за атаки невъзможни или струващи, твърде много време че да са смислени на пракитка.  **DDoS protection Service-а** ще използва автоматичното си засичане и до няколко минути трафика от IP-адресите от които атаката се извършва ще се филтрира.

**Measure** -  Крактосрочно забавяне на представянето на някоя инстанция на приложението, което ще бъде предодвратено от механизмите за сигурност  на приложението за по-малко от минута

**Цялостен сценарии:**

Хакер влиза в системата и опитва да предприеме DoS атака. Системата реагира с ограничаване на броя възможни заявки, изискване на валидация чрез капча, а при претоварване на някоя истанция заявките на атакуващия се разпределят между случайни други инстансции на приложението, което на практика води до неефективност на атаката. Също **DDoS protection Service-а** който притежава bandwidth от десетки хиледи терабайти филтрира лошия трафик преди да го прехвърли към системата на swordsAndBooks.

#### Сценарий 2

**Artifact** - тестване за системите за сигурност на системата.

**Source** - вътрешен за системата хакер

**Stimulus** - опит за кражба , променяне или унищожаване на данни от базата с данни.

**Environment**  - при нормална работа на системата

**Response** -  Ако по-някакъв начин бъдат преодоляни защитите на сървърите и нанесени щети там, данните могат да бъдат лесно възвърнати от специалните бек ъп сървари, за който достъп ще имах хора,  който ще са с ограничен до базите от данни достъп.

**Measure** - Евентуална промяна в данните е практически невъзможна, заради доброто разпределение на роли и права за достъп на работещите по проекта. Дори при еветуално нанасянае на щети по данните на приложението, множеството бек ъп сървари безпроблемни ще възстановят щетите и от данните в тях лесно ще се установи, кой е бил злонамереният вътрешен хакер

**Цялостен сценарий:**

    Служител или служители решават да откраднат и / или изтрият базата от данни на приложението. Поради мерките за предовратяване на такъв вид деиствия възможностите им за нанасяне на сериозни щети са силно ограничени, а възможностите за бързо възстановяване на нанесените щети са достатъчно големи, такаче системата практически да не пострада от атаката

### Решения на системата.

За да се сведе до максимум сигурността на системата са изпозвани множество средсва във физическия и софтуерния слой на архитектурата.Изпозването на ограничаването на физическия достъп до важните за проложението архитектурни компоненти, както и разделянето на различните сървари на различни физически места осигурява голямо ниво на сигурност.Също така воденето на логове при софтуерен достъп на важни части от системата, и силно ограничените права на работещите по системата, и разделянето им на конкретни роли с конкретни отговорности прави атаките от вътрешни служители практически невъможен.

За защита от външни атаки са изпозвани множество защитни технологии, и технологии за редовно записване на статуса на инстанциите с цел премахване на нанесени щети.  Интегрирането на софтуери за ограничаване на заявките, както и на софтуери за защита от роботизирани атаки ( ингетрирането captch - и и други), както и системата за разпределяне на ролите на потребителите осигурява достатчно добри ниво на защита на системата

## Модифицируемост (modifiability)

### Дефиниция

Модифицируемостта на системата описва възможноста за промени и разширяване на системата, както в пракитката тази характеристика се измерва с времето, парите и усложненията,  които биха настъпили при необходимост от промяна на системата.

### Сценарии

#### Сценарий 1

**Artifact** - Тестване на възможностите на системата за добавяне и променяне на фунцконаллност

**Source** - Програмист работещ по системата

**Stimulus** - добавяне или променяне на фунционалност

**Environment**  - при създаване или промяна на фунционалност(разработка)

**Response** - създаване на промени или нови фунционалности в разумно време

**Measure** -  Цената във време пари и усилия ще бъде минмална.

**Цялостен сценарий:**

    Програмист работещ по системата решава да създаде нова функционалност или да промени вече съществуваща такава.Усилията му за разработка са силно намалени от софтуерите за Програмист работещ по системата решава да създаде нова функционалност или да промени вече съществуваща такава.Усилията му за разработка са силно намалени от софтуерите за Програмист работещ по системата решава да създаде нова функционалност или да промени вече съществуваща такава.Усилията му за разработка са силно намалени от софтуерите за помощ на разработката, а разпорстраняването на промените става изцяло автоматизирано , след удобрението им.

#### Сценарий 2

**Artifact** - тестване на възможноста за създаване на нова инстанция.

**Source** - вътрешен за системата администратор

**Stimulus** - създаване на нова инстанция

**Environment**  - при нормална работа на системата

**Response** -  създаването на новата инстанция ще бъде постигнато в разумно време и без значителни усилия.

**Measure** - цената във време усилия и пари ще бъде сведена до минимум

Цялостен сценарий:

Администратор на системата решава да създаде нова инстанция на приложението. Това е би му се отдало лесно поради изпозването на модлулите като npm и bower, който се грижат автоматично за инсталирането компонентите на системата.Благодарение на  git пък кода на системата може да бъде поставен на машина посредством една команда.

### Решения на системата.

Добрата топология на физическата аргитектура на системата позволява на работещите по нея да добавят,променят  и премахват компоненти от нея без това да се отрази на цялостното поведение на системата. Инстурментите като npm, bower git, и други позволяват максимизиране на простотата на процеса по разширяване и променяне на логиката на приложението.

## Възможост за тестване(testability)

### Дефиниция

Възможноста за тестване е характеристиката която се свързва с възможността на системата да бъде изпитвана лесно и пълноценно ( тоест фукнционалностите и бъдат тествани цялостно и ефективно)

### Сценарий

#### Сценарий 1

**Artifact** - Тестване на цялостната фунционалност на модулите и.

**Source** - Тестер работещ по системата

**Stimulus** - пускане на диагностика на системата

**Environment**  - при създаване или промяна на фунционалност(разработка)

**Response** - лесно и ползотворно пускане на тестове(отркиващо проблеми).

**Measure** -  Цената във време ще бъде минмална.Ще бъде постигнат голям процент покритие на тествания код.

**Цялостен сценарий:**

    Тестер работещ по системата решава да пусне тестове. Благодарение на автоамтизирането на системата за тестове това става бързо и осигуряв добри резултати.

#### Сценарий 2

**Artifact** - тестване част от ситемата

**Source** - Тестер работещ по системата

**Stimulus** -  пускане на диагностика на част системата

**Environment**  - при нормална работа на системата

**Response** -  лесно и ползотворно пускане на тестове(отркиващо проблеми).

**Measure** - Цената във време ще бъде минмална.Ще бъде постигнат голям процент покритие на тествания код.

**Цялостен сценарий:**

        Тестер работещ по системата решава да пусне тестове. Благодарение на автоамтизирането на системата за тестове това става бързо и осигурява добри резултати

### Решения на системата.

    Изцяло автоматичното генериране и пускакане на тестове предоставя високи резултати в тази качествена оценка на системата.

## Използваемост и удобство на дизайна (usability)

### Дефиниция

Използваемостта и удобството на дизаина е качеството, което описва удобността на приложението за крайния потребител. Това е водещо качество за нашето приложение,  защото в него големия брои потребители е от основно значение.

### Сценарий

#### Сценарий 1

**Artifact** - Тестване на удобството на системата.

**Source** -Тестер работещ по системата

**Stimulus** - научаване на фунционалностите на системата

**Environment**  - при нормална работа на системата

**Response** - ясни титорияли, който са не задължителни и лесни за  откриване.

**Measure** -  Подорява познаняита на потербителя за работа със системата.

**Цялостен сценарий:**

    Потербител се регистрира в системата и желае да се научи как да я използва максимално пълноценно

#### Сценарий 2

**Artifact** - Тестване на удобството на системата.

**Source** -Тестер работещ по системата

**Stimulus** - максимална улесненост  на потербителя.

**Environment**  - при нормална работа на системата

**Response** - функционалности за гласова навигация , четене на епизоди от книги, и изключително опростен дизаин за работя на потребителите

**Measure** -  Подорява преживяването на потербителя при работа със системата.

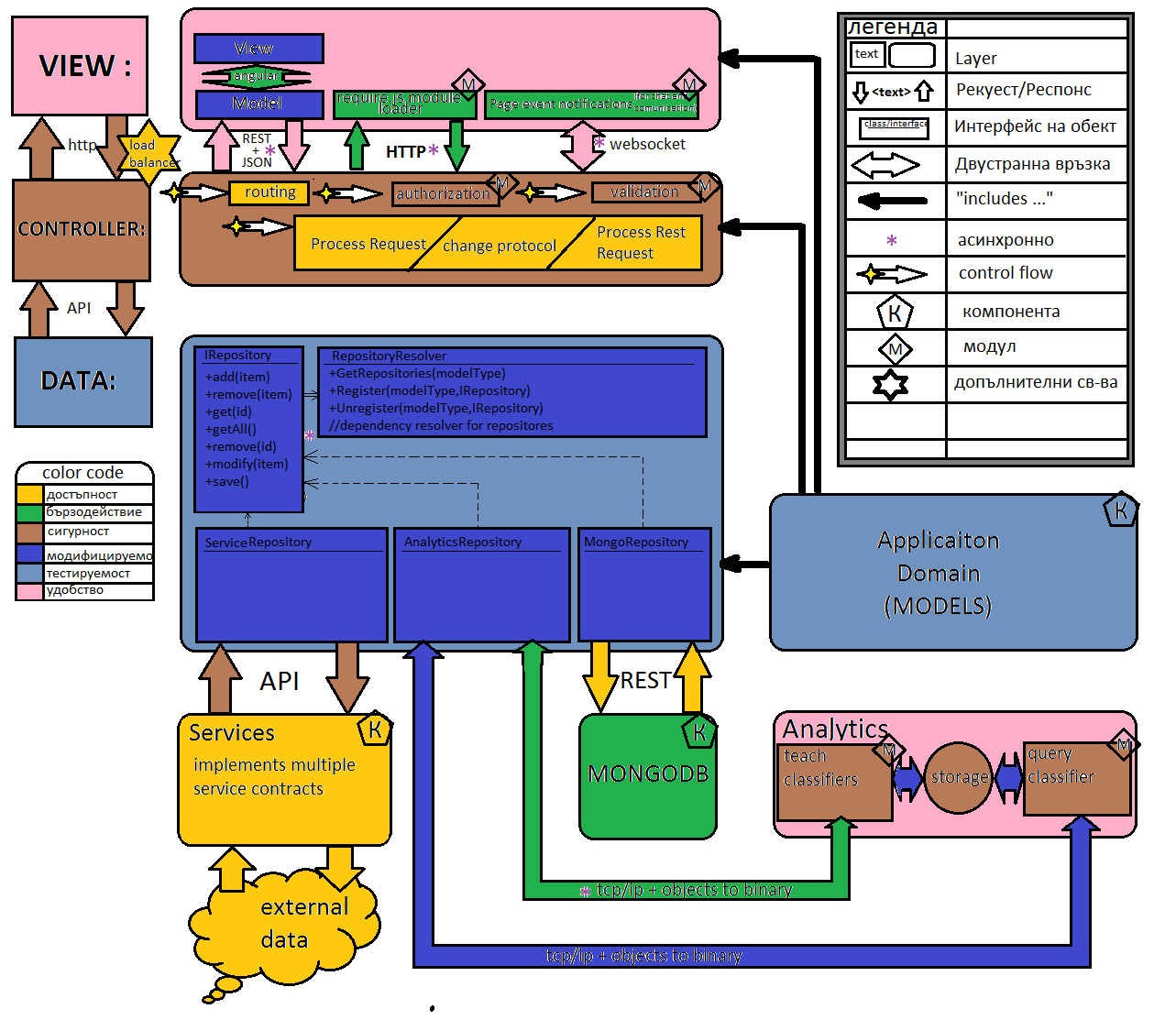
**Цялостен сценарий:**

    Потербител се регистрира в системата и желае да се изпозва системата максимално ефективно

### Решения на системата.

Много опростения дизайн, тестван на случаино подбрани потребители  и създаден посредством методологиите за асемблерово учене ( създаване на дизайн, който да удоволетворява потребностите максимален на максимален брой потребители, като в същотот време не включва никакви излишни фунцкионалности(намиране на минилания дизайн изпълняващ максиамлно количество потребителски нужди )) спомага за потребителските преживавания.Интегрирането на технологии от ново поколение, като фунционалност за четене на книги и употреба на приложението чрез гласови командни, прави приложението достъпно и приятно за всеки човек.

# Архитектура



**Фиг. 1**.*Цялостната софтуерна архитектура на приложението*

## Софтуерна Архитектура

Модулите във системата са разделени на множество слоеве, всеки от които е интерфейсна част и може да бъде заменен без промяна на останалата част от софтуера. За това се ползва комбинация от geddy, MVC (Model-View-Model) бекенд NodeJS фреймуорк,Angular - фронтенд Javascript MVVM (Model-View-View-Model) фреймуорк, който спомага за асинхронно зареждане на данните , самите angular контролери се зареждат асинхронно и зареждат само нужните си модули посредствум модуларна система чрез Require.js. Контролерите не отговарят директно за достъп на данни и за по-сложната бизнес логика , тези неща са изнесени в отделни business layer , data layer и при нужда от достъп на външни данни те са абстрактнати чрез още един слой - service layer-a. Data layer-a също комуникира директно и със системите за обработка на статистически данни , анализ на поведение, класификация и data mining. При разработката за улеснение да разработчиците и за подобряване на качеството на кода се ползва софтуер за тестването на цикломатичната сложност на кода , както и инструменти за автоматична минификация и бъндлинг на javascript и css файлове. Използват се също и множество билд процесори, за улесняване създаването , и подобряване поддръжката на css и javascript са конфигурирани пре-компилатори и за двата езика за фронтенд часта. В случая на CSS това е SASS , а при javascript - Coffeescript. Изображенията са също автоматично минимизирани във web-safe големина. За дефинирането на билд процесите се ползва Grunt framework-а. Тъй като се изисква работа с множество външни javascript и css framework-ци, се ползват bower както и npm пакидж мениджъри за лесна поддръжка на зависимости oт други свързани фреймуорци. Написан е адаптър от npm към require.js , като част от Grunt , така че да може лесно да се зареждат зависимостите на външни фреймуорци.

### Controller layer

Това е слоя в архитектурата на приложението , който отговаря за комуникацията на останалите слоеве. Основните задачи за контролерите тук са да се грижат за правилното разпределение на работата и да осигурават висока ефективност и надежност на системата.Тук са и всички проверки за автентикация и оторизация на потребителите, на системата, а също така и приемането на заявки и разпределянето им към конкретните модули.

Сигурността тук се гарантира чрез множество от инструменти за целта.Системата разчита на автентикирали се потребители, чрез акаунт в системата или акаунт в познатите социални мрежи (google + , facebook , twitter).

В последствие системата запазва в сесията си необходимата информация за потребителя и правата му и преди всеки request извършва проверки, чрез който да установи дали всичко е наред със заявката и дали потребителя има достатъчното ниво на доверие от системата че да предприеме такова дейстивие.

Имплменетирана е също така защита от foreign атаки. Това са атаки, който разчитат на това че потребителя има сесия в системата и го подвеждат да изпълни заявка ,за която той не подозира. Това може да стане чрез кликане на линк, отговаряне на съобщение и дори чрез разглеждане на снимка. Тук контролерите инплементират стандатната защита от такъв тип атаки, а именно отговаряне на всяка заявка те изпращат на клиента уникален ключ. Когато се прави следваща заявка клиента изпраща получения ключ, и след проверка на сървъра заявката се изпълнява само в случай, че изпратеният и полученият ключ са съвпаднали.

Първия слой на приложението, с който комуникират контролерите е дата слоя. Комуникацията тук се изпълнява всеки път когато системата се нуждае от информация за някаква част от логиката си. Благодарение на лесното използване на асинхронни заявки към този дата слой чрез фунционалностите на node js и билилиотеки като async и promise контролера може паралелно да отправя зявки за информация, а благодарение на event модела може да работи с резултатите веднага щом са готови. Това подобрява представянето на системата в пъти, защотото изцяло премахва изчакването между отделните заявки за информация. Предвид динамичната същоност на приложението event модела  комбинран с високата скорост на базата от данни ( mongoDB) осигурават голяма част от бързодеиствието необходимо на системата.

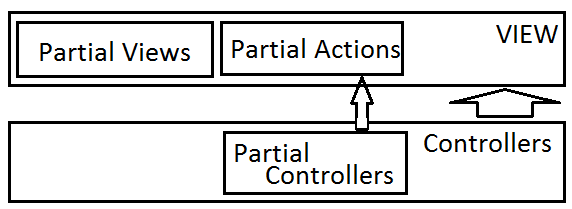
Втория слой на приложението, с който контролерите комуникират е визуалния слой.Тук контролерите взимат всичката необходима информация и я препращат кум визуализационната имплементация. Важни решения тук, които трябва да се вземат са, коя информация да бъде изпратена преди зареждане на страницата и коя, да се изпраща чрез асинхронния модел. Прилагания подход тук е да се определи някакъв минимален сет от необходима информация за зареждане на страницата,  и когато тя е извлечена от дата слоя, да се започне зареждане на страницата, а всичко останало да се изпрати чрез асинхронния модел.Това намалява времето за отговор изключително много, и позволва на системата да хаби минилално количесвто сървърни и клиентски ресурси.

Последния слой свързан с контролерите е web API слоя. API слоя тук изисква от контролерите дадена информация посредством HTTP заявки, и след проверка на подходящите права контролерите извличат тази информация от дата слоя, преопразуват я в подходящия формат желан от заявката( JSON или XML) и я изпращат на web API-то. Тук отново, контролерите се яват арбитър  правата за получаване на информация. Кракто обощение за работата на контролерите ще дадем в следната таблица:

|  |  |
| --- | --- |
| СЛОЙ НА  КОМУНИКАЦИЯ | ИЗВЪРШВАНИ ДЕЙНОСТИ ОТ КОНТРОЛЕРИТЕ |
| **Дата слой** | Определяне на търсената информация и поискването и. |
| **Визуалния слой** | Определяне на правта и предпазване от атаки. Изпращане на подходящата информация, чрез асинхронен модел. |
| **Web API** | Определяне на правта и предпазване от атаки.  Изпращане на подходящата информация в поискания формат |

**Таб. 1**. *Действия извършвани от контролерите при комуникацията им с другите модули на системата.*

### VIEW

**

**Фиг. 2**. *Подробна диаграма на клиентския слой.*

Вю слоя съчетава в себе си клиентската страна на приложението. Тук са събрани визуализацията на приложението, клиенската логика и функционалност.Също така има и няколко функционалности за работа със външни системи минаващи само пред клиентския слой. Кракто обощение на фукнционалностите на клиента може да се намери в следната таблица:

|  |  |
| --- | --- |
| СЛОЙ НА  КОМУНИКАЦИЯ | ИЗВЪРШВАНИ ДЕЙНОСТИ ОТ КОНТРОЛЕРИТЕ |
| **ejs views** | *Създаване на базов HTML код и защита от XSS атаки.* |
| **require.js** | *Динамично зареждане на клиентската логика и оптимизация на необходимите ресурси.* |
| **Фронтенд модели** | *Обработване на информацията и създаване на финални визуализации.* |
| **Фронтедн контролери** | *Асинхронно изискване на информация от сървъра, показване на готовите модели и контролиране на необходимите клиентски логики* |
| **Фронт енд view-та** | *Показване на готовите модели и клиентската им логика.* |

**Таб. 2**. *Описание на функционалностите на всеки от слоевете на клиентската архитектура.*

#### ejs views

Това е слоя,  който комуникира директно с контролерите на бекенда на приложението. Този слой сформира основния изглед на страницата,и се грижи за ескейпване, на който да бъдат показвани такаче да се предовратят всички опити за XSS атаки. Когато този слой бъде зареден, той  подава информацията си на фронт енд моделите, който да я заредят на клинета с подходящи стилове и допълнителна фунционалност.

#### require.js

require.js е фунционалност, позволяваща зареждането на клиентски код, само когато е необходим. Чрез този модул се зареждат ресурси на клиента асинхронно и само когато те са необходими. Този подход има няколко предимстрва. Първото такова е подобряването на скороста на клинетската страна от приложението, което води до много по-добро взаймодеиствие с потребителите и подобрява цялостното изживявяне при употреба на системата.

Второто предимство предоставяно от този модул е възможноста за зареждане на модули асинхронно, което прави първоначалното показване на страницата много по бързо, и свежда трафика по мрежата и загубата на усезамо от потребителите време за зареждане на ресурси до минимум.

    Третото предимство е , че употребата на този модул предоставя възможност за изключителна лесна работа с и създаване на код, което значително увеличава възможносите за разширяване на системата, както и за създаването на тестови еденици.

#### Фронтенд модели

Фронтенд моделите са модула, който директно отговаря за генерирането на HTML кода и визуализацията на системата. Този слой получва готовите вюта от ejs  модула и използваики заредените от require.js файлове сглобява данните и функционалностите, който да бъдат показани на потребителя.

#### Фронтедн контролери

Фронтенд  контролерите са тези, който се грижат за връзката между сървъра и клиента, както и между отделните копоненти на клинетската архитектура.

Основното задължение на контролерите е извикването на асихронни заявки към сървър страната за получаването на нови данни. Факта че тези извиквания са базирани на система от събития позволява по голямо бързодействие, както и по голяма интуитивност при разработка, водеща до по-добри възможности за разширяване на системата.

Друга функционалнот на този подслой е да следи за необходимост от промята във фронтенд view-тата и да предприема необходимите действия, когато има такава.Тук отново благодарение на модела на събитията системата постига максимално бързодействие.

#### Фронт енд view-та

Това е часта, в която приложението се изобразява.Тук има две основни функционалности извършвани от този подслой.Тук се извършват всички визуализации на HTML , canvas, и други библиотеки и тяхните клиенткси пресмятания и логики.Също така тук се предприемат потребителски контролираните промени по домейн дървото на приложението.Друга функционалност на този слой е оправянето на директни заявки към външни сървиси като например web API-то за четене на теск на google, чрез което всеки потребител ще може да слуша книгите вместо да ги чете. Това става чрез model-view-view-model архитектурата на angular , промени по модела , ъпдейтнати асинхронно директно въздействат на показаното във view-то, а промени на мап-нати данни във view-то променят модела.

### MODEL

Във проекта се използва методологията за POCO (Plain old CLR Objects) модели, които лесно могат да бъдат сериализирани , във нашия случай до JSON , тъй като по-голямата част от изпълнимия код на приложението е написан или компилиран до javascript, заедно със ajax API-тата и MongoDB базата данни на проекта и това улеснява преноса на данни между слоевете и тяхното използване.

Домейна е изграден от книги които потребителите могат да четат или да пишат, всяка една от тях е изградена от епизоди които имат множество разклонения. Взаимно тези три ентитита имат статус във книгата който се ползва от инстанциите на уиджетите на конкретния епизод за да определи какво да прави след това. Дефинициите на уиджети са всъщност пойнтъри към view-та които се ползват за да се рендерира дадена негова инстанция заедно със текущия статус във даден епизод.Всеки модел има методи, който се извикват асинхронно и позволяват бърза работа с данните в модела както и изпълняване на join и по сложни зявки,който да не натоварват представянето на системата. Моделите комуникират със останалите части на ситемата посредством дата слоя на проложението, който е реализиран чрез ORM написан на javascript и реализиращ максимална производителност при комуникацията със останалите слоеве.

#### Widget Model

Модела за уиджет представлява съвкупноста от дефиниции на различни уиджет . В същност всеки уиджет ще притежава свой уникален модел, койтода съдържа необходимата информация за изобразяване и работа с данните на конкретната уиджет инстанция. Така един път зареден моделът на всеки конкретен уиджет ще се грижи асинхронно за работатана свойте view-та, и по този начин системата няма да бъде зависима от свързаноста на компонентите си и дори някои от уиджетите да изпадне в състояние на грешка това няма да се отрази на работа на цялостната система, и ще бъде бързо остранено от процесите стартирани от контролерите,  за контрол и отстраняване на грешки в уиджет системите.

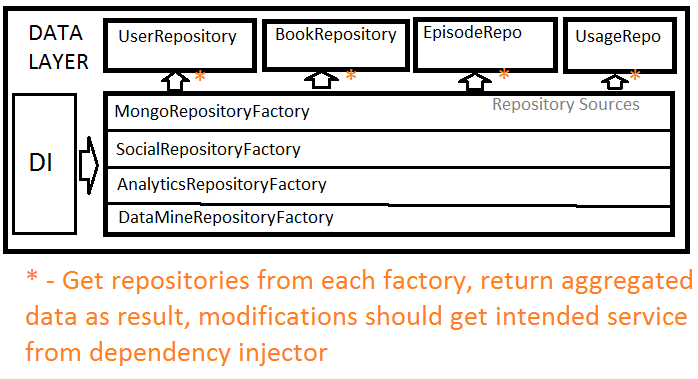
#### Books Model

Модела на книгите, ще съдържа в себе си обширна информация и  имплементация на множество алгоритми като **iterative deepening search** за намиране на крайща на книгите, както и за търсене на конкретни епизоди.Също така ще съдържа, информация и алгоритми за **статистическо учене**, с цел спомагане на потребителя и предвиждане действията му, при играене или писане на книга. Тези модели също така ще държът множество  връзки към епизоди, потребители и други нужни на системата за да фунцкионира.Връзките тук ще бъдат реализирани посредсвом стандатните за mognoDB външни ключове под формата на id -та на съответните обекти. Свързващата таблица между потребител и книга статус тук ще запазва прогреса на потребителя в дадена книга. Имплементирани са също възможности за промяна на данните в статус от chrontab процеси, чрез който да се постигне максимален реализъм в системата.

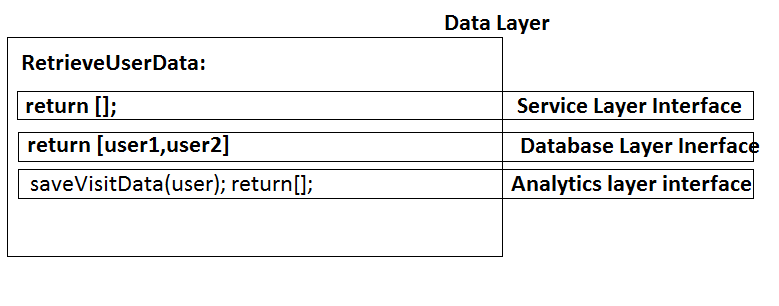
#### User Model

Потребителския модел, съдържва в себе си информация за потребителя, проверки за права, както и информация използвана от изкустения интелект на системата за предвиждане предпочитанията и интересите на потребителя. Изготва се **decision tree,** което се пази в модела на всеки потребител, чрез което се предвиждат дейстивята му във различни ситуации на употреба на системата.Тук се пазят множество асоциации към други модели, защото водещата същност на системата са потребителките модели.

### DATA

**

**Фиг. 3**. *Подрбна диаграма на дата слоя.*

Data-layer-а служи като абстрактен интерфейс за достъпа до данни от различните под-системи, бази данни и сървиз нива. Базата от данни която се достъпва от този слой е базирана на **MongoDB**, no-sql база от данни, отделно се ползва **Memcache** сървер за запазването на сесиини данни на потребителите, за да може по-лесно да се диструбитира системата. Дейта лейъра предоставя интерфейс за достъп до данни които са дошли от външни сървиси , чрез service layer-а , както и до системите за анализ и data-mining на данни. Data layer-а работи като chain of responsibility като всяка заявка за данни се обработва от множество слоеве които могат да върнат или да не върнат данни. Това спомага за абстракването на логиката за взимане на данни, както и за data mining и analytics на такива. Интерфейсти за тези две нива съществуват като бримки от веригата на отговорност, които обработват данните при получаванет им (например извличане на епизод от книга при даден потребител).

**Фиг. 1**.*Архитектура на Data Layer-а.*

### SERVICE Layer

Това е слоят,който свързва системата ни , със външни източници на данни. Такива източници могат да бъдат API-тана външни системи, като тези насоциалните мрежи,  от където може да се взима информация за потребители искащи да се автентикират със акаунитите си там в нашата система. Може също да се събира информация относно харесванията и интересите на потребителите за да може алгоритмите на системата да се обучават пълноценно и да прилагат максимално интелекта си към потребителите.

Други източници на данни са биха могли да бъдат и услуги предлагащи новини и последни информации относно тенденциите в жанра на книгите игри зада можеда бъдат интегрирани тези знания в системата.

Не на последно място са и сървисите за обучаване на системата, като например интефейси за взимане на стандатрни текстове за обучение на **алгоритмите за синтактичен анализ** на ситемата посредсвом,който ще могат да се генерират автоматично резюмета на книги и епизоди, както и препоръки към пишештите потребители.

Основна задача на SERVICE Layer-а е да привежда данните в разбираем за системата формат. Данните, който присигат от външни източници е възможно да пристигат в множество различни формати, но след обработка достигат до системата ( и по спецялно до дата слоя ) само във JSON или XML формат.По този начин се спазва изцяло принципа за толерантност на системата към външни източници на информация.

#### Логин като сървиз

Тук ще разгледаме,автентиткирането и оторизирането на потребители посредсвтом външни сървиси, като социалните мрежи и други. Порцедурата е стандатна и е реализирана от модула на системата passport.js. Това, което се случва тук е изпращането на заявка към съответния сървис, която казва, че някой от нашата система иска да се автентикира чрез съответната социална мрежа. Тогава социалната мрежа предлага на потребителя външен от нашата система, интерфейс за въвеждане на име и парола.След успешен логин системата ни получава данни за потребителя, привежда ги в подходящ формат, след което запазва необходимите права и информация в сесията и позволява на потребителя да използва функционалностите на системата.

##### Фейсбук

Логиът със фейсбук е реализиран чрез модула facebook-passport.js, който е часто от passport.js. Сървисите на фейсбоок предоставят API-та както за получаване на информацията в  JSON , така и  XML формати. Тук системат ани получава потребителски имена, и други допълнителни данни от социалната мрежа, който се интегрират и използват за автентикиране и оторизиране.

##### Туитър

Логиът със фейсбук е реализиран чрез модула twitter-passport.js, който е часто от passport.js. Сървисите на Туитър предоставят API-та както за получаване на информацията в  JSON , така и  XML формати. Тук системат ани получава потребителски имена, и други допълнителни данни от социалната мрежа, който се интегрират и използват за автентикиране и оторизизране.

#### Външен сървис за четене на текс.

Сървиса представлява функционалност за четене на текс, с цел да се услесни употребата на приложението. Този сървис се използва посредством API осигурен от google, към който се изпраща текста и мета данни към него в JSON формат, а отговора е получен под формата на звуков файл-резултат от прочитането на текста.Този звуков файл се пуска при клиента директно при получаването му след което бива изтрит, което реално не изисква памет нито на сървъра нито на клиента на приложението. Благодарение на множеството функционалности и добрия дизайн на google API-то за четене на текст, тази функционалност изсиква минилана скорост на интернет връзка и може да чете разбираемо  на почти всички езици.

#### WhoIs сървис

Външен сървис който се ползва за да се геолокира IP-аддрес за да могат да се извлекат по детайлни данни за потребител при посещението му на уебсайта.

### ANALYTICS

Слоят за анализ на поведението на потребителите служи за да се определи потреблението на уебсайта, интересът към дадени части от него успешността на съдържанието и параметри като : bounce rate, engagement ratio и множество други.

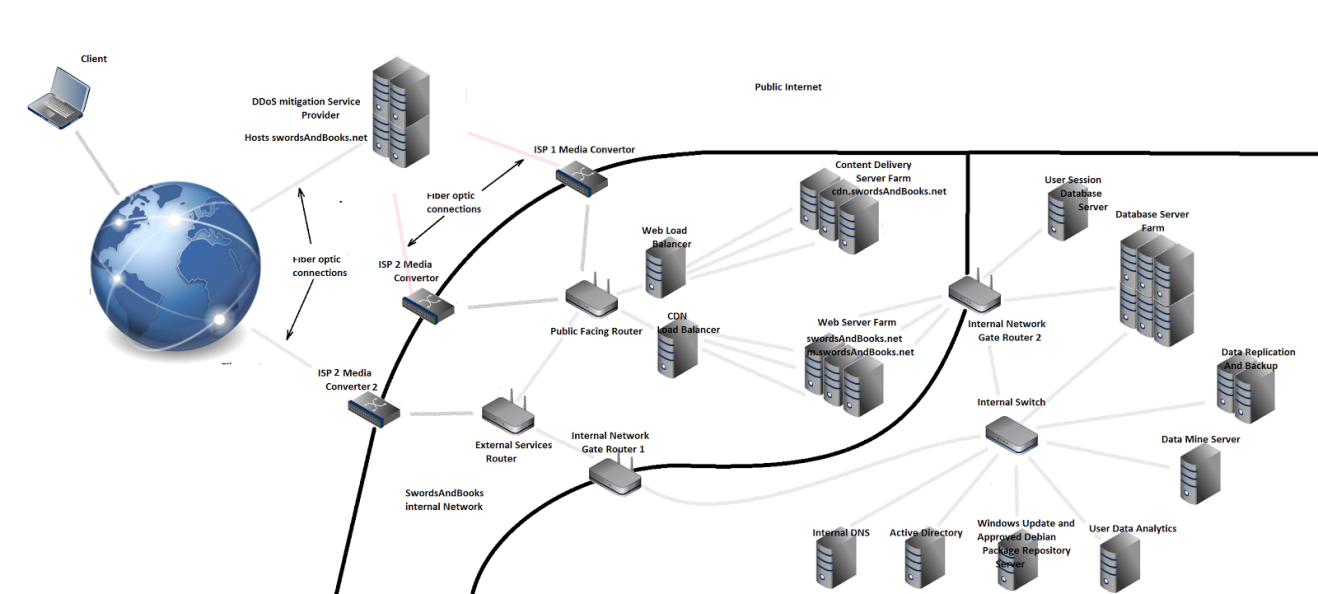
Анализирането на потребителски данни става на база на алгоритми за искуствен интелект , като **k-nearest neighbours**, който може да се ползва за класификация на даден потребител, оценяване успешността на енгенджмънта на съдържанието, следене на робот който краулва уебсайта за за пропуски и redirect loop-ove.

Също се ползва да измерва промените във горепосочените параметри при A-B тестване , при заменяне на даден уиджет с друг, като ефект на подобряването на рейтинга на дадена книга, изчислено със оценка на статистическата грешка на база на оценка на хипотези , създадени чрез data mining на данните.

### DATA MINING

Data Mining layer-a е отговорен за създаването на агенти - класификатори които да анализират данни и да класифицират потребители , посещения да определят дали дадени посещения са от роботи на база на usage patternи , да разделят потребителите на профилни карти, както и да предоставят AB-тестване на уиджети в епизодите на книгите игри, така че да се определи по-добро разпределение на съдържанието на книгата. Това става на базата на **Бейсови класификатори** , както и на **decision tree-s**.

## Физическа Архитектура

*[](http://faculty.washington.edu/jtenenbg/courses/360/f04/sessions/schneidermanGoldenRules.html)*

**Фиг. 4**. *Диаграма на физическата архитектрура*

Физическото разделение на сървисите във мрежата е трислойно, всеки от слоевете имащ специфична цел относно сигурност и функционалност. Най-външния слой съдържа само външен DDoS migitation service , който е конфигуриран със хостнейма на уебсайта. До него вътрешната мрежа на системата се свърва по две резервирани оптични линий от два различни инърнет сървиз провайдери за да се подсигури висока взаимна достъпност на двата слоя един между друг. Целият трафик между двата слоя е криптиран за да се подсигури, че не е възможна main-in-the-middle атака чрез вкарване на физическо устройство между двете локации.

На ръба на демилитаризираната зона седят три медия конвертора , два от които са за директен достъп до DDoS migitation service слоя, а един за достъп до външни сървизи. Медия конверторите от DDoS migitation слоя са свързани със рутер за публичен контент, докато единият за сървисите е свързан със рутер съдържащ правила спецялно за това.

Публичния рутер рутира трафикът към контента на уебсайта през да load balancer-а , един за уеб фермата на уебсайта, и един за контент деливъри сървера.

CDN load balancer-а разпределя трафика във content delivery server farm-a. А content delivery server-фермата служи за място на което се запазват статичните ресурси във уебсайта - като изображения, медия стриймове (аудио , видео и комбинирани), флаш и java аплети.

Web server-a също се load-balance-ва. Уеб сървер фермата обслужва рекуести към swordsAndBooks.net и m.swordsAndBooks.net. За да се запази съществуването на сесията на потребителя уеб сърверите имат достъп до най-вътрешната част от мрежата през Internal Network Gate Router 2, там съществува user session server(Memcache server), на който се пази сесията на потребителя , както и базата данни.

    Най-вътрешната част на мрежата се занимава с поддръжката на цялата мрежа, анализ на данни и бекъпване, в нея се намират: user session server-a, базата данни, бекъп сървера, data mining сървера,data analysis сървера ,DNS , Аctive Directory и ъпдейт сървери.

Бекъп сървера пази бекъпи от базата данни , data mine сървера и аналитикс сървера. Дейта майнинг сървера се грижи за създаването на нови агенти чрез обработка на данни от базата данни, докато аналитикс сървера се грижи за класификацията и обработката на данни чрез вече съществуващи такива. Той пази също статистическа информация за потреблението на сайта, и резултати от А-Б тестове.

Цялата вътрешна мрежа ползва Active directory сървер за налагането на политики в нея, DNS сървера е на отделна физическа машина от него. Ъпдейтите преди да бъдат приложени на сърверите трябва да бъдат прегледани и удобрени от администраторите, за това ъпдейт сървера служи като репозитори за пакети които са удобрени и за уиндоус ъпдейт сървер на удобрени ъпдейти.

## Архитектура на фронтенда

### Require JS

require.js е фунционалност, позволяваща зареждането на клиентски код, само когато е необходим.

Чрез този модул се зареждат ресурси на клиента асинхронно и само когато те са необходими. Този подход има няколко предимстрва. Първото такова е подобряването на скороста на клинетската страна от приложението, което води до много по-добро взаймодеиствие с потребителите и подобрява цялостното изживявяне при употреба на системата.

### Асинхронно зареждане на епизоди

Интрерфейса за зареждане на епизоди е основен в системата, защото се приема, че по-голямата част от времето потребителя ще прекарва там. За да се подобри визуализацията и потребителското преживяване на тази страница е имплементирано асинхронно зареждане на епизоди. Поради характеристиките на физическата архитрекура това ще повиши значително производителноста и времевите характеристики на системата. Същноста на асинхронното зареждане на епизоди се съдържа в няколко важни компонента. Първия компонент е, че веднъж заредени базисните компоненти на интерфейса за четене няма да се презареждат на всеки епизод, а ще се преизползват. Това, което ще се променя ще бъде съдържанието на епизода, който се чете и функционалностите , конкретни за този епизод, който трябва да бъдат заредени.

Когато един епизод се зарежда неговите компоненти са обработени от логикатана фронденд архитектурата и визуализирани за потребителя.Докато потребителя работи ( чете,слуша, провежда битка ) над епизода системата, изкуствения интелект се възползва от жанра на книгите-игри и факта, че от този епизод ще има ограничен брой следващи епизоди, пресмята кой от тях са по- вероятни да бъдат избрани ( базирано на **статистическо учене**  и анализ на поведението на потребителя до сега) и ги зарежда информацията, която обаче не би издала съдържанието на епизода в системата. Така когато потребителя направи изобра си, времето за чакане е само необходидмото на клинектската архитекутра да извлече самото съдържание на епизода (останалото вече е заредено в системата) и да го представи във предварително подготвените модели. Това създава впечатлението за потребителя за четене на истинска книга , защото времето за преминаване между епозидите е минимално и не се усеща интеракцията с световната мрежа. В случаи, че потребител избере епизод преди данните за него  да бъдат предварително извлечени от системата на извличането им се дава максимален проритет, и това не води до значително забавяне в зареждането на ресурси.

## Архитектура на процеса по разработка.

Това са архитектурите, който ще бъдат използвани повреме на разработката на приложението. Внимателния им избор и интегрирането им с технологиите на разработка позволяват значително подбрение на възмоностите за лесно модифициране и разширяване на проекта.

### npm

npm е мениджър на пакети над node js, която позволява лесно инсталиране на модули осигуряващи различни функционалности за приложението. Предмствата тук са осигурено от една страна от факта, че всеки модул поставен с npm се зарежда локално за системата и така не намалява представянето на другите части от системата, който не зависят от нея. От друга страна разможаването на инстанции на приложения използващи npm е изключително ленсо и точно за това този модул е един от най-предпочитаните в световен мащаб. Той(npm) автоматизирано записва имената на всички модули, който са поставени на системата. В последствие благорадение на този механизъм за да настройм проложението си на нова машина е необходимо единствено да се възползваме от командата npm install, която автоматизирано ще инсталира всички необходими модули  за работата на системата. Това е един от основните компоненти, който значително подобряват възможностите за лесно разширяване и подобряване на системата.

### Grunt

#### Темплейти за билдване и оценки на качеството на кода

Системата ще използва темплейтни езици, който ще позволяват преизползването на множество готови елементи чрез написването само на няколко реда код. Такива ще са паршали за визуализация и стуктурите от данни като Бейсови и синтактични мрежи както и някой специфични структури необходими при създаването на **невронните мрежи** на системата.

За оценки качестовото на кода ще спомагат така наречените хинтери като  js-link и  js-hint. Това са програми, който следят в реално време стуктурата на кода и подсказват на пишещите когато допускат грешки или лоши практики. Употребатана подобни системи повишава качеството на кода, което води до много по лесно подържане , размножаване и разширяване на системата.От друга страна по-добрият код води до по добра работна атмосфера,, което винаги е предпоставка за по-добри резултати.

#### Unit test-ове

Unit тестовете са реализирани посредством автоматизираната система на geddy за създаване на тестове.Тя генерира готови за попълване такива след създаването на всеки нов файл с автоматизирания генератор. Това е много полезно първо, защотото позволява създаването за голям брой тестове за минимално време, второ защото автоматичното генериране осигурява голям процент тестван код и трето защото се избягват сложните итегрирания  и пускане на тестове при инсталация на нова инстанция.

### Bower

Bower е мениджър на пакети над клиентския js, която позволява лесно инсталиране на модули осигуряващи различни функционалности за приложението. Предмствата тук са осигурено от една страна от факта, че всеки модул поставен с Bower се зарежда локално за системата и така не намалява представянето на другите части от системата, който не зависят от нея. От друга страна разможаването на инстанции на приложения използващи Bower е изключително ленсо и точно за това този модул е един от най-предпочитаните в световен мащаб. Той(Bower) автоматизирано записва имената на всички модули, който са поставени на системата. В последствие благорадение на този механизъм за да настройм проложението си на нова машина е необходимо единствено да се възползваме от командата bower install, която автоматизирано ще инсталира всички необходими модули  за работата на системата. Това е един от основните компоненти, който значително подобряват възможностите за лесно разширяване и подобряване на системата.

### Git и Github

Това са системите за контрол на версията , за централизация на репозиторита и за анализ на работата по проекти.

Посредством множеството фунционалности на системата Git , кода се разработва в отделни бранчове и след тестване и удобряване бива пренасян в основните хранилища. Тази методология осигурява реализиране на добри мениджмънт техники за управление на разработката, както и увеличаването на сигурността на приложението.

Чрез фунцкионалностите за следене на хронологина на промените и визуален анализ на развитието на прокета се постига значително подобрение в продуктивонстта при подръжка на системата.

Чрез услугите предлагани от Github за отдалечено създаване на хранилища , е много по-лесно да се подобри сигурността при евентуални проблеми с някоя инстанция защото има резервни копия в глобалната мрежа. Също така е много по лесно да поставим инстанция на всяка,  машина на която има нужда стига тя да има достъп до интернет.

Много полезни са също и приложенията за анализи над проекта, който позволяват да се наблюдават характеристики като свързаност на модулите на проекта , количествен анализ на кода и други. Посредством такива анализи е възможно да бъдат подобрени всички качествени атрибути на системата.

### Аgile and Kanban

В процеса на разработка на системата хората са организирани на екипи от до шест човека всеки работещ по модул, които спазват скръм методологията , с разликата че на края на всеки два спринта се прави synchronization meeting между скъм мастърите на екипите за да се види до къде е всеки от тях и дали има изисквания към другите. Друга разлика е че екипите работят със карти като във Kanban методологията, но те са свързани със спринтове.

### Физически характеристики на елементите на системата.

|  |  |
| --- | --- |
| УСТРОЙСТВО | КОЛИЧЕСТВО И ХАРАКТЕРИСТИКИ |
| **Рутиращо Устройство** | четири на брой - три от тях едногигабитови, един двугигабитов. |
| **Мрежови Суитч** | Гигабитов суитч |
| **Сърверна машина** | 8-ядрена с хипертрединг на 16 с 16 гигабайта рам, 20 на брой. |
| **Медия Конвертор** | Оптичен към лан медия конвертор, едногигабитов, 3 броя. |

**Таб. 3**. *Устроиства необходими за изграждане на физическата архитектура.*

## Мрежова Архитектура

### Външни интерфейси

Системата притежава един външен IP адрес , който служи за достъп до външни сървизи , VPN на служителите но не служи за изходна точка на уеб интерфейс.

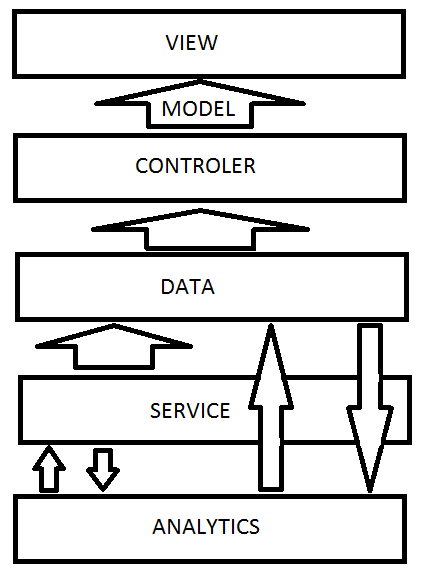
Закупени са две оптични влакна , едно от (ISP1) и едно от (ISP2) , директно достигащи до  DDoS Protection Service providera. Всяко едно влакно предоставя до 2.4 Гигабита неограничен трафик до DDoS Protection Service Provider-а. Целия трафик по тези две връзки е криптиран със 3\*DES алгоритъм , със генериран на база време мастър ключ , който се ползва при разпадане на връзката, и последователно генериране на последователни ключове иначе. Това подсигурява физическата сигурност на оптичните влакна, с това че man in the middle атака не може да има успех.

### Вътрешни интерфейси и ноудове, събнети.

Изключвайки най-външния anti-DDoS слой, мрежата е разделена на две зони на база на изискваната сигурност на достъпа и конфиденциалността на съдържаните данни на база на одитирана оценка при загуби. Първия слой е демилитаризирания слой, във него се съдържат ресурси външно видими за потребители от системата , въпреки че връзката е прокси-ната през още един слой. Това определя един събнет в който се съдържат уебсървера , контент деливъри сървера и техните съответни лоад балансери. Другият слой е вътрешната защитена мрежа , тя съдържа софтуер който налага политики за сигурност , ъпдейт сървер който менажира ъпдейтваемостта на другите машини от мрежата , бази данни, session storage-и, както и сървери свързани с бекъпване, анализ и обработка на данни. Всички коаксиални кабели във този слой трябва да са от тип “Screened shielded twisted pair”, освен това часта от сградата в която се съдържат вътрешните сървери е изолирана на базата на фарадеева клетка , така че да се блокират всички външни и вътрешни интерференции , както и потенциални изтичания на данни.

# Алтернативни архитектури (2 бр.)

## Алтернативна архитектура 1

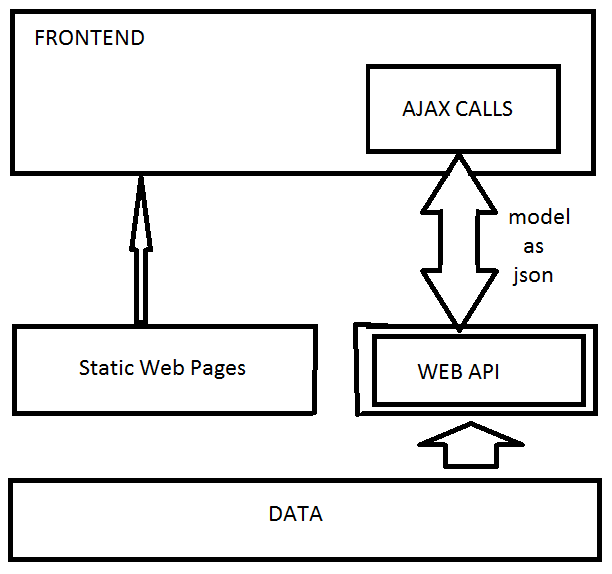
**

**Фиг. 5**. *Алтернативна архитектура*

Тази архитектура представлява стандартна архитектура за уеб приложение в която приложението работи на няколко слоя през който данните преминават.

На диаграмата се вижда че комуникацията е на няколко нива. Дата слоя комуникира директно с долните два, и също така с слоя на контролерите. Контролерите от своя страна подават инфромация на клиентската част информация. Тук клиенткста част е значително опростена..Оснонвата разлика със избраната от нас архитектура е точно в клиентската част. Опростяването в тази алтернативна архитектура на пръв поглед може да изглежда полезно, но се оказва, че за приложение от типа на нашето, където клиентската логика трябва да работи с голямо количество данни, проста клиенска архитектура се оказва твърде не продуктивна. Изобра на тази алтернативна архитектура би довел до **значително намаляване на бързодействието на системата** , което вероятно ще наруши техническите изисквания описани в този документ, подари което и тя не бе избрана за реализиране на този проект.

## Алтернативна архитектура 2

**

**Фиг. 6**. *Алтернативна архитектура*

Тази архитектруа се базира на стандартния подход на комуникация на клинета само чрез REST API. Тази аргитектура е често предпочитана при разработка на Уеб приложения, който работят с ресурси каквото е и описаното в този документ. Употребата на такава архитектура също е считана за добра практика поради яснотата на кода и фунцкионалността на прилжението. Избора на такава архитектура за този проект, обаче  се оказва неефективна защото би довела до невъзможност за създаване на множество от фунцкионалностите за улесняване на потребителите, което значително би намалило значително **използваемостта и комфортноста при употребата на приложението от потребителите.** Тъи като потребителското преживяване е от най-голяма важност в това приложение , това е и основната причина тази архитектура да не бъде предпочетена за разработка на това приложение.

# Заключение

С изпълнение на всички изисквания и характеристки, който се описани в този документ се смята че системата swordAndBooks има потенцияла да се превърне във водеща сила на пазара, и да има потенцяла да подържа достъпност по целия свят и възможност за работа с брой потребители сравним с този на най-големите социални мрежи.

# Ресурси

*[1]*[*https://www.owasp.org/index.php/Top\_10\_2013-Top\_10*](https://www.owasp.org/index.php/Top_10_2013-Top_10)

*[2]http://faculty.washington.edu/jtenenbg/courses/360/f04/sessions/schneidermanGoldenRules.html*

*[3]https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA*

*[4]http://en.wikipedia.org/wiki/Git\_%28software%29*

*[5] http://css-tricks.com/whats-great-bower/*

*[6]http://clintconklin.com/optimizing-multiple-javascript-files-with-grunt-and-requirejs/*

*[7] http://en.wikipedia.org/wiki/Npm\_%28software%29*

*[8] http://backbonetutorials.com/organizing-backbone-using-modules/*

*[9] http://requirejs.org/docs/api.html#usage*

*[10] http://en.wikipedia.org/wiki/Distributed\_System\_Security\_Architecture*

*[11] http://www.sitemaps.org/protocol.html*

*[12] http://en.wikipedia.org/wiki/Sitemaps*

*[13] http://www.robotstxt.org/meta.html*

*[14] http://en.wikipedia.org/wiki/Robots\_exclusion\_standard*

*[15]http://www.vervesearch.com/blog/how-to-implement-the-relauthor-tag-a-step-by-step-guide/*

*[16] http://www.hongkiat.com/blog/facebook-open-graph-wordpress/*

*[17]http://www.socialmediaexaminer.com/10-ways-to-add-facebook-functionality-to-your-website/*

*[18] http://www.ics.uci.edu/~andre/papers/T2.pdf*

*[19] http://ifs.host.cs.st-andrews.ac.uk/Books/SE7/Presentations/PDF/ch12.pdf*

*[20] http://en.wikipedia.org/wiki/Distributed\_computing*

*[21]* *http://en.wikipedia.org/wiki/Web\_testing*

*[22]http://thenextweb.com/apps/2013/11/28/guide-testing-web-app-steps-approach-testing-get-sessions/*

*[23]* *http://moz.com/learn/seo/meta-description*

*[24]http://googlewebmastercentral.blogspot.com/2009/09/google-does-not-use-keywords-meta-tag.html*

*[25]* *http://www.w3schools.com/tags/tag\_meta.asp*

*[26]* *http://en.wikipedia.org/wiki/Search\_engine\_optimization*

*[27]http://www.oracle.com/us/products/applications/atg/large-scale-ecommerce-platforms-1931115.pdf*

*[28]* *http://www.statsoft.com/Textbook/Data-Mining-Techniques*

*[29]* *http://css-tricks.com/thinking-async/*

*[30]* *http://deanhume.com/home/blogpost/loading-css-asynchronously/7104*

*[31]* *http://en.wikipedia.org/wiki/Kanban*

*[32]* *http://en.wikipedia.org/wiki/Agile\_software\_development*

*[33]http://docs.oracle.com/cd/E11882\_01/server.112/e40540/dist\_pro.htm#CNCPT006*

*[34]* *http://en.wikipedia.org/wiki/Network\_architecture*

*[35]* *http://www.ics.uci.edu/~fielding/pubs/dissertation/net\_app\_arch.htm*

*[36]* *http://programmers.stackexchange.com/questions/244598/modular-web-app-network-architecture*

*[37]* *http://web.uvic.ca/~erikj/seng422/resources/rest\_paper.pdf*

*[38]http://programmers.stackexchange.com/questions/67618/alternative-patterns-for-web-development-non-mvc*

*[38]* *http://www.optimizesmart.com/geek-guide-bounce-rate-optimization-google-analytics/*

*[39]* *http://en.wikipedia.org/wiki/Bounce\_rate*

*[40]* *http://www.sitepoint.com/3-ab-testing-tools-compared/*

*[41]* *http://en.wikipedia.org/wiki/A/B\_testing*

*[42]http://www.wikihow.com/Customer-Behavior-Analysis-and-Segmentation-Marketing-Ideas*

*[43]* *http://en.wikipedia.org/wiki/Consumer\_behaviour*

*[44]http://datascienceseries.com/assets/blog/GREENPLUM\_Analyzing\_customer\_behavior-web.pdf*

*[45]* *http://bg.wikipedia.org/wiki/%D0%9A%D0%BD%D0%B8%D0%B3%D0%B0-%D0%B8%D0%B3%D1%80%D0%B0*