Bikesurf Berlin

Проект по „Софтуерни архитектури“, 2017 г.

Факултет математика и информатика, Софийски университет

Изготвили:

80948, Звездалина Димитрова Димитрова, zvezdi.dim@gmail.com
Компютърни Науки, 4ти курс, 2ри поток, група 5
80950, Георги Валериев Павлов, georgipavlov94@gmail.com
Компютърни Науки, 4ти курс, 2ри поток, група 7
80967, Никола Ясенов Божинов, nbozhinov@uni-sofia.bg
Компютърни Науки, 4ти курс, 2ри поток, група 5
81008, Диана Антонова Генева, dageneva@gmail.com
Компютърни Науки, 4ти курс, 2ри поток, група 7
81050, Ангел Павлов Ангелов, hextwoa@gmail.com
Компютърни Науки, 4ти курс, 2ри поток, група 7

Ръководител: доц. Димитър Биров

2017 г.

Съдържание

1 Увод

За да вникнем в целите на проекта, първо трябва да се добие основна представа за неговите мащаби. Колелата са представени на масовата публика през 19 век и оттогава те стават важен фактор не само в спорта, но и в транспорта. Текущо колелата наброяват 1 милиард. Това означава, че грубо всеки седми човек притежава велосипед – приблизително същата сметка като при автомобилите.

Ако погледнем как стои Берлин на световната сцена, то ще заключим, че колоезденето е образува голяма част от трафика. По последни данни на 1000 жители се падат средно по 710 колела. Това е означава, че грубо 13% от транспорта е велосипеден. Може да се каже, че Берлин като метрополис заема челно място в класацията за велосипедни общества. Това не е всичко – гражданите на столицата имат достъп до 620 километрови велоалеи, а дори са и изградени специални улици за колоездачи (Fahrradstrassen).

Когато е изградена такава огромна мрежа в огромното 3.5 милионно общество на Берлин, тя трябва да бъде достъпна за всички. Затова е важно да се изгради начин за ефективното ѝ управление.

1.1 Тема на проекта

Един от начините да се изгради достъпна колоездачна мрежа е всеки да има достъп до велосипед, независимо дали притежава такъв в домакинството си. Това е практика, прилагана в повечето метрополиси, които решават проблема, като предлагат по някакъв начин колела на заем. В Лондон например системата **Santander Cycles** [SNTC] представлява общодостъпни рампи за колела, в които гражданите могат да наемат велосипед за определено време, като има много такива спирки из целия град.

Темата на нашият проект е свързана с начин за приемане и предлагане на колела за временна употреба в Берлин. Той се базира на директната връзка между наемателя и човекът, решил да предлага колелото си назаем.

1.2 Цели на проекта

Целта на нашият проект е да се създаде платформа, която улеснява комуникацията между наемащия и отдаващия колело. Първоначалната цел на проекта е Берлин, но на платформата може да се запише всеки град, изявил желание да участва. Във всеки град потребителят има достъп до информация за рампите на регистрираните наемодатели и съответно може да поиска да вземе под наем съответно колело. Тук нашата платформа трябва да даде възможност за комуникация между двамата, включваща уговаряне на място и час за взимане и оставяне на велосипеда. Останалите потребители могат да оценяват своите пътувания и да избират такива колела, които удовлетворяват техните изисквания. Сайтът, който изграждаме, трябва да предлага достъп до цялата тази информация.

Въпреки че подобни проекти вече съществуват, в тяхната реализация се крият множество проблеми – както софтуерни, така и архитектури. Затова важна цел на нашия проект е да изгради по-добра система от вече съществуващите.

Друго нещо към което се стремим е да се направи архитектурата по такъв начин, че регистрирането на изцяло нов град да става лесно и удобно, без намесата на системен администратор, което лиспва напълно в текущите реализацията.

Изключително важно е да се стремим към сигурност, защото тази платформа изисква работа с чувствителна информация. Освен паролите на потребителите, които очевидно трябва да са защитени, част от ифнормацията за колелата трябва да се пази допълнително. В противен случай би имало заплаха от кражби.

Цел на проектът е администраторите на съответните градове да имат контрол и той да е независим. Трябва да изградим архитектурата по такъв начин, че Recovery Time при падане на някой сървър да е минимален.

1.3 Резултати

Продуктът BikeSurf подпомага комуникацията между доброволците, заявили желание за участие в съответния град, регистриран на сайта на платформата. Регистрираните наемодатели имат право да слагат обяви за колелата си и да бъдат разглеждани и взимани от регистрираните наематели.

Заради децентрализираната система и използването на ORM, всеки нов желаещ град може да се регистрира лесно с позволението на администратора. Използването на модели прави заявките независими към sql базата, така че просто трябва да се инсталиране на съответната база и сървърен файл.

За да защитим потребители си, използваме Bcrypt за пазене на паролите. Подсигуряваме се, че те не са ботове, чрез ауторизация на лична карта/паспорт и по този начин силно намаляваме шанса от атака, тъй като потребителите ни са базирани на роли със съответни права. Информацията за конкретното местоположение на велосипеда се пази в тайна непостредствено до взимането му от наемателя.

Изборът на децантрализирата архитектура решава проблемите в проекта свързани с правата на администраторите и тяхната власт. Освен това, по този начин сървърите са сравнително независими, което позволява запазване на част от системата при падането на сървър, т.е. и много по-добро време за възстановяване.

2 Описание

2.1 Изисквания към софтуера

2.1.1 ИЗИСКВАНИЯ КЪМ СОФТУЕРА

Важна част от изискванията на проекта са свързани с роли. В подобен проект, различните потребители имат различни права за достъп до данните.

Разпознаваме четири вида потребители, които са съответно:

1. Гост

Потребителите от тип гост са с най-малко права. Те нямат регистрация на сайта и нямат право да наемат или отдават колела.

2. Потребител

Потребителите са преобладаващата роля на сайта. Те се разделят на две подкатегории – наематели и наемодатели. Потребителите разполагат с всички права на гостите, освен това имат регистрация на сайта, разполагат със собствен профил и могат да оставят коментари в публичните дискусии и форума.

1) Наемател

Наемателите са потребители, които могат да вземат колела под наем.

2) Наемодател

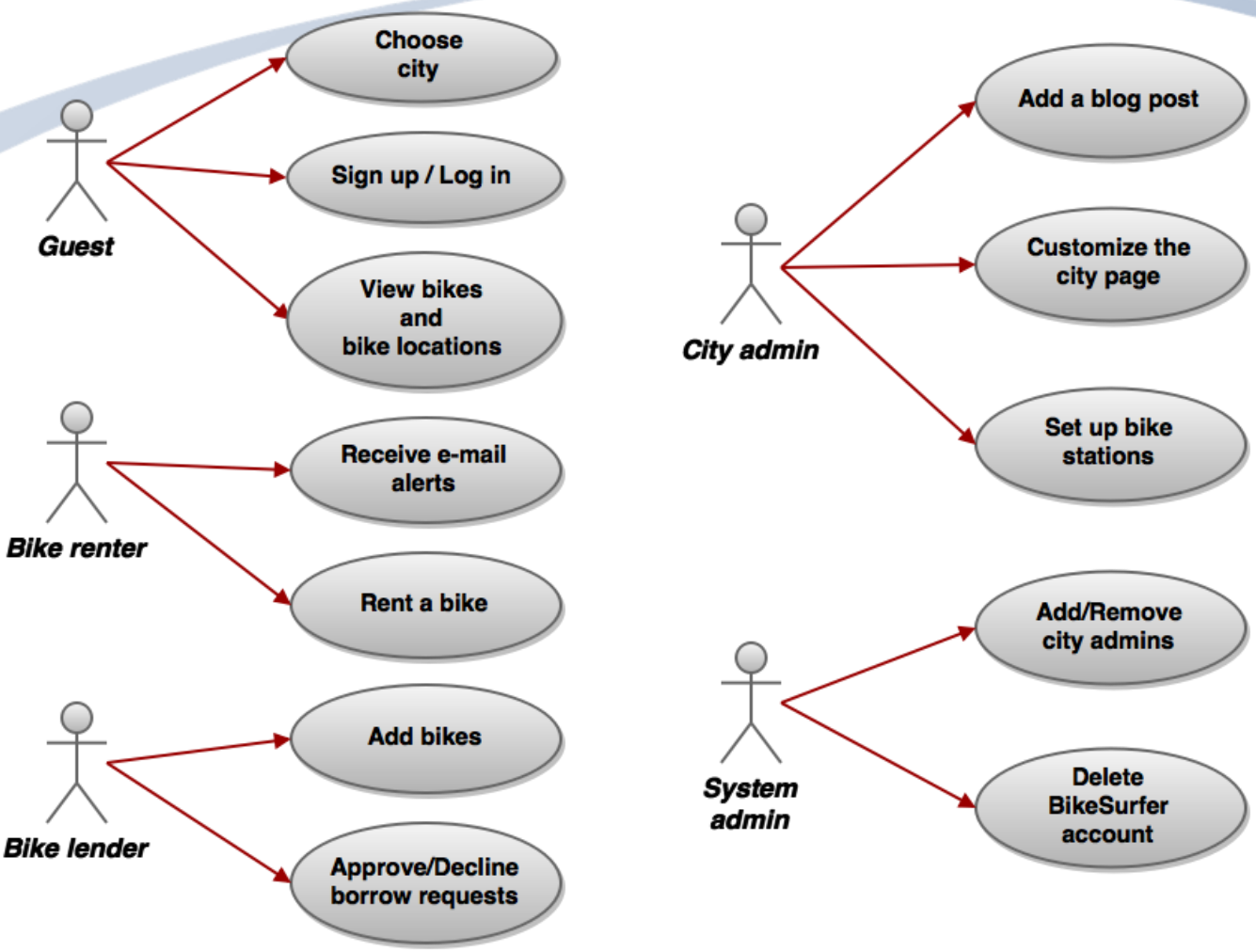
Наемодатели са потребители, които освен, че могат да вземат колела под наем, могат и да отдават колела. Те имат правото да потвърдят или отхвърлят молба за наемане на тяхно колело.

3. Администратор на град

Всеки град в BikeSurf системата си има собствен администратор, който се грижи за сайта на определения град, както и за стендовете със колела. Те могат да контролират молбите за заемане на колела.

4. Системен администратор

Системният администратор отоговаря за цялостната система и може да контролира всички неини модули.



Фиг. 1. *Use case диаграма за BikeSurf*

Ще разгледаме поотделно функционалните изисквания към всяка от тези групи. Всяко функционално изискване има уникален идентификационен код, например “FR305”. Буквите са съкращение за Functional Requirement (функционално изискване). Те са последвани от 3-цифрен код. Първата цифра представлява ролята, за която е функционалното изискване. Следващите две цифри съставят номера на изискването, като всяко следващо изискване е 5 номера след предното, за да има възможност в бъдеще да бъде добавено ново фукнционално изискване на правилното място, без да трябва да се изменят номерата на абсолютно всички изисквания. За приоритизация на функционалните изисквания ще използваме метода MoSCoW [MSCW], като ще ги степенуваме по категории – M: Must have, S: Should have, C: Could have, W: Would like but won't get.

2.1.1.1 Изисквания към гост

- FR105: Достъп до новинарския бюлетин

  Независимо дали потребителят е влязъл в системата, той трябва да вижда основните събития, свързани с проекта, които ще се намират на главната страница.

  MoSCoW приоритизация: M
- FR110: Избор на град

  Всеки потребител ще има възможността да избере сайта на града, който иска да разглежда.

  MoSCoW приоритизация: M

- FR115: Препратка към подобни сайтове

  Тъй като проектът е масивен, той кооперира с много сайтове. Целта е да има удобен начин потребителят да достъпва сходни услуги, като например couchsurfing – след като потребителят е избрал безплатно колело, той трябва да бъде пренасочен към сайт, който служи за намиране на място за преспиване.

  MoSCoW приоритизация: S
- FR120: Достъп до условията на сайта

  Проектът разчита на взаимодействие между хора, затова всеки участник трябва да може да бъде запознат с правата и задълженията си.
  MoSCoW приоритизация: M
- FR125: Достъп до дарителски фондове

  В bikesurfing наемодателят отдава велосипедът си безплатно, затова за поддържане на системата, както и за покриване на разходите за обслужването на самите колела се разчита на дарения. Те трябва да бъдат достъпни както по банков път, така и с виртуални пари – Bitcoin..
  MoSCoW приоритизация: M
- FR130: Регистрация и вписване

Гостът трябва да има достъп до форма за регистрация, ако още не е създал профил на сайта. За удобство, трябва да има свързване с профили от други сайтове (Facebook, Google, couchsurf). Ако потребителят вече има профил, той трябва бързо да може да го достъпи от менюто.
MoSCoW приоритизация: M

- FR135: Блог

  Сайтът поддържа блог, който трябва да може да бъде разглеждан от гости, дори те да не могат да публикуват. Публикациите трябва да бъдат разделени по градове.

  MoSCoW приоритизация: S

- FR140: Разглеждане на колела

  Макар гостите да нямат право да запазват колела или оставят коментари на страниците на потребителите, те трябва да имат достъп до снимките и информацията за тях, така че, след като се регистрират, да могат да запазят избраното от тях колело. Освен това, те трябва да виждат върху календара вече заетите дати, в които колелото е резервирано.
  MoSCoW приоритизация: M

- FR145: Търсене на колела

  Гостите трябва да имат достъп до формата за търсене и филтъра за колела, в който да попълват изискванията си – например, наличие на фарове, големина на колелото и други.
  MoSCoW приоритизация: M

- FR150: Достъп до често задавани въпроси

  Всеки посетител на сайта трябва да може да се информира как да борави със сайта и каква функционалност има той..
  MoSCoW приоритизация: S

2.1.1.2 Изисквания към потребителя

2.1.1.2.1 Наематели

- FR205: Наемане на колело

  Само вече регистрираните потребители трябва да имат достъп до бутон за запазване. Освен това тези потребители трябва да имат идентифицирана и потвърдена лична карта или паспорт.

  MoSCoW приоритизация: M

- FR210: Получаване на известия по пощата

  За всякакви промени по резервациите, регистрираните потребители трябва да имат право да получават информация на избран от тях имейл адрес.

  MoSCoW приоритизация: M

- FR215: Промяна на личния профил

  Наемателите трябва да могат да редактират личната си информация, която се показва на сайта, да сменят аватара или имейла за кореспонденция от страницата на личния си профил.

  MoSCoW приоритизация: M

- FR220: Достъп до чужди профили

  За да могат да изберат колелото си, наемателите трябва да могат да преглеждат и чужди профили, за да се запознаят по-подробно с информацията за наематодателите.

  MoSCoW приоритизация: C

- FR225: Достъп до форум

  Освен да преглеждат мненията, наемателите трябва да могат да оставят съобщения и да създават теми във форума.

  MoSCoW приоритизация: S

- FR230: Оставяне на съобщения

  Регистрираните потребители могат да оставят коментари на страницата на колелата, които са наемали с информация за колелото и пътуването, което са направили с него.

MoSCoW приоритизация: S

- FR235: Кандидатстване за наемодател

  Наемателите могат да кандидатстват за наемодатели, като кандидатурата им трябва да бъде одобрена от администратор на град.

  MoSCoW приоритизация: C

- FR240: Кандидатстване за преводач на страница

  Наемателите могат да кандидатстват за преводач на страница на град, като кандидатурата им трябва да бъде одобрена от администратор на сайта на града.

  MoSCoW приоритизация: W

2.1.1.2.2 Наемодатели

- FR245: Получаване на известия по пощата

  За всякакви промени по резервации на техни колела, наемодатели трябва да имат право да получават информация на избран от тях имейл адрес.

  MoSCoW приоритизация: M

- FR250: График

  Преглеждане на всички заявки за заемане на тяхно колело, както и филтри за одобрени и чакащи.

  MoSCoW приоритизация: C

- FR255: Одобряване или отхвърляне на заявки за наемане на тяхно колело

  Наемодателят трябва да има възможност да одобри или отхвърли всяка заявка за колело, което му принадлежи.

  MoSCoW приоритизация: M

- FR260: Редактиране на резервации

  Наемодателят може да променя някаква част от информацията, свързана с заявките към техните колела. Преди взимането на велосипеда, те трябва да дадат информация за точното му местоположение и паролата на катинара.

  MoSCoW приоритизация: S

- FR265: Добавяне на статии в блога

Той се намира на главната страница и е информативен за събития, свързани с проекта.

MoSCoW приоритизация: W

- FR270: Добавяне на колело

  Наемодателите могат да добавят нова обява за колело, в която са длъжни да добавят информация за велосипеда.

  MoSCoW приоритизация: M

2.1.1.3 Администратор на град

- FR305: Промяна на страница на град

  Избиране на цветова гама, фонт за блога и други модификации, свързани с външния вид на страницата за определен град.

  MoSCoW приоритизация: C

- FR310: Одобряване на кандидатура за наемодател

  Това е една от най-важните функции на администратора на град, защото това позволява добавянето на нови колела в системата. Администраторът трябва да подсигури коректността на кандидатурата и да я одобри или отхвърли.

  MoSCoW приоритизация: M

- FR315: Премахване на наемодатели

  Администраторът трябва да може да премахва наемодатели, ако прецени, че такава намеса е нужна.

  MoSCoW приоритизация: M

- FR320: Добавяне на стенд в системата

  След одобряване на наемодател, администраторът трябва да добави в системата стенд, на който ще се добавят колела от наемодателя.

  MoSCoW приоритизация: M

2.1.1.4 Системен администратор

- FR405: Добавяне и премахване на администратори на град

Сисадмините стоят най-отгоре в йерархията на проекта и трябва да могат да ауторизират лица, които да стоят на чело на град.

MoSCoW приоритизация: M

- FR410: Триене и промяна на потребители

  Администраторът има право да премахва потребители от сисмата, ако се налага.

  MoSCoW приоритизация: M

- FR415: Достъп до статистика

  Администраторът трябва да има достъп до информация за обща статистическа информация за сайта – например, брой профили, общ брой дни, в които са запзени колела и всякаква друга информация от подобен вид.

  MoSCoW приоритизация: S

2.1.2 НЕФУНКЦИОНАЛНИ ИЗИСКВАНИЯ

По-важна част от програмистка гледна точка са нефункционалните изисквания към проекта. Те са координирани с желанията на клиена и могат да бъдат разделени на няколко категории:

1. Използваемост
2. Надежност
3. Модулярност
4. Ефикастност и бързодействие
5. Поддръжка
6. Сигурност
7. Скалируемост

2.1.2.1 Използваемост

- NFR105: Добър изглед

Добре изглеждаш продукт, който се поддържа на последните версии на следните браузери: Google Chrome, Opera, Mozilla Firefox, Microsoft Edge, Tor Browser.

- NFR110: Удобен за поддръжка
- NFR115: Визуализация на различни екрани
  Трябва да изглежда добре на 4', 5', 7-9', 13-15' и по-големи от 17' дисплеи.
- NFR120: Лесно използваемо
  4/5 потребителя трябва да умеят да запазват колело в рамките на 5 минути, след като са гледали видеото за "добре дошъл".
- NFR125: Изискване за софтуер
  Не е нужно инсталирането на допълнителен софтуер от страна на потребителя.

2.1.2.2 Надежност:

- NFR205: Време за възстановяване
  Максималното време, в което системата се възстановява трябва да е по-малко от 10 минути.
- NFR210: Стабилност на системата
  Системата не трябва да бъде офлайн повече от три пъти годишно.

2.1.2.3 Модулярност:

- NFR305: Самостоятелни сайтове на градове
  Един град не трябва да зависи от другите градове. Ако сайтът на един град е недостижим, това не трябва да влияе на другите.
- NFR310: Инсталирането на нова версия не трябва да променя личните настройки и съдържанието на базата данни
- NFR315: Добавянето на нов град не трябва да афектира другите

2.1.2.4 Ефикасност и бързодействие

- NFR405: Времето за отговор на заявка трябва да е не повече от 200мс
- NFR410: Потребител трябва да може да се регистрира с 5 клика или по-малко

- NFR415: Потребител трябва да може да запази колело с 10 клика или по-малко

2.1.2.5 Поддръжка

- NFR505: Кодът трябва да спазва всички стилови изисквания
- NFR510: Системата трябва да включва unit тестове, които покриват 99% от всеки branch
- NFR515: Кодът трябва да спазва KISS (keep it simple, stupid)
- NFR520: Лесен за поддръжка – да не отнема повече от 10 човекочаса за запознаване с проекта и промяна на елемент в него

2.1.2.6 Сигурност

- NFR605: Да се използва само криптирана комуникация между клиенти и сървър
- NFR610: Паролите и други чувствителни данни да се пазят по разумен начин

2.1.2.7 Скалируемост

- NFR705: Системата да бъде стабилна с 5.000 заявки в секунда
- NFR700: Системата да бъде стабилна със 100.000 потребители
- NFR715: Системата да бъде стабилна с 500.000 оферти за колела

2.1.3 ВЗАИМОДЕЙСТВИЕ НА НЕФУНКЦИОНАЛНИТЕ ИЗИСКВАНИЯ

- Използваемост и поддръжка

  Когато възниква конфликт между изискването да върви на всички браузери и изискването да е лесен за поддръжка, слагаме фокус върху поддръжката. Затова ограничаваме браузърите в изискването до най-новите версии на най-често използваните от фокус групата браузери.

- Използваемост и бързодействие

  От една страна трябва да има интуитивен дизайн, а от друга времето за отговор да е под 200мс. Тук слагаме фокус върху бързодействието, като ограничаваме тежките визуални трансформации и анимации и залагаме на изчистен интерфейс.

- Ефективност и скалируемост

Когато сблъскваме изискванията за отговор под 200мс и изискването да поддържаме 5.000 заявки в секунда, ще наблегнем на скалируемостта, без да превишаваме лимит от 400мс за отговор. Това изисква кодът да бъде вертикално скалируем, макар това да влияе на поддръжката.

- Модулярност и поддръжка

  Тук наблюдаваме положително влияние на изискването за модулярност върху лесната поддръжка, защото модулярността изисква добре разделен и подреден код и налага някакви стандарти в кода.

2.2 Анализ и дизайн

Ще разгледаме избора на дизайн, направен въз основа на анализ от гледна точна на съществуващия вече дизайн и от наша.

2.2.1 РЕАЛЕН САЙТ(BIKESURF.ORG) [BKSRF]

Сайтът е създаден от Фабиан Бурко (Fabian Barkhau) през 2014 година. Подходът е да се наблегне на свободния фреймуърк Django [DJNGO] и неговата характерна архитектура. Преди да подходим към анализа, трябва да разгледаме как са покрити изискванията дадени от клиента. Ще разгледаме както функционалните, така и нефункционалните изисквания и ще видим предимствата и недостатъците на текущия подход. Важно е да се провери до каква степен тези изисквания са отразени.

2.2.1.1 Преглед на функционалните изисквания

В сайта, който е реализиран, съществуват съответните категории потребители, които имат и съответните им права. Ще разгледаме всеки отделен тип поотделно и доколко техните правила за изпълнени.

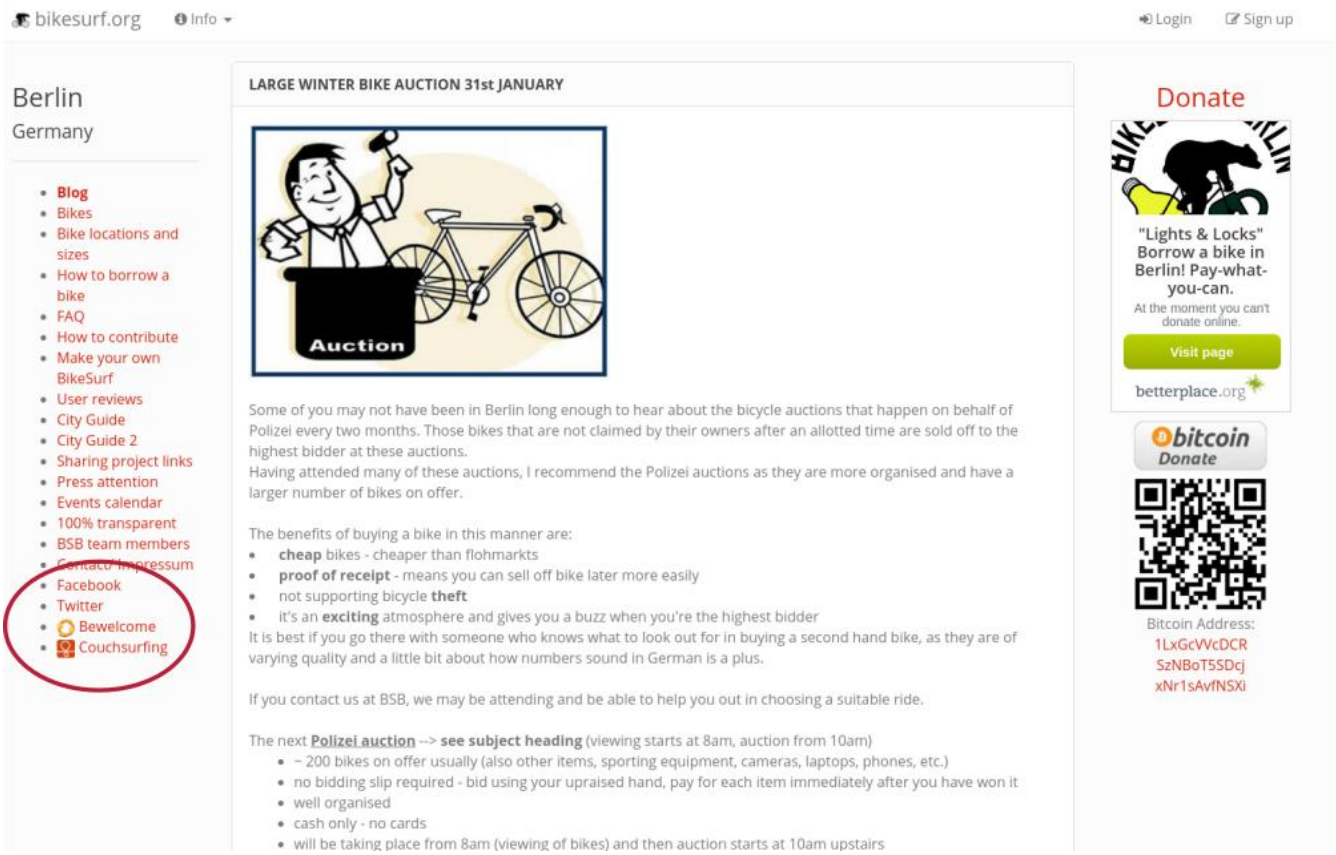
- Функционални изиксвания към гост

  Гостът има достъп до новинарския бюлетин.

  Гостът може да избира град, но, поради липса на модуларност, сайтовете на всички градове се намират на същия сървър. Това означава, че ако падне

сървърът, няма да има достъп до нито един от градовете, а това е лошо преценен бизнес риск.

Препратката към подобни сайтове работи, но е на неудобно за клиента място, така че условието да е виден е нарушено.



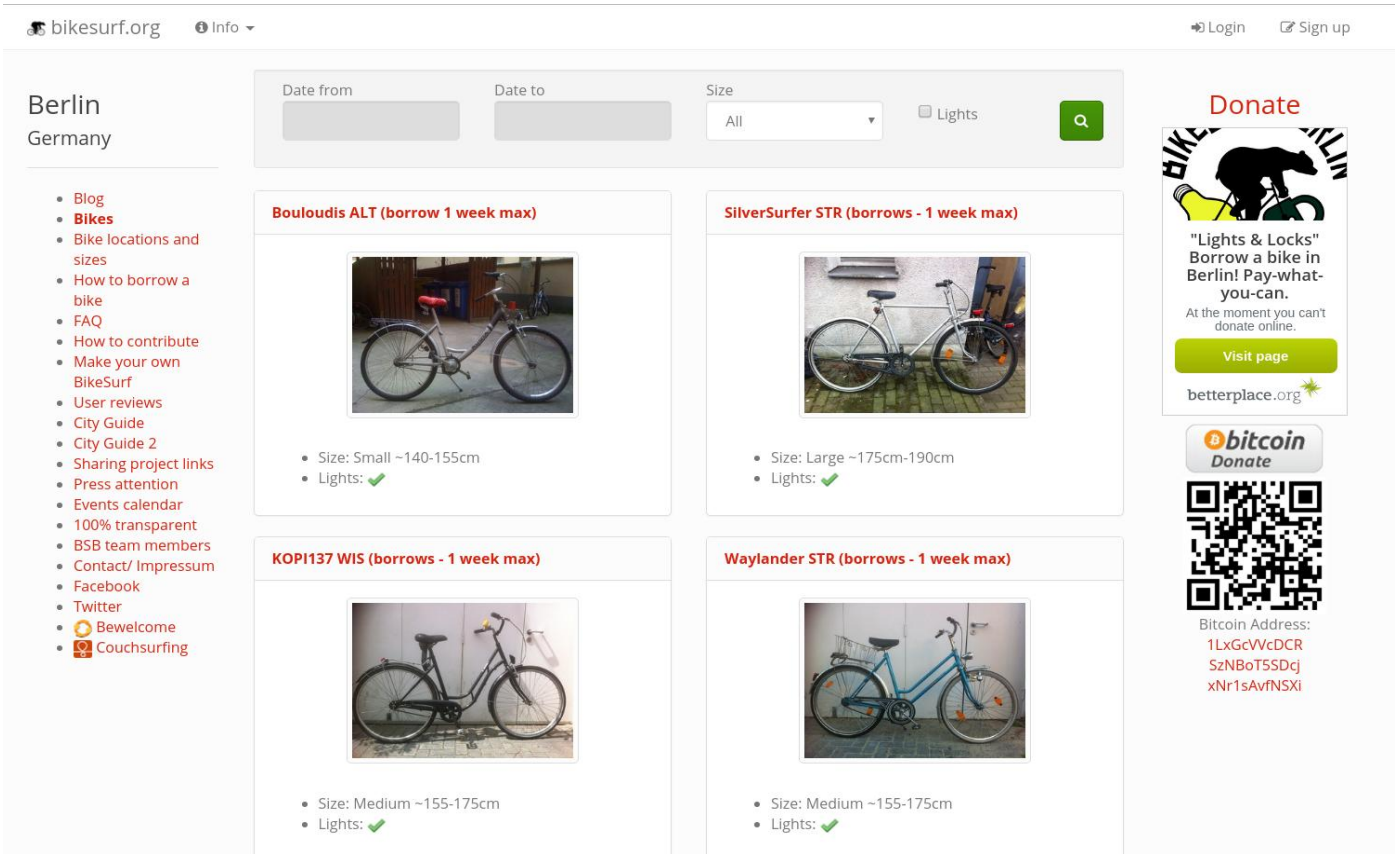
Фиг. 2. *Линкове в bikesurf.org*

Достъпът до условията на сайта е осигурен.

Достъпът до дарителските фондове се намира вдясно на сайта, но част от линковете са неактивни, което нарушава изискването.

Регистрацията на потребители е активна, но не е интегрирана с други подобни сайтове, което нарушава изискването.

Нерегистрираният поребител може да разглежда колелата, но не и да ги филтрира едновременно с това, защото в таблицата за колелата не се пази цялата им информация, което влошава потребителското изживяване и нарушава единството на



информацията.

Фиг. 3. *Търсене на колела в bikesurf.org*

Търсачката за колела не е динамична и потребителят не може да въвежда критериите си. Информацията за колелата се пази в таблици, които са картинки, което затруднява изключително много госта – от него се изисква ръчно да търси в картинката. По този начин няма нужда от използване на база данни – данните се пазят под формата на картинки, включително и картата с местоположението на колелата.

Bike name	Height		Lights? (b = battery; d= dynamo)		Backpedal braking system?	Quick release saddle?	Number of functional gears
	Frame (cm)	Crossbar (cm) - minimum inside leg size	Front	Back			
Small GoldenTexo	46	50	YES (b)	YES (b)	YES	NO	1
Small ClubMate	50	45	YES (b)	YES (b)	YES	NO	2
Small AdventureTime	53	53	YES (b)	YES (b)	YES	NO	3
Small Bertha	46	75	YES (b)	YES (b)	NO	YES	7
Small Bouloudis	45	45	YES (b)	YES (b)	YES	YES	3
Small BlueThunder	41	36	YES (b)	YES (b)	YES	YES	1
Small AnnaMaria	49	61	YES (d)	YES (d)	NO	YES	18
Small Velosiped	48	61	YES (d)	YES (d)	NO	YES	18
Small RedTube	52	58	YES (d)	YES (d)	YES	YES	1
Small mRrOBOT	45	65	NO	NO	YES	YES	7
Small Waylander	52	58	YES (d)	YES (d)	YES	NO	1
Medium Dominatrix	53	57	YES (d)	YES (d)	YES	YES	1
Medium TourKotti	46	75	YES (b)	YES (b)	NO	NO	18
Medium Amelia	50	50	YES (d)	YES (d)	YES	NO	3
Medium Skippy	51	75	YES (d)	YES (d)	NO	NO	18
Medium Yellow Submarine	54	45	YES (b)	YES (b)	YES	NO	3
Medium Alaska	53	78	YES (b)	YES (b)	YES	NO	1
Medium B-Live	54	55	YES (d)	YES (b)	YES	NO	1
Medium Maxine	52	50	NO	YES (b)	YES	YES	3
Medium Francois	53	56	YES (b)	YES (b)	YES	NO	1
Medium POWERBOTTOM	50	77	YES (b)	YES (b)	NO	YES	24
Medium Alva	53	58	YES (b)	YES (b)	YES	NO	1 (functional)
Medium AfroSamurai	38	55	YES (d)	YES (d)	YES	NO	1
Medium BlackDynamite	50	70	YES (d)	YES (d)	NO	YES	24
Medium Wazig	52	78	YES (b)	YES (b)	YES	NO	1
Medium Erinium	49	50	YES (d)	YES (d)	YES	NO	3
Medium StreetFighter	52	80	YES (d)	YES (d)	YES	NO	4
Medium AzuraCoast	53		YES (d)	YES (d)	NO	NO	18
Medium Gudereit	57	82	YES (b)	YES (b)	NO	YES	24
Medium Sherlock	50	82	YES (b)	YES (b)	YES	NO	3
Large AmyKTM	55	83	YES (b)	YES (b)	NO	NO	14
Large Odysseus	55	60	YES (d)	YES (b)	YES	NO	1
Large GorillaBiscuits	55	82	YES (b)	YES (b)	NO	YES	18
Large Kermit	57	82	YES (b)	YES (b)	YES	NO	1
Large KreuzbergRunner 22	55	83	YES (d)	YES (d)	YES	NO	3
Large Primitivo	59	45	YES (b)	YES (b)	YES	NO	3
Large SoylentGreen	60	85	YES (d)	YES (b)	YES	NO	1
Large Baikal	52	80	YES (d)	YES (d)	NO	NO	7
Large SilverSurfer	57	84	YES (d)	YES (d)	YES	NO	1

Фиг. 4. *Филтър за колела на bikesurf.org*

- Функционални изисквания към потребителя

  Резервацията и комуникацията между потребителите се случва изцяло през имейл адрес, което нарушава функционалното изискване за комуникация

през сайта. Това отново означава липса на база данни, защото съобщенията не се съхраняват в базата на сайта.

Промяната на личния профил изисква намеса на администратор. Освен това снимките на личните карти и паспортите се пазят в публично достъпни URL-та (последователни) , което дори не подлежи на обсъждане като практика.

Останалите изисквания от клиента са удовлетворени безпроблемно.

- Админи и сисадмини

  Тук реализацията съвпада с функционалните изисквания.

2.2.1.2 Нефункционални изисквания

- Използваемост

  Сайтът е тежък и трудно разбираем – стилистично зле оформен, заради твърде много информация, която объркава потребителя. Достъпен е от всички браузери. За релаизацията на сайта се ползва WordPress.

  Сайтът не е удобен за поддръжка, заради избора на технология и фактът, че много малка част на кода е извън готовите страници на Django.

  Сайтът не се поддържа на всички дисплеи, защото дизайнът му не е скалируем, заради избора на картинки като източник на ифнромация.

  Сайтът не изисква използването на допълнителен софтуер.

- Надеждност

  Сайтът се базира върху един сървър, което означава, че нефункционалното изискване за максимално време за възстановяване е нарушено, тъй като е много по-трудно да се локализира проблемът. Освен това, тъй като не е добре разделен, няма достатъчно разделение и на администраторите, които трябва да оправят проблема. Въпреки това, след като бутнахме сайта (с 100 паралелни заявки), той се възстанови за под 10минути, така че тази част от надеждността е спазена.

- Модулярност

  Сайтът не е добре модулиран – сайтовете на градовете не са еднакви, защото няма задължителна информация.

Фактът, че са на един сървър значи, че сайтът на всеки един град пряко се влияе от останалите.

- Ефикасност и бързодействие

  Поради изборът на технология и архитектура, времето за отговор на сайта стига до 2 секунди. Кодът не е отворен и не се знае дали конвенциите за кода са спазени.

- Поддръжка

  Кодът не е публично достъпен и собственикът отказва кооперация. Точно затова се преминава към създаването на нов проект.

- Сигурност

  Това е може би най-големият проблем на сайта. Той не е сигурен по нито един критерий за сигурност. В момента екипът успя да изтегли над 4000 снимки на лични карти и паспорти (и един акт за раждане).

  Сайтът не използва https връзка, което означава, че праща логин формата в прав текст. Това нарушава всякакви конвенции за сигурност, тъй като паролите и друга чувствителна информация не са криптирани.

- Скалируемост

  С последователни заявки, при 5000 заявки, 50% достъпват сайта с време за отговор 2 секунди, което нарушава желание 200мс, а средното време за отговор е 3 секунди.

  При 100 заявки в секунда едновременно, сървърът престава да работи. Времето за достъп надхвърля 15 секунди.

  Общият анализ върху текущата реализация на сайта е, че тя не спазва нито функционалните, нито нефункционалните изисквания към системата. Това се дължи на лош избор за технологии и лоша архитектура, като като най-сериозният недостатък на сайта е липсата на модулярност.

2.2.2 НАШИЯТ ПОДХОД

2.2.2.1 Функционални изисквания

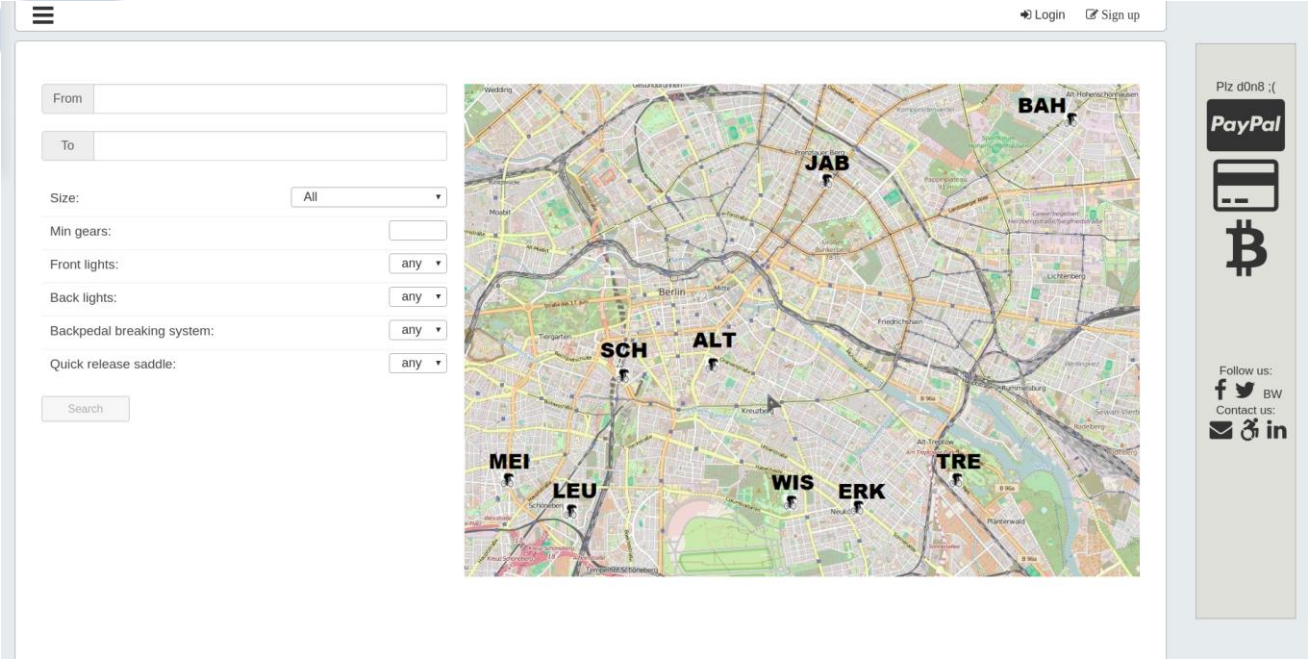
Анализът върху функционалните изисквания наложи нуждата от разделяне на потребителите, като тук нашето решение съвпада с това на оригиналния сайт. Съществуващите роли ще имат различни права на достъп до данните и операциите.

- Изисквания към госта

  Изискванията към госта са спазени като проблемите са решение с наличието на gateway сървър, в който се пазят само регистрациите и от него се пренасочва към различните градове. Фактът, че потребителят трябва да има достъп до филтър и търсачка, навежда към мисълта за използване на база данни. Това подобI guessed that maybe a Russian guy heard the song but couldn't remember/understand the words and made up his own versionрява търсенето, свеждайки го до динамично.
- Изисквания към потребителя

  Изискванията отново са удовлетворени. Спазването на функционалните изисквания за разменяне на съобщения при резервация, отново изисква работа с база, за да могат да се пазят съобщения без външната намеса на имейл сървър.

  Търсачката за колела е динамична и обвързана с прегледа на колела.



Фиг. 5. Филтър на нашата реализация на bikesurf

- Изисквания към администратори

  Заради избора на администратори на градове и главни администратори, логично е да има разделение на сървъри. Това означава всеки град да има отделен сървър, спазващ обща структура, който да има отделен администратор.

2.2.2.2 Нефункционални изисквания

По-важните от програмистка гледна точка изисквания са нефункционалните. Те подкрепят избора за модулярна архитектура и добавят изисквания към избора на технология.

- Използваемост

  За да работи на всякакви дисплеи, изборът е собствен CSS, което позволява по-голяма гъвкавост при разработката. Също така позлването на Flex и Media queries при дизайна значи, че сайтът ще променя динамично размера си. Така сайтът ще изглежда добре на всякакъв вид дисплеи.

  Не се ползват плъгини, флаш плейър или джава, което значи, че няма нужда от използването на допълнителен софтуер.

- Надеждност

  Има тестове, които покриват над 90 процента от кода. Тестовете на системата означават, че има по-малка възможнст за падане на сърврвъра поради програмистка грешка.

  Изборът на технология е Ruby и използването на rack [RACK]. Това означава, че заявките се обработват в отделни нишки и по този начин зависването на една, няма да прекъсне останалите.

- Модулярност

  Заради изискванията от клиента, най-удобно ще е системата да бъде разбита на отделни сървъри – по един за всеки град и един gateway сървър. По този начин добавянето и махането на градове няма да зависи от другите сайтове и да ги афектира по никакъв начин.

- Ефективност и бързодействие

  Дизайнът на сайта позволява бързия достъп до важна информация, заради наличието на странично и горно меню.

  Времето за отговор на заявка е между 100 и 200 милисекунди, заради изборът на Ruby, Rack и Sinatra [SNTRA].

- Поддръжка

  За лесна поддръжка трябва да се спазват стандартите в Ruby, а иммено – не повече от 80 символа на ред, не повече от 20 реда на метод и 80 реда в клас. Спазването се подсигурява с gem-а rubocop.

  Системата се тества с unit тестове, които се извикват при всяка в кода.

  Системата е добре планираната, заради своята модулярност и не позволява повтаряне на излишна информация. Също така базата данни е нормализирана и в нея не се пазят картинки, например.

- Сигурност

  Паролите се хешират и не се позволява достъп до чувствителни данни (като местоположение на велосипед) за потребители без съответни права.

Приложението се достъпва през rack интерфейс и пред него може да стои криптиращ сървър, който праща вече криптирани данни до нашето приложение.

- Скалируемост

  Отново поради изборът на Ruby и rack, системата се държи добре при много заявки. Освен това, тъй като данните са разпределени между сървърите на различните градове, няма пренатрупване на данни, което позволява поддържането на много потребители. Фактът, че трябва да се пазят картинки за всяко от 500000 колела, навежда на мисълта, че те не трябва директно да се съхраняват в базата данни, а да се пази път до физическите картинки, които сървърът пред приложението ни директно достъпва.

2.3 Архитектура

Софтуерните архитектури са важни за успешното разработване на софтуерни системи. Софтуерните системи се изграждат за да удовлетворят бизнес целите на организациите. Архитектурата е мост между тези (често абстрактни) бизнес цели и крайната (конкретна) система.

Има различни начини, по които функционалните и нефункционалните изисквания могат да бъдат удовлетворени. Те налагат някакви софтуерни ограничения върху избора на технологии и изборът на архитектура. Тъй като проектът е под формата на сайт, няма как да се избяга от архитектура тип клиент – сървър.

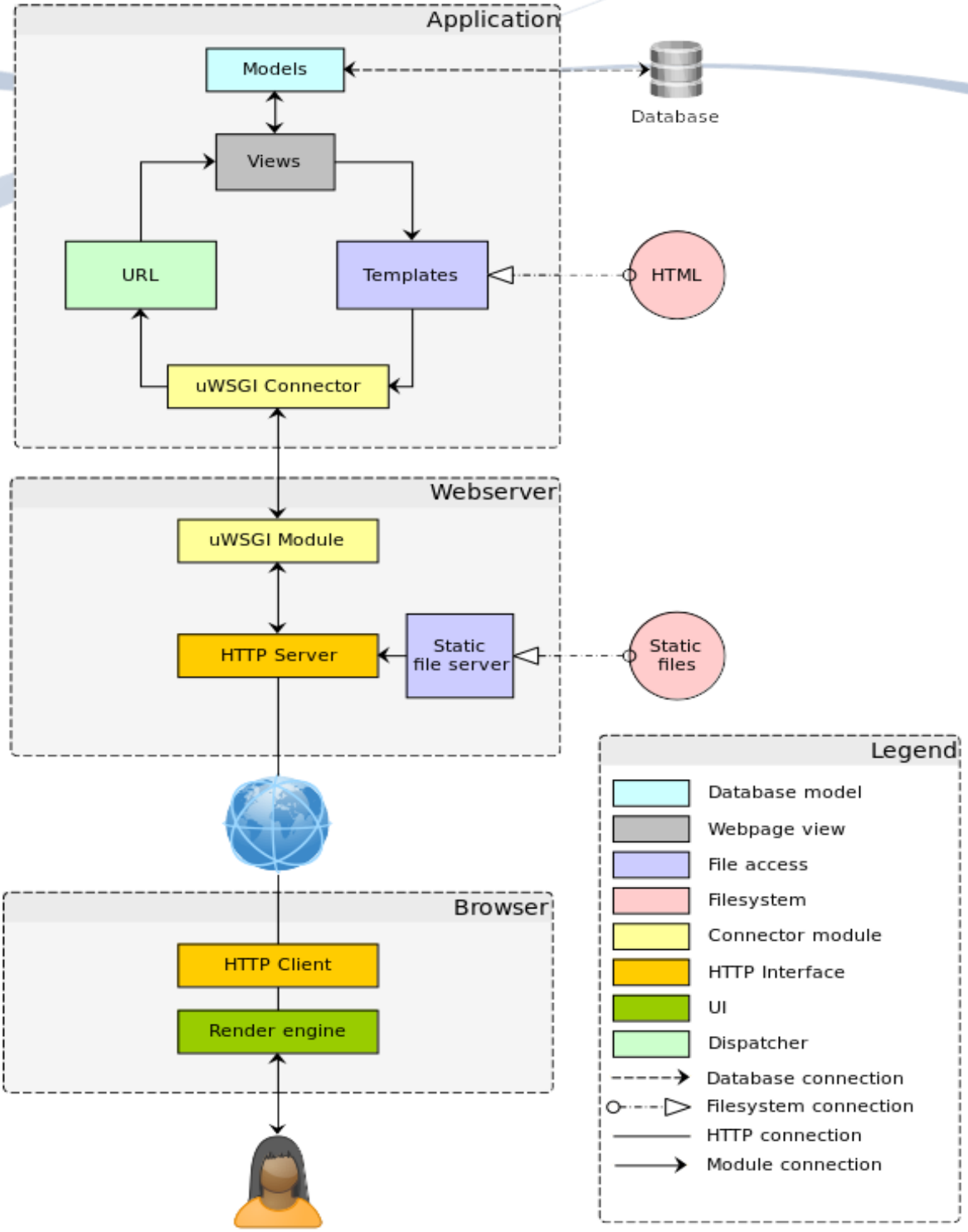
За да разгледаме архитектурата в проекта, първо ще видим тази, която предлага текущия сайт bikesurf.org, а после ще наблегнем върху алтернативно предложение за архитектура, реализирано от нашия екип. Всяка от двете архитектури има своите предимства и недостатъци, които са свързани със нейните спецификации.

2.3.1 ТЕКУЩА АРХИТЕКТУРА НА BIKESURF.ORG

2.3.1.1 Технология

Текущата реализация на проекта е реализирана с framework-а Django. Това означава, че архитектурата е изградена с по стандарт на Django, тоест е тясно свързана с избора на технология.

2.3.1.2 Общ преглед на архитектурата



Фиг. 6. *C&C диаграма на bikesurf.org*

На фигура 6 (C&C диаграма) се виждат обособени три части. Въпреки това, те не могат да се разделят по такъв начин, че всяка да е самостоятелна, понеже браузърът визуализира статични HTML страници. От диаграмата става ясно, че връзката между базата данни и приложението има ролята на комуникационен конектор и неговият тип е за достъп до данни (Data access connector). Тази връзка е

непълноценна, понеже в базата се съхраняват данни само за потребителите. Данните за останалите компоненти, като колела и стендове, са статични (под формата на картинки).

2.3.1.3 Описание на текущата архитектура

Архитектурата на проекта е клиент-сървърна, което е неизбежно в подобно клиентско приложение. Тя е двуслойна, защото няма добре изграден трети слой.

- Първи слой

  Първият слой се образува от базата данни. Той се нарича още и слой на данните и съдържа в себе си както абстракцията на базата, така и физическата информация. Тази част е изключително важна и почти задължителна за подобен тип система. В случая, архитектурата е силно дата-центрирано, което също няма как да се избегне, поради темата и целите на bikesurf. Въпреки това, проектантите са решили да не наблягат върху употребата на база данни.

- Втори слой

  Вторият слой се състои от няколко части, които обаче не могат да се обособят в отделен слой. Точно поради тази причина, не можем да наречем архитектурата трислойна, макар че тя е много близо до нея. В случая браузерът само показва вече готовите HTML-и, а не включва някаква допълнителна обработка – като javascript, например.

  1. Модели

     Първата част на втория слой са моделите, които служат за комуникация с базата данни. В случая, извън служебните за Django модели, има само един модел – Колело. Той служи са описанието на този обект и неговата репрезентация в паметта. В модела са описани всички полета, както и техният тип.

  2. View(Изгледи)

     Следващият компонент, изгледи, служи за управлението на това какво вижда потребителят на екрана си. Те се използват, за да обработват

информацията, получена от моделите. Това става по такъв начин, че тя директно може да се интегрира със следващия компонент – HTML темплейтите.

3. Темплейти

   HTML темплейтите са HTML файлове. Те самите са шаблони – направени са така, че на определени места да бъде вмъквана специфична информация. Тази информация се взима от изгледите и представлява преработена информация от базата данни.

4. Webserver

   Уебсървърът служи за предаване на информация между приложението и крайния клиент през интернет. Web Server Gateway Interface е универсална спецификация за общуване между уеб сървъри и уеб приложения чрез езика за програмиране Python. За да изълни заявка, страната на сървъра изпълнява прилжоението и осигурява информация за средата и callback функции за страната на приложението. От другата страна, приложението изпълнява заявката и връща отговор чрез осигурената callback функция. В случая между двете страни не стои нищо и няма осигурено API.

   Уебсървърът може да достъпва сървър със статични данни, на който са качени изображенията на колелата. Това се прави с цел да не се пази огромно количество информация в базата данни, защото това би довело до бавен трансфер на данни. Уебсървърът взима тази информация от статичния сървър и го попълва в своите страници при нужда.

5. Браузър

   Следващият компонент на втория слой е бразърът, който клиентът използва, за да достъпи приложението. Тук той има единствената роля да изобразява (render) HTML страниците, създадени в шаблонните HTML-и. На браузерът не се случва никаква допълнителна обработка на данните чрез скриптов език.

6. Диспечер

Последният компонент от този слой на двуслойната архитектура е URL диспатчерът. Неговата роля е правилно да съпоставя изглед с URL адрес. Например ако на браузера настъпи някакво събитие – например клик с мишката върху бутон - се генерира ново URL и зад него стои опрделен шаблон.

2.3.1.4 Предимства

Главното предимство на този вид модел е, че по никакъв начин не натоварва браузъра, тъй като той има единствена роля да показва вече изгенерираните HTML-и. Освен това, тъй като не се използват всичките възможности на браузера , това означава, че приложението работи и на повече видове такива.

Този модел донякъде има и плюса, че за всяка страница се прави само по една заявка, която зарежда всички данни. Това може да е както положително, така и отрицателно качество.

Базата е проста, тъй като не съдържа повече от един невграден модел. Това означава, че и заявките до нея ще се изпълняват много по-бързо, отколкото ако цялото пазене на информацията разчиташе на база данни.

2.3.1.5 Недостатъци

Най-големият проблем е липсат ана разделение, а и съответно фактът, че имаме двуслойна, а не трислойна архитектура. Липсата на модулярност води до много по трудна поддръжка, защото самият код е по-труден за разчитане. Друго следствие от текущата архитектура е много по-трудното интегриране на нов град, защото няма как да се даде по-малка част от целия проект, която да бъде deploy-ната при новия клиент. Всъщност, всичко за новия град трябва да се намира на същия сървър, а това нарушава някои от нефункционалните изисквания.

Тъй като се използват готови HTML-и, е много трудно да се създаде интерактивен дизайн на сайта. Тъй като от всяко цъкане на потребителя се генерира нов HTML, то интерактивните елементи от интерфейса биха изисквали ново

препращане към страница. Това е както бавно, така и непрактично, защото се предава голямо количество излишна информация.

Спрямо избора за съхраняване на информация, също могат да се намерят някои минуси. Такъв е например фактът, че тъй като не се ползва база, то за да се визуализира информация на сайта, тя трябва да бъде въведена по неудобен за потребителя начин. Важна задача на софтуера е той да бъде полезен и удобен за крайния си потребител и да улеснява максимално работата му.

2.3.1.6 Оценка

Общата оценка на текущата реализация е по-скоро негативна. За една система, изградена върху тези цели, е важно да бъде скалируема. Освен това трябва да е удобна за поддръжка, защото, ако кодът не е написан добре, той трудно ще бъде променян и поправян при нужда.

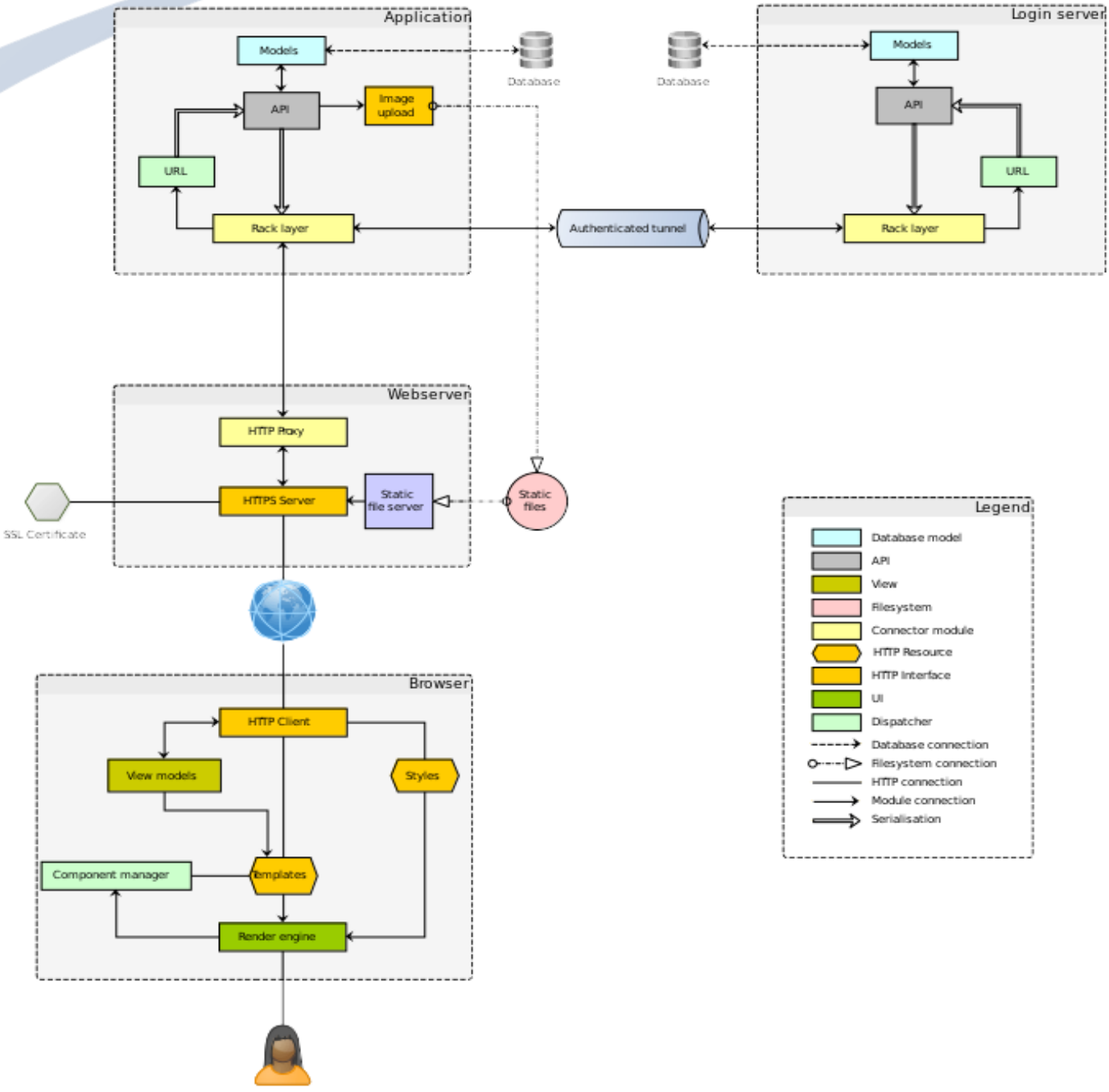
Когато се измисля архитектура за проекта е важно да се измисли начин за добро обособяване на отделните елементи. Това би решило голяма част от проблемите, които текущо съществуват в релизацията, като твърде малко едновременно изпълняващи се заявки и трудност при добавяне на нов град. Трябва да се избягва зависимост между логически независимите компоненти в една система, както се получава в bikesurf.org.

2.3.2 АЛТЕРНАТИВНА АРХИТЕКТУРА

2.3.2.1 Технология

Изборът за технология, на която да се реализира архитектурата, е комбинация между Ruby за backend и javascript и HTML5 за frontend. Това позволява по-добра поддръжка, заради избора на предимно млади и навлизащи езици за програмиране, както и ясно разделение на отгворностите.

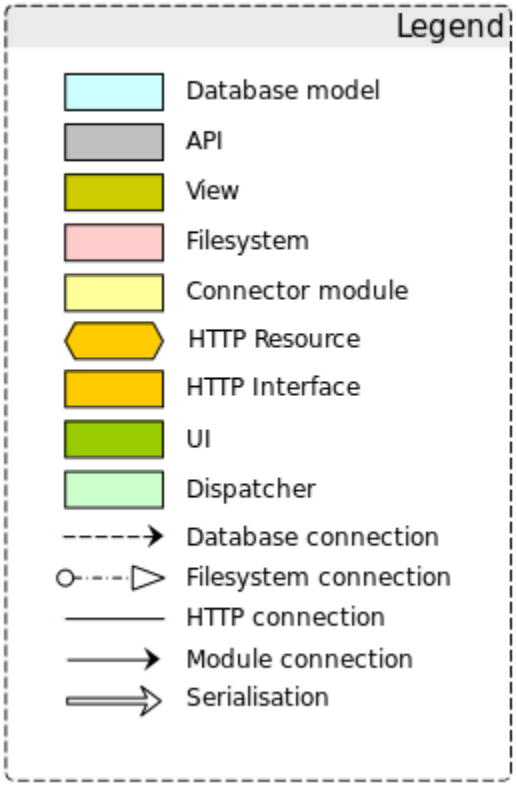
2.3.2.2 Общ преглед на архитектурата



Фиг. 7. *C&C диаграма - общ преглед на архитектурата*

Моделите на базата данни си комуникират с публично приложно-програмен интефейс (API). Той комуникира с HTTPS сървъра през Rack слоя. HTTPS сървъра комуникира с клиента. Браузърът получава тази информация и обработва във своите

компоненти и я представя чрез View модели на крайния потребител. За четимост, всеки компонент на C&C диаграмата е кодиран в определен цвят в легенда, която е изкарана на фигура 8.



Фиг. 8. *Легенда на елементите на архитектурата*

Архитектурата на нашия проект е от тип клиент-сървър, понеже той е уеб базиран, но за разлика от оригиналния проект използваме трислойна архитектура, защото ползволява по-голяма гъвкавост. Чрез този вид архитектура се осигурява разделение между обработката на приложението и обработката и съхранение на данни. Освен всички други предимства на модулния софтуер, трислойната архитектура е проектирана да позволява независима промяна на който и да е от трите слоя, така че да могат да се отразят промените в изискванията или смяната на технологиите. Клиентът, в лицето на браузъра, включва в себе си презентационната

и част от бизнес логиката, за разлика от предишния проект. Пример за това са верифицирането на формите за търсене на колела и организира достъпа на потребители до отделни компоненти на системата, спрямо правата, които имат. Сървърът използва хранилище за данни, което в нашият случай е релационна база данни.

Може да се каже, че приложението ни е центрирано около бази данни (Database-centric). Това ни позволява да изпозваме динамична търсачка, а това е едно от функционалните изисквания за проекта.

Тази архитектура може да бъде разгледана на фигура 7.

2.3.2.3 Описание на алтернативната архитектура

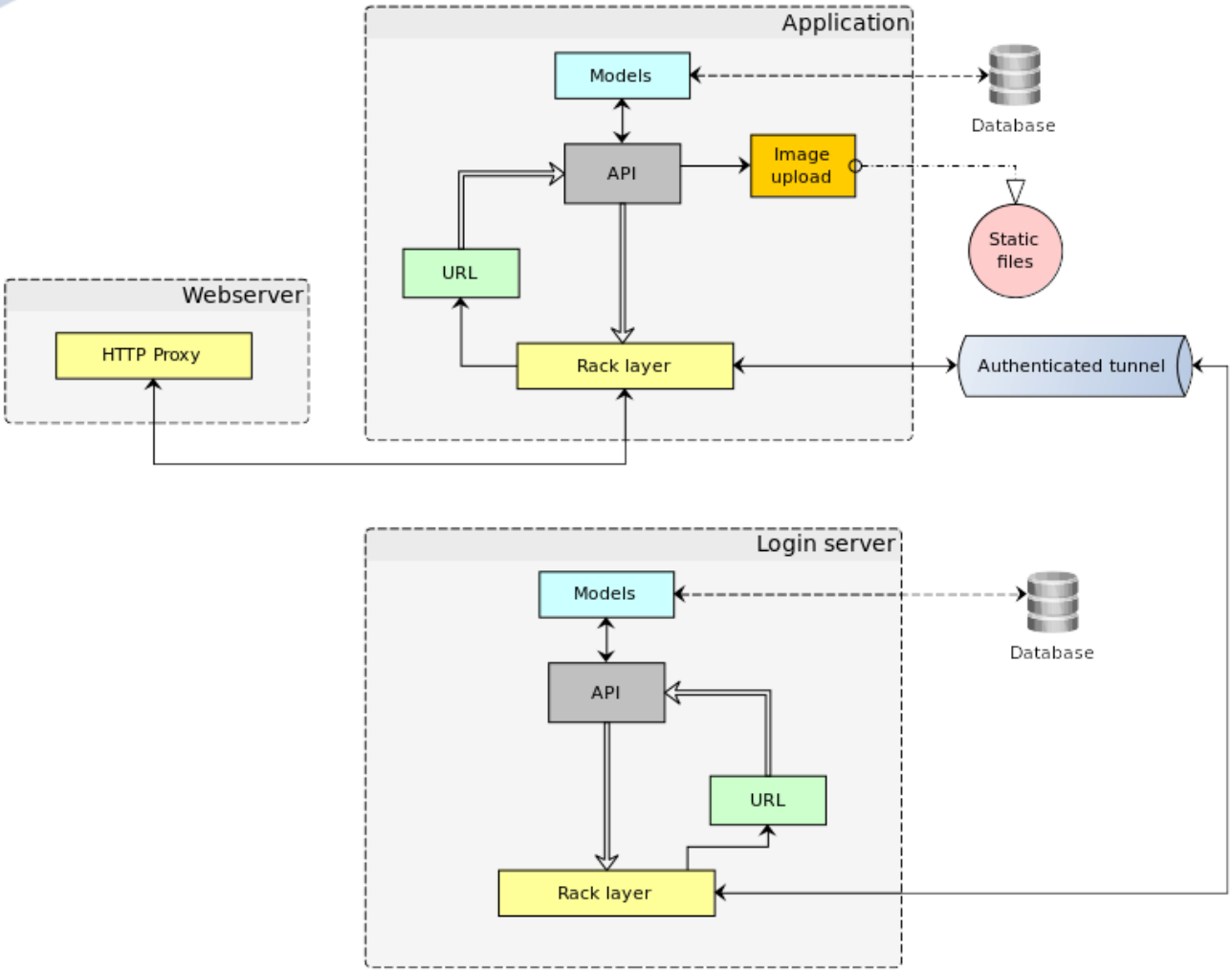
Тук ще опишем по-подробно различните слоеве на нашата архитектура.

- Първи слой

  Първият слой е слоят на данните, който съдържа схемата на базата и физическата информация. Базата данни, която използваме, е Postgresql [PSQL]. Базата данни се използва за съхранение не само за информация за потребителите, а и информация за колелата, стендовете, резервациите. Това ни ползволява динамично да правим сложни заявки както от фронтенда и бакенда. Потребителите на сайта е улеснен, понеже не трябва да търси статични картинки в браузъра, а може направо да филтрира информацията, за да получи точно каквото иска. Релационната база ни позволява единна репрезантация и съхранение.
- Втори слой

  Това е слоят, в който се реализира по-голямата част от бизнес логиката. Той е междинен слой между базата и браузъра. В него се обработват данните във вид, готов за показване. Има два големи модула – логин сървър и основно приложение.

1. Логин сървър

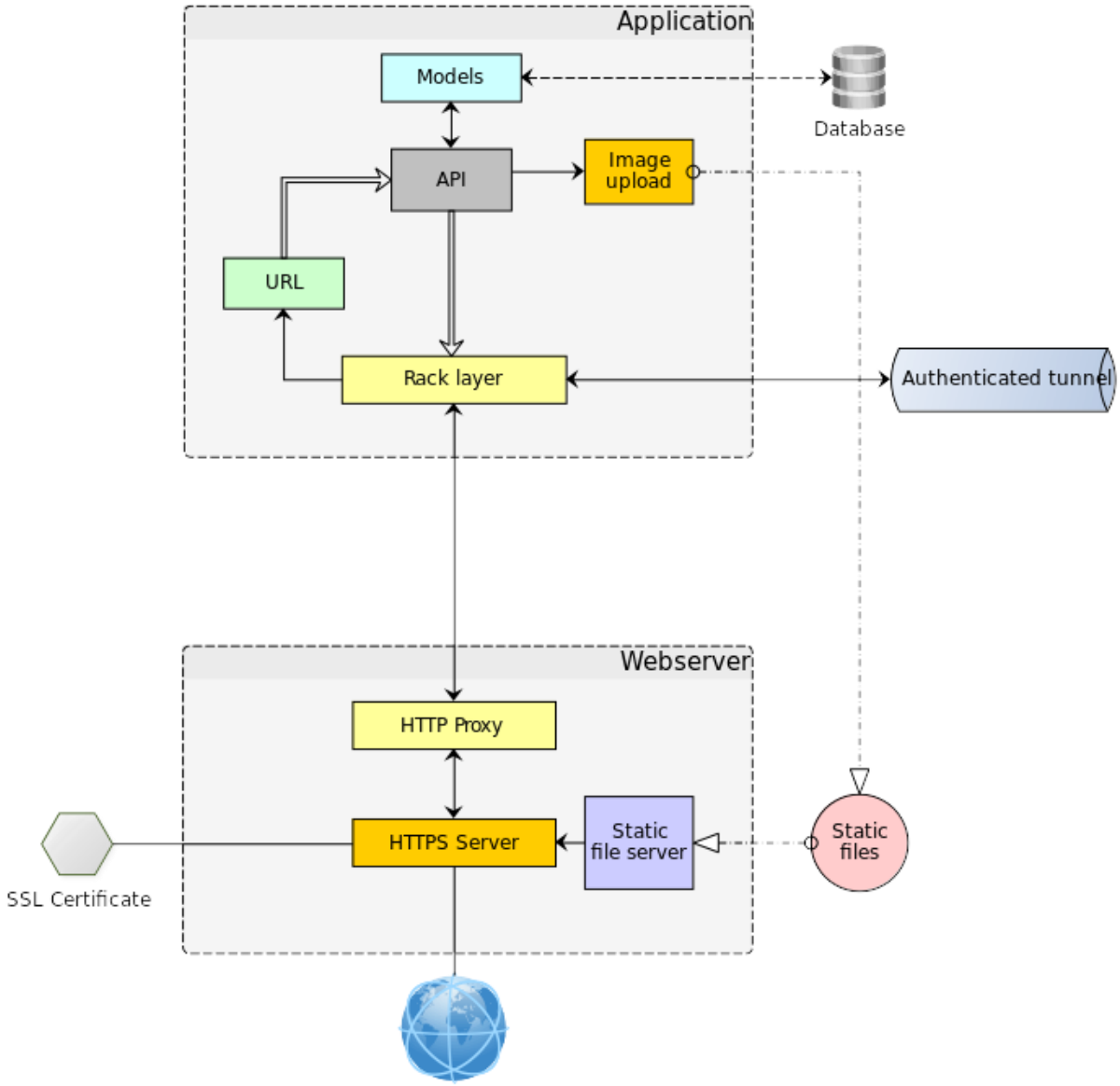


Фиг. 9. *Връзка между основно приложение и логин сървър*

Логин сървъра служи като портал, през който се вписва или регистра потребител. По този начин проектът става скалируем. Добавянето на нов град означава закачане на нов сървър за град към логин сървъра. Така потребителят се логва на едно място и след това бива препращан към сървъра на града, който е избрал. Добавянето и изтриването на градове не влияе на потребителската база, понеже акаунтите на всички потребители, заедно с настройките им, се пазят на логин сървъра. Порталът също ни предоставя гъвкавост на системата. При падане на един от град-сървърите, цялостната

система продължава да функционира. Моделите, които служат за комуникация с базата данни, описват потребителите, техните лични настройки, градовете и техните сървъри. За всеки от моделите има съответна таблица в базата данни, и връзката между базата и логин сървъра се осъществява чрез DataMapper [DTMPR] обекти. Екраните на логин сървъра представляват форма за регистрация и форма за избор на град, като всяка от тях реализира HTTPS заявка към сървъра, за да достъпи необходимата информация от базата данни. Както може да се види на фигура 9, информацията тече по криптиран канал.

2. Основно приложение (сървър на града)



Фиг. 10. *Комуникация на основното приложение с уеб сървъра*

Основното приложение се занимава с по-голямата част от бизнес логиката на проекта. Всяка инстанция на този сървър представлява един град. Някои от действията в главното приложение изискват автентикация, при което потребителят бива препратен към логин сървъра. След като потребителят се логне, той вече е удостоверен да извършва тези действия. Базата данни за всеки град е отделна, което ни осигурява модулярността на по-високо ниво.

2.1. Модели

Моделите се използват за улесняване на комуникацията с базата данни. Базата данни се свързва с концепцията за обектно-ориентираните езици за програмиране като се създава виртуална-обектна база данни. Тази технология се нарича обектно-релационно съответствие (ORM). В нашия проект тя е осъществена чрез Ruby библиотеката DataMapper. В реализацията на проекта е създаден клас за всяка отделна същност, за разлика от оригиналния проект, където има клас само за Колело.

2.2. Приложно-програмен интефейс (API)

API е начин за унифициране на обръщенията към базата, като привежда данните в удобен вид. Този модул комуникира с останалите компоненти чрез специален вид конектори, които служат за сериализиране и десериализиране на данните. Идеята за добавяне на API в приложението е, че може да бъде преизползван, ако се смени начинът на показване на данните, тоест фронтендът.

2.3. Rack layer и HTTP Proxy

Тези модули осъществавят комуникацията между приложението и уеб сървъра. приложно-програмен интефейс. Rack слоят служи като посредник между руби кода и сървъра, като привежда HTTP заявката в четим от интерпретатора на руби вид. Когато получава отговор от уеб сървъра, той пренасочва заявката към друг модул, който се грижи за извикването на правилните функции в кода.

2.4. URL

Тази част от приложението представлява диспечер на заявки – той слуша за отговор от свързващите модули и, в зависимост кой адрес е достъпен, пренасочва към правилната функция.

2.5. Image upload – работа с изображения

Функционалните изисквания към проекта включват работа с изображения, за да може да се поддържа галерия на колелата. За да не се пренатоварва

базата данни, те се запазват на статичен сървър и в базата се пазят само пътищата до тях. Това може да се случи през API, при поискване на клиента и извикване на съответната функция в бекенда.

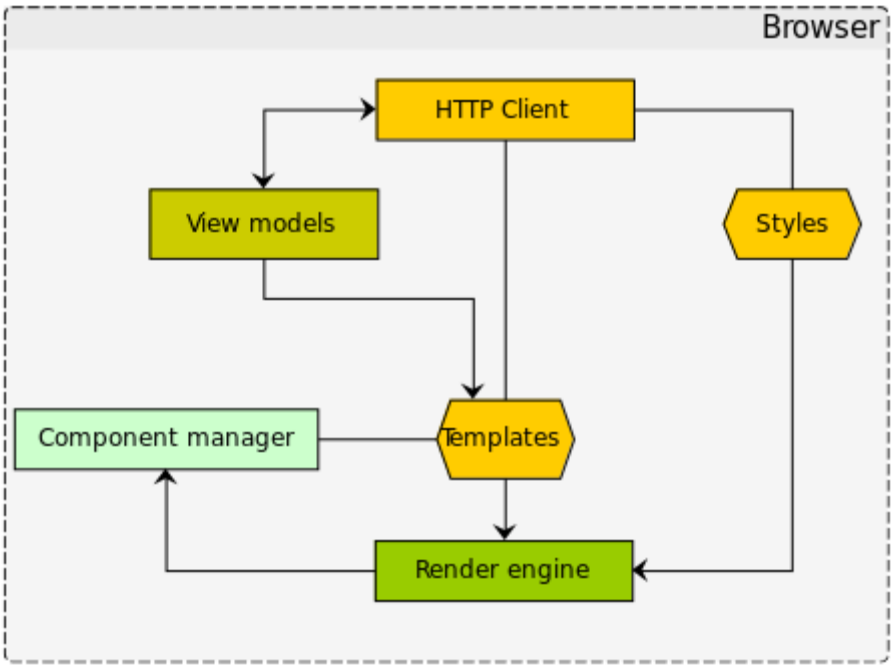
2.6. HTTPS server

Тъй като работим с чувствителни данни, е важно да има криптирана връзка. Това е свързано с нефункционалното изискване за сигурност, което е и най-сериозният проблем на оригиналното приложение на bikesurf.org. Такава връзка се осъществява чрез използването на HTTPS заявки за комуникация между отделните части на приложението. HTTPS протоколът е наложен с инсталация на SSL сертификат на сървъра.

- Трети слой

Тъй като сме уеб базирано приложение, третият слой е в браузерът. Той се нарича презентационен слой и описва както дизайна на страницата, така и някаква малка част от бизнеслогиката.

Архитектурата на презентационния слой е описана на Фиг. 11. Да я разгледаме по-подробно.



Фиг. 11. *Презантационен слой*

1. Мениджър на компоненти

   Мениджърът на компонентите отговаря за избирането на текущия компонент според URL-а на текущата страница. Компонентът отговаря за репрезентирането на една страница от сайта. Всеки компонент разполага със собствен Изглед (View) и собствен модел на изгледа (Veiw Model). Всеки път, когато бъде сменен URL-ът на сайта, се зарежда съответният компонент, като при зареждане на компонента се визуализира шаблонът му.

2. Модел на изглед (View Model)

   Всеки компонент има модел на изгледа, който е свързан с изгледа (шаблона) и реализира функционалността на определена страница на сайта. Освен, че реализира действията при кликане на различни части от страницата, моделът отговаря за обработването на информацията, подадена от потребителя, преди тя да бъде изпратена към сървъра под формата на HTTPS заявка.

3. Шаблон (Template)

   Шаблонът на един компонент представлява HTML кода за тази страница, който се зарежда при избиране на компонента. Тъй като приложението ни е във вид на една страница (Single Page Application), този шаблон се зарежда на точно определено място на страницата, което обаче от гледна точка на нормалния потребител изглежда като че сайтът си е сменил адреса и е заредил нова страница. За да избегнем постоянното зареждане на шаблон от файл, след като шаблонът бива зареден за пръв път, той се запазва в кода на страницата, за да може при следващо извикване да се достъпи директно.

4. HTTP клиент

   HTTP клиентът отговаря за пращането на заявки към сървъра и получава и предава отговорите към моделите за изглед или към визуализатора.

5. Стилове

   Стиловете се вземат от HTTP клиента чрез заявка към сървъра и биват връщани към визуализаторът, който ги използва в дизайна на страницата.

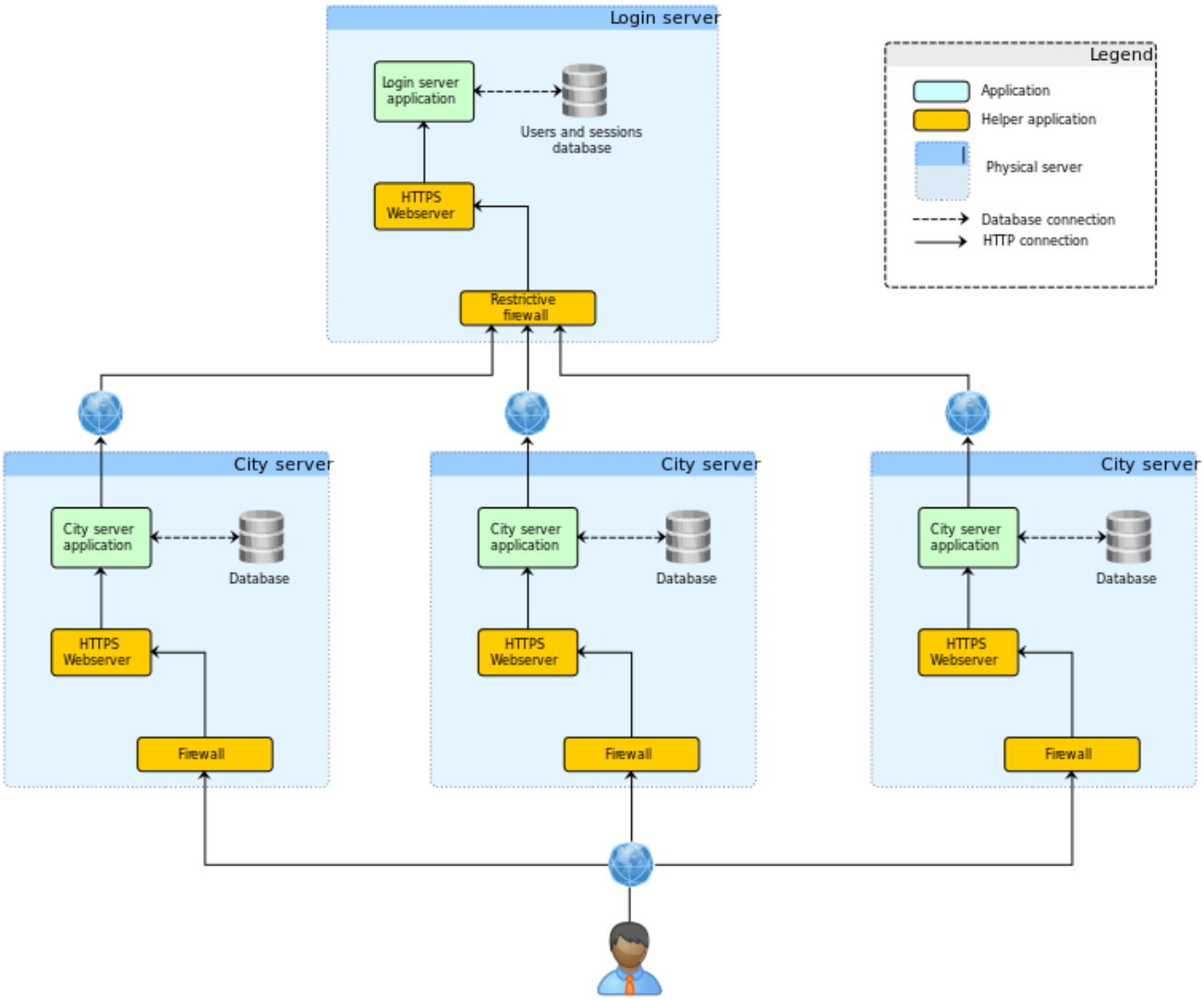
6. Визуализатор

   Визуализаторът зарежда шаблоните и стиловете на екрана и визуализира различните елементи от страницата.

2.3.2.4 Инсталация и deployment

Инсталирането на проекта и пускатенто на сървъра се случва с прост скрипт. Това е възможно благодарение на използваните технологии и реализацията на деплоймънт процеса в кода. Всички зависимости се инсталират автоматично. Така че вдигането на сървър за град е почти автоматизирано, като се изключи, подготвянето на средата(инсталирабне на Руби и Bundle) създаването на базата данни и попълването на променливите на средата. Като всичко това може да се автоматизира на по- късен етап.

Единствената трудност възниква от това, че имаме отделен сървър, който служи като портал за пренасочване към сървърът на спецефичен град. Трябва да се конфигурира логинг сървъра да праща заявки към web server-a (nginx в нашия случай), да се създаде база – за което вече има автоматизиран скрипт. След като се вдигне сървър за град, администратор трябва ръчно да въведе в базата на логин сървъра запис за пренасочване към новосъздадения сървър за град.



Фиг. 12. *Връзки между сървърите на приложението*

2.3.2.5 Предимства

Основото предимство на алтернативната архитектура е, че тя е скалируема. Това се базира на няколко избора при проектирането. Първият е силната модулярност на проекта, което дава възможност за лесно интегриране на системата на нов компютър. Вторият е използването на база данни, още повече използване на ORM за обръщение към нея. Това дава възможност за използване на която и да е sql базирана база данни,а използването на такава гарантира бързи и удобни заявки.

Друго важно предимство е гъвкавост. Всъшност, това е много важно свойство за един проект, а алтернативната архитектура демонстрира гъвкавост какво в

бекенда, така и във фронтенда. Това е така заради изборът на технология – от страна на сървъра ORM модели, а от страна на браузера собствени HTML темплейти. Като цяло алтернативната архитектура не зависи от кой знае какви готови компоненти и позлволява гъвкавост при дизайна на продукта.

Когато говорим за предимства, не бива да пропускаме сигурността, която ни осигурява изпозлването на HTTPS и валидирането на потребители в отделен сървър. Това ни предпазва до някъде от възможно изтичане на чувствителни данни.

2.3.2.6 Недостатъци

Въпреки многото предимства, като всеки продукт този има и своите недостатъци. Всъщност някои от най-важните предимства имат странични ефекти, които трябва да споменем.

Фактът, че алтернаивната архитектура предлага силно разделение, ни поставя пред проблема за управлението им. Всеки сървър ще има нужда от отделно менажиране, което означава, че трябва да има специални хора, които се грижат за това.

Друго следствие от архитектурата е, че при проблем в сървърът за верификация на потребители, не могат да се създават сесии изобщо. Това означава, че потребителите в отделните градове няма да могат да достъпят профила си до времето на оправяне на сървъра.

Вярно е, че текущият дизайн предоставя голяма гъвкавост при изграждането на фронтенд и изборите, свързани с дизайна. Това обаче изисква и клиентските устройства да са по “умни”, защото в такава регистрация имаме изпълнението на javascript при самия клиент.

2.3.3 СРАВНЕНИЕ И ОЦЕНКА

Освен да разглеждаме поотделно архитектурите, трябва да ги съпоставим, за да може да изкажем крайна оценка.

2.3.3.1 Изпълнение на функционалните изисквания

Спрямо изпълнението на функционалните изисквания, може да се каже, че алтернативната архитектура е по-добра. Това е така, защото тя изпълнява всички функционални изисквания, за разлика от оригиналната архитектура. Например, желанието да има динамична търсачка е удовлетворено, благодарение на базата данни.

2.3.3.2 Изпълнение на нефункционални изисквания

Естествено, не маловажно е как се държат двете системи по критериите на нефункционалните изисквания. Тях отново можем да разгелдаме, като използваме основното разделение на категории:

- Използваемост

  Относно добрият изглед, може да се каже, че алтернативната архитектура има превес, заради неинтуитивния дизайн на оригиналната. Това се случва поради лошо разделение и дизайн при bikesurf.org.

  Наблюдаваме по-интересен резултат при поддържането на различни браузери. Оригиналната архитектура, например, върви на почти всички такива, защото клиентската страна няма никаква друга роля, освен показването на HTML-и, а за това не се иска кой знае какво. От друга страна, алтернативната архитектура изпълнява javascript на браузера, което е поне някакво минимални изискване. Въпреки това, новото предложение изпълнява нефункционалното изискване да върви на всеки от изброените браузери в NFR105.

  За поддръжката на кода сравнението е трудно, защото кодът е твърде различен. Оригиналната архитектура е лесна за поддръжна, защото използва готови шаблони от Django. Трудност има обаче при желание за промяна, защото дървеният дизайн прави това изключително трудно. Новата архитектура, от друга страна, изисква малко повече писане, за да се достигне изградения в проекта стандарт на разделение. Това ни носи много по-гъвкав и скалируем продукт. За един продукт от такъв размер е по-важно да може да се

развива, отколкото всеки да може да го поддържа, ако това развитие не е с цената на твърде много труд. В случая допълнителните изисквания на алтернативната архитектура не биха затруднили никой програмист, запознат с продукта.

Алтернативната архитектура изпълнява по-добре изискването NFR115, а именно визуализация на различни екрани. Това е така, защото използва технологии, които позволяват динамично оразмеряване на компонентите, за разлика от оригиналната архитектура.

Трудно е да се премери точно колко "лесно използваем" е един продукт. Проведохме тест с реални клиенти, които да погледнат едната и другата реализация. Мнозинството смятат новия дизайн на сайта за много по-интуитивен, така че и тук точката взима алтернативната архитектура.

Относно допълнителния софтуер, нямаме оплакване – тук и двете архитектури изпълнява изискването.

- Надеждност

  Отново трудна категория за сравнение, защото липсва пълната информация да се направят изводи. Въпреки това, можем да кажем, че в оригиналната архитектура е много по-трудно да се локализира проблема, заради липсата на разделение, което може значително да забави времето за възстановяване.

- Модулярност

  Относно изискванията за модулярност, няма съмнение, че по-добре изпълняващият изисквания проект е алтернативния. Цялата му архитектура е съсредоточена в добре модулирани компоненти.

  Оригиналната архитектура не спазва дори изискване NFR305 – самостоятелни сайтове на градове – всичко върви на един сървър и, при проблем в сайта на един град, нито един не работи. Това е много сериозно нарушаване на изискване, с което алтернативната архитектура се справя. Макар че е вярно, че при неработещ login сървър не могат да се създават сесии, поне за клиента е достъпна някаква част от функционалността на проекта. Той може да разглежда страниците на колела и да използва филтъра. Ако такъв проблем

настъпи при оригиналната архитектура, клиентът не е способен да достъпва сайта изобщо.

- Ефикасност и бързодействие

  Сериозно предимство на алтернативната архитектура се вижда в при ефикасност. При проведените тестове, оригиналният сървър вдигна времето си на 15 секунди, когато сървърът се натовари.

  За бързодействието може да се каже, че и двете архитектури нямат проблем с изискването.

- Поддръжка

  Тъй като в оригиналната архитектура всъщност няма много код, не можем да кажем, че той рязко нарушава стилови изисквания. Разликата при поддръжката се вижда при правило NFR510 – единственият тест при оригиналната реализация проверява дали 1+1=2. Тук сериозно предимство има алтернативната архитектура.

- Сигурност

  Дори няма нужда да се коментира, че тук алтернативната архитектура се справя по-добре. При оригиналния проект има липса на криптирана връзка. Дори няма сертификат на сървъра. Паролите се пращат в прав текст, но дори това не е най-лошото. Много по-чувствителна информация като снимки на лични карти и паспорти са напълно публично достъпни.

- Скалируемост

  Това е другата категория, в която оригиналният проект не спазва нито едно от изискванията.

  Системата спира да бъде стабилна много преди 5000 заявки, когато сървърът пада, за разлика от алтернативната архитектура. При нея няма никакъв проблем, заради избора да се използва rack.

  Като цяло оригиналният проект е изключително нескалируем. Това е така, поради различките на първия слой от архитектурата. За добавяне на ново колело, при оригиналния проект трябва да се променят картинки. Докато при новата архитектура само се прави запис в базата данни. Старият проект не

само че не покрива изискванията, ами и създава работа на крайния потребител, което е недопустимо.

2.3.3.3 Оценка

Като всеки реален софтуерен проект, не можем да кажем, че алтернативната архитектура няма своите проблеми. Въпреки това, изискванията, които налага, са необходимни и напълно постижими. Те не струват почти нищо за всеки програмист от гледна точка на поддръжката на кода, а и изискванията за браузера наистина не са много. Казвайки това, трябва да отблежим, че алтернативната архитектура се държи много по-добре от оригиналната и при функционалните, и при нефункционалните изисквания към проекта. От тази гледа точка, нейните недостатъци са нищожни пред плюсовете, които носи.

3 Заключителни коментари

Проектът Bikesurf е предизвикателен, защото трябва да се постигне баланс между лесна поддръжка и гъвкава структура. Проектът хем трябва да е скалируем, хем да не създава допълнителна работа за програмистите. Точно заради това, алтернативната архитектура, която е предложена в проекта, е точното решение за подобен тип проблеми. Тя добре балансира в структурата си тези противоречиви изисквания и създава добър краен продукт. Като програмисти ние знаем, че удобството и защитата на клиента стоят на челно място при изборът на архитектура. Точно затова предимствата, които носи новата архитектура, не са реализирани за сметка на сигурността на потребителите, което я прави и отличен избор за проекта bikesurf.

Библиография

[SNTC] https://tfl.gov.uk/modes/cycling/santander-cycles – Сайт на Лондонската система за колела (последно използвана на 02.2017)

[MSCW] https://en.wikipedia.org/wiki/MoSCoW_method – MoSCoW метод за приоритизация на изисквания (последно използвана на 02.2017)

[BKSRF] https://github.com/F483/bikesurf.org – Хранилището на текущата имплементация на сайта (последно използвана на 02.2017)

[DJNGO] https://www.djangoproject.com/ - Уеб фреймуърк за Пайтън (последно използвана на 02.2017)

[RACK] https://rack.github.io/ - Минималистичен интерфейс посредник между уеб сървър и Руби (последно използвана на 02.2017)

[SNTRA] http://www.sinatrarb.com/ - Руби фреймуърк за рутиране (последно използвана на 02.2017)

[PSQL] https://www.postgresql.org/ - База данни (последно използвана на 02.2017)

[DTMPR] http://datamapper.org/ - ORM за Руби (последно използвана на 02.2017)