Содержание

Введение

По данным Всемирной туристской организация ООН (UNWTO), ежегодно более одного миллиарда человек совершает поездки заграницу [1]. Если же добавить сюда тех, кто перемещается по городам своего региона или страны, то общее число путешественников число станет по-настоящему огромным.

С помощью сервиса, представленного в данной работе, путешествия станут еще интереснее: маршрут, отображенный на интерактивной карте, позволит не сойти с пути и посетить все намеченные места, а хорошо составленный дневник о поездке, даже через многие годы поможет вновь пережить приятные былые ощущения.

Актуальность работы становится понятна, если оценить как много людей делится историями своих путешествий в социальных сетях.

Новизна данной работы выражается в отсутствии на данный момент аналогов, со столь же широкой функциональностью. Практическая ценность работы заключается в появлении работающего продукта, который можно использовать в реальной жизни.

Целью данной работы является создание сервиса, позволяющего пользователям в интерактивной форме вести блог о своих путешествиях. Для наилучшего достижения этой цели при написании работы были поставлены следующие задачи:

- провести анализ предметной области: в частности, рассмотреть существующие решения, обозначить целевую аудиторию;
- составить техническое задание, наиболее полно описывающее желаемый результат;
- путем сравнения между собой, осуществить выбор средств реализации, которые наилучшим образом будут удовлетворять функциональным потребностям сервиса.

1. Анализ предметной области

1.1 Обзор существующих решений

Прежде всего рассмотрим уже функционирующие проекты, назначение которых близко к задуманному нами. Анализ их достоинств и недостатков позволит лучше продумать структуру нашего сервиса, его интерфейс, а также некоторые функциональные возможности.

В результате поиска в сети Интернет, было найдено 3 ресурса, каждый из которых в различной степени близок к задуманному нами сервису. Это веб-сайты:

- TravelMap [2];
- Tripline [3];
- TripBlan [4].

TravelMap

Из всех найденных ресурсов этот веб-сайт является наименее функциональным. Он позволяет пользователям наносить места на карту и добавлять к каждой геометке текст или изображения. Среди интересных функций можно выделить подсчет дистанции между соседними геометками, а также возможность указать каким способом было преодолен каждый из отрезков (пешком, на велосипеде и т.д.).

Сайт предлагает 3 типа отображения материала. Все они представлены на рис.1, это:

- карта + краткие заметки с фотографиями;
- только фотографии;
- только заметки с фотографиями.

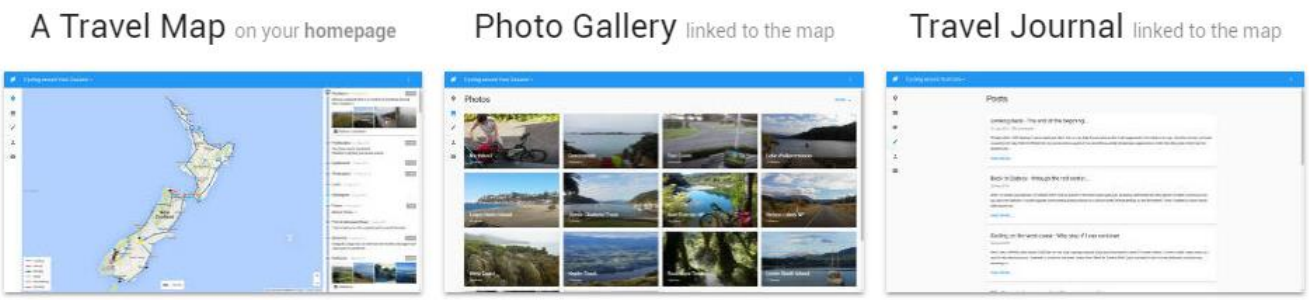

Рисунок 1. Типы отображения материала

На рис. 2 представлен тип отображения «карта + краткие заметки с фотографиями». Такой тип отображения близок к тому, что задумано нами, однако если в данном случае визуальный акцент сделан на карту (ее размер вдвое превышает непосредственно сами заметки), то в задуманном нами сервисе именно заметки являются ключевым элементом страницы, а карта лишь дополняет их.

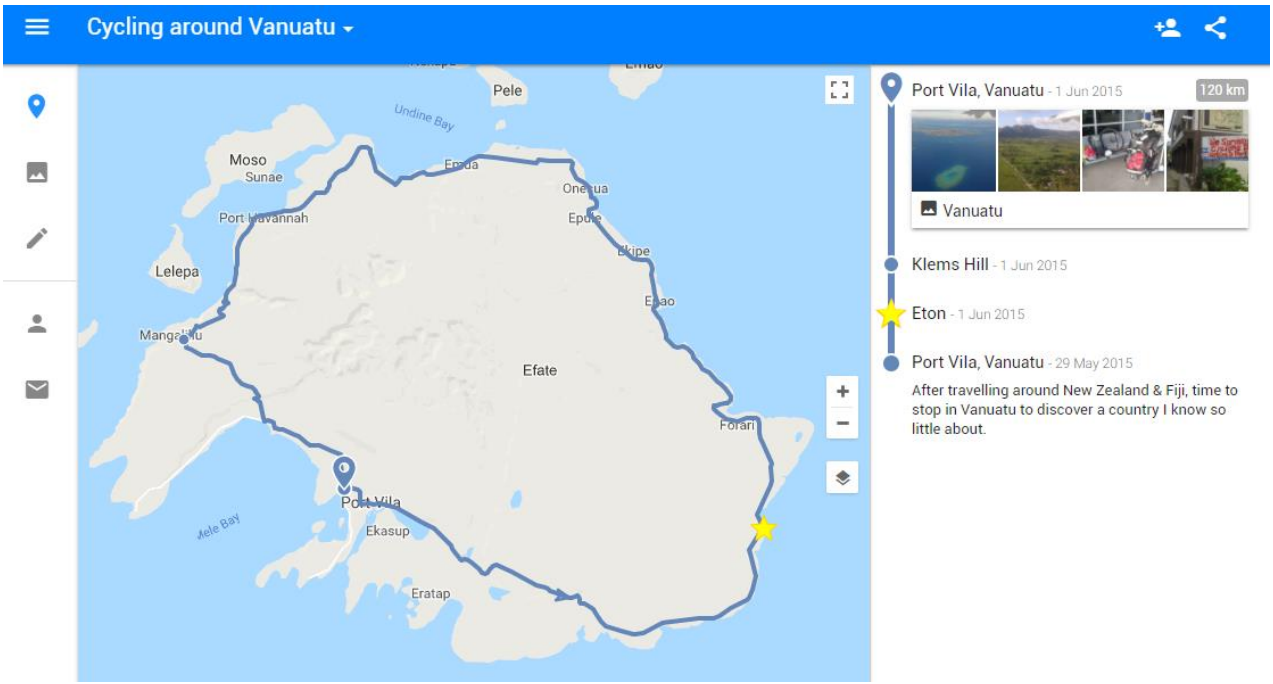

Рисунок 2. Тип отображения «карта + краткие с заметки с фотографиями»

По информации на сайте, в базе содержатся сведения о почти 50 000 путешествий, а количество пользователей составляет 42 000.

Tripline

Этот сайт является наиболее старым из рассматриваемых (первое упоминание датируется 2009-м годом). В своем текущем исполнении он уже больше похож на задуманное нами, нежели ресурс, рассмотренный перед этим.

Акцент в данном случае сделан именно на заметках (рис. 3), но при этом карта (рис. 4) существует отдельно от них, и при чтении материала перемещения на карте не отображаются.

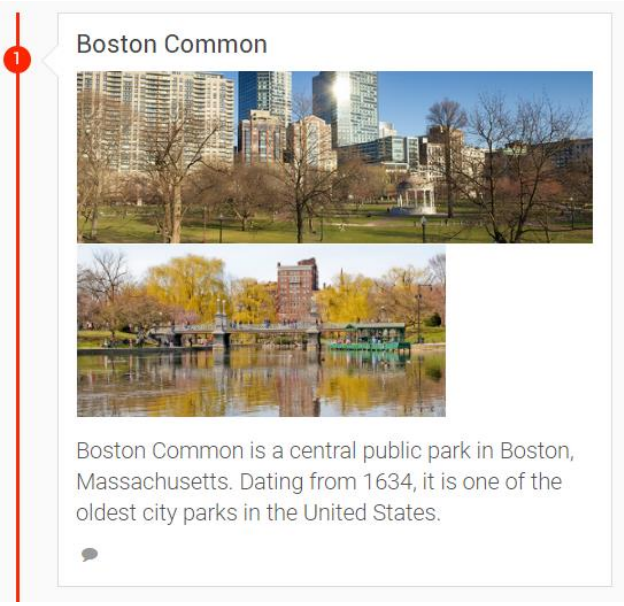

Рисунок 3. Внешний вид заметки на сайте Tripline

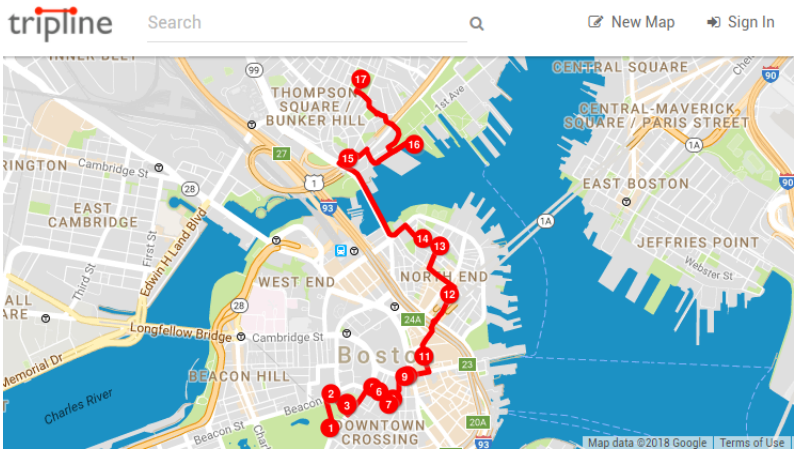

Рисунок 4. Внешний вид карты на сайте Tripline

Из сильных сторон сервиса можно выделить функционирующие мобильные приложения для операционных систем Android [5] (количество

скачиваний более 1 000) и iOS [6] (количество скачиваний достоверно установить не удалось).

Среди слабых сторон — факт размещения большого количества спам-контента, под видом путешествий. Модерация таких записей, по-видимому, отсутствует.

TripBlan

Этот ресурс, напротив, является самым молодым (первое упоминание датируется ноябрём 2017-го года), и при этом как визуально, так и функционально он наиболее схож с задуманным нами сервисом. В отличии рассмотренных раннее сайтов, визуальный акцент в данном случае сделан именно на заметки, а карта при этом является лишь хорошим интерактивным дополнением (рис. 5).

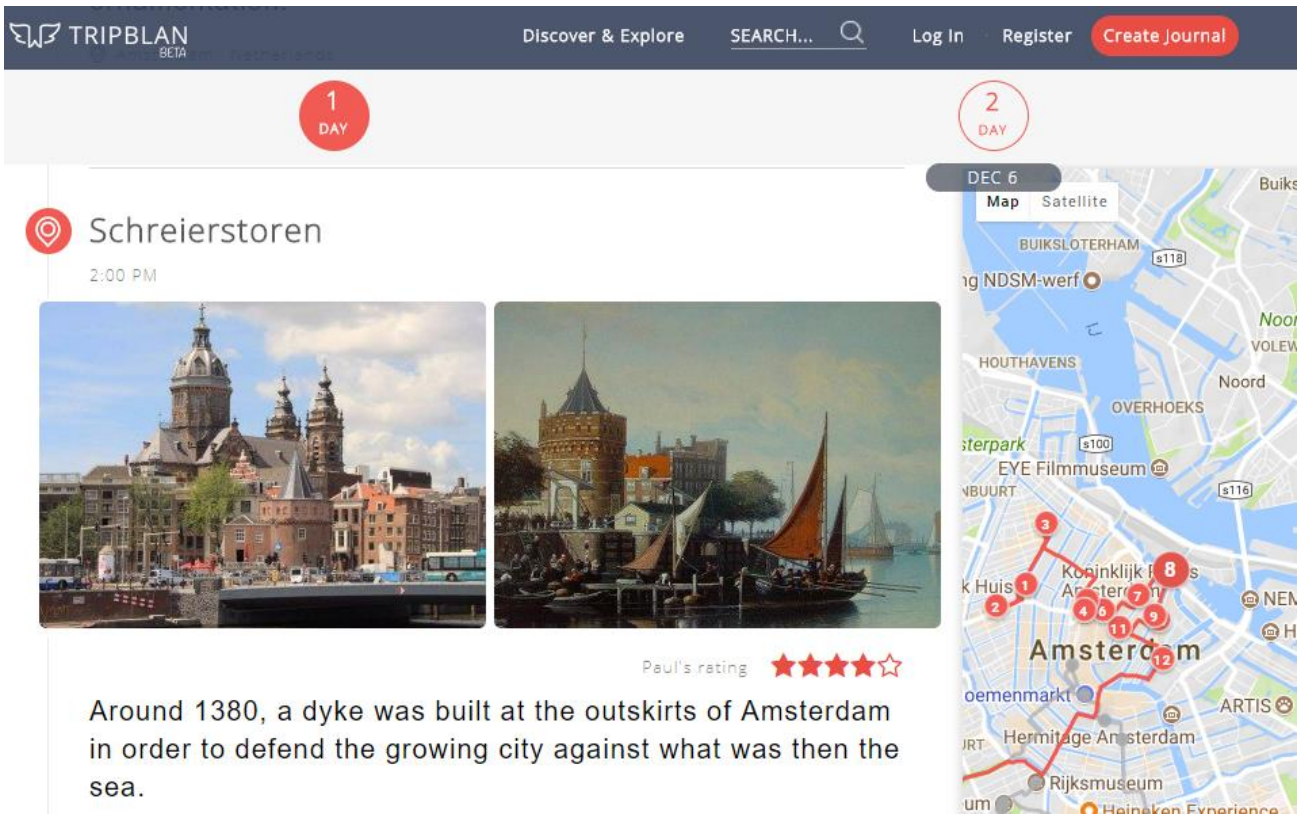

Рисунок 5. Интерфейс сайта TripBlan

На данный момент сайт хранит информацию лишь о небольшом числе путешествий (около 30), однако по информации от разработчиков в скором времени ожидается выход приложений под операционную систему iOS.

1.2 Сравнение существующих решений

Проведем сравнительный анализ обнаруженных веб-сайтов как между собой, так и с нашим сервисом. Для этого построим таблицу с наиболее значимыми критериями (табл. 1).

Таблица 1. Сравнительный анализ существующих решений.

Критерий	**TravelMap**	**Tripline**	**TripBlan**	**Разрабатываемый сервис**
Год возникновения	2014	2009	2017	2018
На чем сделан акцент	карта	заметки	заметки	заметки
Наличие интерактивной карты	да	да	да	да
Синхронизация карты с заметками	нет	нет	да	да
Регистрация с помощью социальных сетей	Facebook, Google	Facebook	Facebook	Facebook, Google
Наличие мобильных приложений	нет	Android, iOS	нет, планируется к выпуску для iOS	нет, но будет предоставлен API
Карта посещенных стран	нет	нет	да	да
Поиск по пользователям	нет	да	нет	да
Поиск по путешествиям	да	да	да	да
Языки интерфейса	английский, французский	английский	английский	русский, английский
Возможность просмотра только фотографий	да	нет	да	да
Метеорологические данные в заметках	нет	нет	нет	да

Можно заметить, что нет сайта, который удовлетворял бы всем перечисленным критериям. По этой причине разработка собственного сервиса становится еще более актуальной.

1.3 Целевая аудитория

Чтобы лучше понимать, как именно должен быть устроен наш проект, необходимо обозначить его целевую аудиторию. Для этого исследуем статистические данные сайтов, тематика которых близка к тематике нашего сервиса, а именно:

- сайты о путешествиях;
- сайты по продаже авиабилетов;
- сайты бронирования отелей.

Данные будем получать из открытых источников — через используемые сайтами счетчики статистики, либо непосредственно через сами сайты, статистические данные на которых как правило указаны в разделах для рекламодателей.

Ежемесячная русскоязычная аудитория популярного сайта бронирования отелей Ostrovok [7] составляет более 3 000 000 человек. Гендерный портрет аудитории выглядит следующим образом:

- женщина — 67%;
- мужчина — 33%.

Возрастной портрет, в свою очередь, представлен следующим образом:

- посетители от 25 до 34 лет — около 34%;
- посетители от 35 до 44 лет — около 22%;
- посетители от 45 до 54 лет — около 18%.

Сайт о путешествиях Тонкости туризма [8] ежемесячно также посещает более 3 000 000 человек. По данным сервиса статистики LiveInternet [9], наиболее популярными демографическими группами среди его посетителей являются [10]:

- женщины (от 25 до 34 лет) — 21.1%;
- женщины (от 18 до 24 лет) — 20.7%;

- мужчины (от 25 до 34 лет) — 15.3%;
- мужчины (от 18 до 24 лет) — 12.4%.

Таким образом, можно предположить, что нашим сервисом преимущественно будут пользоваться люди, в возрасте от 18 до 55 лет, причем среди женщин доля будет выше.

По своему поведению это активные Интернет-пользователи, имеющие аккаунты во социальных сетях. По образу жизни это люди, предпочитающие активных отдых.

2. Техническое задание на работу

2.1 Предмет разработки

Предметом разработки является сервис, состоящий из веб-сайта, посвященного созданию пользовательских блогов о путешествиях, и программного интерфейса приложения (API) [11], который в дальнейшем может быть использован для обмена данными с мобильными приложениями и иным программным обеспечением.

2.2 Требования к безопасности

- Хранение паролей должно осуществляться с использованием алгоритмов хеширования;
- Все пользователи, за исключением администратора, должны иметь возможность управлять только тем контентом, который им принадлежит. Редактирование материалов других пользователей исключено;
- Необходимо препятствовать созданию аккаунтов спам-ботами, а если это не удастся, то их последующей активности на сайте;
- Требуется осуществлять фильтрацию принимаемых от пользователей данных, использовать защиту от распространенных сетевых угроз, таких, например, как SQL-инъекции [12] или межсайтовые подделки запросов (CSRF) [13];
- Должна существовать возможность внесения любого IP-адреса в черный список, исключая таким образом возможность просмотра сайта.

2.3 Требования к графическому оформлению

Основными целями, которые необходимо достичь правильным выбором визуального оформления сервиса являются:

- удобство взаимодействия;
- его запоминаемость;
- привлекательность внешнего вида.

В визуальном оформлении сервиса должны использоваться в основном светлые и контрастные цветовые решения, создающие у посетителей впечатление чего-то позитивного и приятного.

При этом в дизайне недопустимо применение:

- любых мигающих баннеров;
- всплывающих окон, появление которых не было инициировано посетителями;
- излишне длинных текстов, не разделенных на абзацы.

2.4 Требования к представлению

Графическая структура сайта должна базироваться на основе макетов, представленных на рис. 6 и рис. 7. В частности, сайт можно разделить на логические области:

- верхняя часть сайта. Включат в себя:
  - — логотип;
  - — меню с основными ссылками.
- область осинового контента страницы;
- нижняя часть сайта. Включат в себя меню и разнообразные «служебные» ссылки вроде «О сайте», «Контактная информация» и т.д.

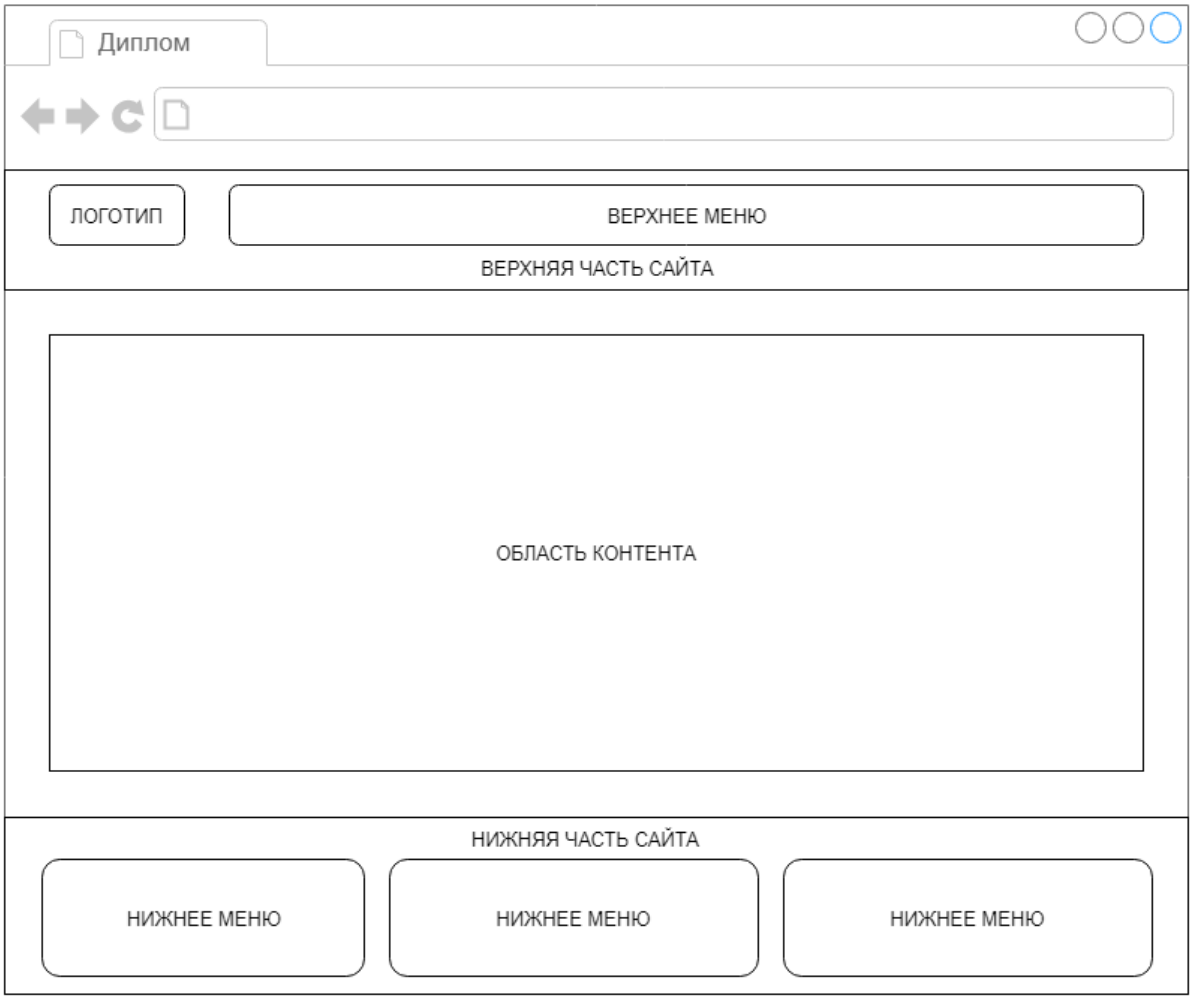

Рисунок 6. Графическая структура типовой страницы сайта

Структура страницы с историей о путешествии схожа с остальными страницами сайта, за исключением особого оформления области контента. Здесь в хронологическом порядке будут отображаться места, посещенные пользователем.

Каждая из записей может содержать текстовый комментарий, изображение или видеофайл, дату посещения места и геометку, совокупность которых будет отображаться на расположенной справа интерактивной карте.

Рисунок 7. Графическая структура страницы блога

2.5 Требования к разделению доступа

Сервисом предусмотрено 3 пользовательские роли, в зависимости от наличия которых предоставляется доступ на совершение тех или иных действий, а также доступ к различным разделам.

- посетитель (незарегистрированный пользователь);
- пользователь;
- модератор;
- администратор.

Особняком стоит роль «заблокированный посетитель», доступ которой запрещен по IP-адресу. В этом случае запрещена не только любая активность, но даже просмотр сайта становится недоступен.

В виде таблицы приведем некоторые отличия в привилегиях (табл. 2):

Таблица 2. Сравнение привилегий различных ролей пользователей.

	посетитель	пользователь	модератор	администратор
Просмотр опубликованного контента	да	да	да	да
Просмотр пользовательских профилей	да	да	да	да
Публикация материалов	нет	да	да	да
Редактирование своих материалов	нет	да	да	да
Редактирование всех материалов	нет	нет	да	да
Осуществление блокировки пользователей	нет	нет	да	да
Просмотр журнала логов	нет	нет	нет	да

2.6 Требования к локализации

Поскольку большинство пользователей сети Интернет являются англоязычными [14], то в целях увеличения потенциальной аудитории сайта, имеет смысл реализовать поддержку интерфейса сайта как на русском, так и на английском языках.

Язык при этом, может быть установлен одним из следующих способов:

1. автоматически исходя из географического расположения IP-адреса — наиболее актуально для посетителей сайта, не имеющих аккаунта;
2. принудительно через форму смены языка — наиболее актуально для уже зарегистрированных пользователей, но также доступно и обычным посетителям.

2.7 Требования к надежности

Сервис должен соответствовать следующим требованиям к надежности:

- отказоустойчивость: параметры хостинга должны быть выбраны не только в соответствие с ожидаемой нагрузкой, но даже с некоторым запасом;
- ведение журнала событий системы: своевременная информация о мелких ошибках и угрозах способна предотвратить дальнейшие более неприятные последствия;
- регулярное создание резервных копий базы данных.

2.8 Требования к административной части

Доступ к административной части сервиса должен быть разрешен только пользователям с ролью «администратор».

К функциональности административного раздела предъявляются следующие требования:

- возможность просмотра списка всех пользователей, манипуляции с ними;
- возможность просмотра всех материалов сервиса, в том числе снятых с публикации, манипуляции с ними;
- наличие журнала логов;
- возможность блокировки IP-адресов.

2.9 Требования к хостингу

Выбирая хостинг-провайдера, мы должны быть уверены, что наш сервис сможет корректно функционировать. Для этого хостинг в обязательном порядке должен поддерживать все необходимые компоненты, указанные в технических требованиях выбранного в дальнейшем фреймворка/системы управления содержимым.

2.10 Требования к взаимодействию с внешними сервисами

Предполагается, что сервис будет взаимодействовать с рядом внешних интернет-ресурсов, например, социальными сетями (для авторизации пользователей), картографическим сервисом (для отображения активности пользователей на карте) и метеосервисом (для отображения погоды). Это накладывает следующие обязательства:

- архитектура должна быть выстроена таким образом, чтобы совершать минимально возможное число запросов к внешним интернет-ресурсам. Помимо прироста производительности, это выгодно и по экономическим соображениям, поскольку, как правило, после превышения разрешенного числа запросов за указанный период, дальнейшее взаимодействие возможно только за дополнительную плату;
- сервис должен передавать внешним интернет-ресурсам только те данные, которые действительно необходимы для решения той или иной задачи.

3. Выбор средств реализации

3.1 Фреймворк, система управления содержимым

Для повышения производительности эффективности работы сервиса целесообразно использовать фреймворк или систему управлениям содержимым. Каждое из этих решений позволяет упрощает решение таких регулярно возникающих задач как:

- шаблонирование;
- маршрутизация;
- валидация и обработка форм;
- взаимодействие с базами данных.

Использование фреймворков повышает гибкость в ходе разработки и позволяет сэкономить значительное количество времени. Кроме того, большинство фреймворков имеет встроенные средства защиты. Как правило, они связаны с SQL-инъекциями, подделкой межсайтовых запросов и межсайтовым скриптингом.

Наиболее востребованным веб-фреймворками на сегодняшний день являются Laravel [15] и Symfony [16], а среди систем управления содержимым значительную долю рынка сложных многофункциональных сайтов занимает Drupal [17]. Именно эти решения мы и сравним.

Laravel — бесплатный веб-фреймворк с открытым кодом, предназначенный для разработки с использованием архитектурной модели MVC (Model View Controller — модель-представление-контроллер). Стоит отметить, что Laravel использует часть компонентов Symfony.

Среди преимуществ Laravel можно выделить:

- обширную функциональность;
- возможность создания гибкой административной панели;
- обеспечение безопасности баз данных;
- регулярные релизы;
- популярность и активное сообщество;
- масштабируемость.

Symfony, как и Laravel тоже является бесплатным веб-фреймворком с открытым кодом, и тоже придерживается архитектурной модели MVC. Он наделен такими же преимущества, однако время разработки одних и тех же задач на Symfony и Laravel будет отличаться не в пользу первого. Связано это с более строгой архитектурой. Но по этой же причине Symfony оказывается в выигрышном положении, если необходимо разрабатывать сложный, многофункциональный сайт. В этом случае у нас есть возможность детально настроить под себя любой из компонентов.

Drupal — система управления содержимым (CMS), используемая также как каркас для веб-приложений (CMF), написанная на языке PHP. Является свободным программным обеспечением. Что интересно, что, как и Laravel, Drupal 8 (последняя стабильная версия) тоже использует часть компонентов Symfony. Однако в остальном, разница существенна. Drupal уже «из коробки» дает возможность публиковать материалы разнообразного типа, создавать формы и отображать данные в таком, виде, который нам требуется. Также данная CMF берет на себя заботу о вопросах безопасности.

В итоге выбор был сделан в пользу фреймворка Symfony, который позволит реализовать задуманную функциональность наиболее гибким образом.

3.2 Серверный язык программирования

Поскольку Symfony это PHP-фреймворк, то вопрос выбора серверного языка программирования не актуален, однако мы можем выбрать его версию. Согласно требованиям фреймворка, минимально допустимой версией является 5.5.9, но целесообразно использовать наиболее актуальную в данный момент версию 7.2.

3.3 Веб-сервер

Выбор веб-сервера осуществим через сравнение двух популярных решений, а именно Apache [18] и nginx [19], совокупная доля которых на рынке превышает 50%.

Apache, по состоянию на 2018-й год является самым популярным и распространенным открытым веб-сервером [20]. Он надежен и гибко-конфигурируем. Сервер возможно расширить с помощью дополнительных загружаемых модулей и исполнять программы на большинстве языков программирования, не используя внешнего программного обеспечения. Apache поддерживается на большинстве операционных систем, включая Unix, Linux, Mac OS X, Windows и другие.

Веб-сервер nginx так же довольно распространен среди пользователей, его часто выбирают из-за его низкого потребления ресурсов и устойчивости, а также из-за его широких возможностей. Но nginx не способен самостоятельно обрабатывать динамический контент. Для обработки запросов к PHP, nginx должен передать запрос для исполнения внешнему процессору, дождаться генерации ответа и принять. И только после этого пользователь может получить результат.

Хотя каждый из этих веб-серверов обладает очень схожими свойствами, не стоит воспринимать их как взаимозаменяемые средства. И Apache, и nginx обладают своими преимуществами и важно понимать, какой веб-сервер выбрать в какой ситуации.

Часто можно встретить использование связки nginx + Apache. В этом случае статические данные будет обрабатывать nginx, а динамический контент — Apache. Такое взаимодействие снижает уровень нагрузки на сервер. Данное решение следует применять для обеспечения работы ресурсов с высокой посещаемостью, тогда как для сайтов с низкой посещаемостью такая связка не даст значительного результата.

В итоге, в качестве веб-сервера было принято использовать Apache, так как разрабатываемый сервис наполнен динамическим контентом и данное решение позволит оптимально обеспечить его эффективную работу.

3.4 Система управления базами данных

Symfony способен корректно работать со следующими СУБД:

- MySQL [21];
- PostgreSQL [22];
- SQLite [23].

Чтобы выбрать используемое решение, проведем сравнение.

MySQL — это самая популярная из всех крупных серверных БД. Хотя MySQL и не пытается полностью реализовать SQL-стандарты, она предлагает широкий функционал.

Преимуществами этой СУБД являются:

- простота: MySQL легко устанавливается. Существует много сторонних инструментов, включая визуальные, облегчающих начало работы с БД;
- множество функций: MySQL поддерживает большую часть функционала SQL;
- мощность и масштабируемость: MySQL может работать с действительно большими объёмами данных, и неплохо походит для масштабируемых приложений;

- скорость: пренебрежение некоторыми стандартами позволяет MySQL работать производительнее, местами срезая на поворотах.

Недостатки:

- известные ограничения: по определению, MySQL не может сделать всё, что угодно, и в ней присутствуют определённые ограничения функциональности;
- вопросы надёжности: некоторые операции реализованы менее надёжно, чем в других реляционных СУБД.

PostgreSQL — наиболее продвинутая реляционная СУБД, ориентирующаяся в первую очередь на полное соответствие стандартам и расширяемость. Она пытается полностью соответствовать SQL-стандартам ANSI/ISO.

PostgreSQL отличается от других реляционных СУБД тем, что обладает объектно-ориентированным функционалом, в том числе полной поддержкой концепта ACID (Atomicity, Consistency, Isolation, Durability). Хотя эта СУБД не так популярна, как MySQL, существует много сторонних инструментов и библиотек для облегчения работы с PostgreSQL.

Из преимуществ можно выделить:

- полную SQL-совместимость;
- расширяемость: PostgreSQL можно программно расширить за счёт хранимых процедур;
- объектно-ориентированность: PostgreSQL — не только реляционная, но и объектно-ориентированная СУБД.

Недостатки при этом следующие:

- производительность: в простых операциях чтения PostgreSQL может уступать своим соперникам;

- популярность: из-за своей сложности инструмент не очень популярен;
- хостинг: из-за вышеперечисленных факторов не все хостинг-провайдеры поддерживают данную СУБД.

SQLite — это библиотека, встраиваемая в приложение, которое её использует. Будучи файловой БД, она предоставляет отличный набор инструментов для более простой (в сравнении с серверными БД) обработки любых видов данных.

Когда приложение использует SQLite, их связь производится с помощью функциональных и прямых вызовов файлов, содержащих данные (например, баз данных SQLite), а не какого-то интерфейса, что повышает скорость и производительность операций.

Из преимуществ SQLite можно выделить:

- файловая структура: вся база данных хранится в одном файле, что облегчает перемещение;
- отлично подходит для разработки и даже тестирования: во время этапа разработки большинству требуется масштабируемое решение. SQLite, со своим богатым набором функций, может предоставить более чем достаточный функционал, при этом будучи достаточно простой для работы с одним файлом.

Из недостатков:

- отсутствие пользовательского управления: продвинутые БД предоставляют пользователям возможность управлять связями в таблицах в соответствии с привилегиями, но у SQLite такой функции нет;
- невозможность дополнительной настройки: SQLite нельзя сделать более производительной, изменив настройки.

По итогам сравнения было решено применить в разработке MySQL, т.к. этот инструмент весьма гибок в обращении и при необходимости позволяет подстроиться под потребности благодаря широкому спектру настроек и режимов работы.

3.5 Клиентский язык программирования

Интерактивность элементов, их анимация, а также плавность переходов на сайте — все это будет осуществляться с использованием языка JavaScript [24] и одной из ее библиотек jQuery [25]. В частности, с их помощью будут сделаны:

- расстановка геометок на карте;
- прокрутка хронологической ленты;
- слайдшоу пользовательских фотографий;
- всплывающие уведомления;
- валидация полей на стороне пользователя.

3.6 Язык разметки

В качестве языка гипертекстовой разметки будет использоваться HTML5 [26]. Это пятая версия HTML, в которой сделан акцент на улучшение уровня поддержки мультимедиа-технологий с одновременным сохранением обратной совместимости, удобочитаемости кода для человека и простоты анализа для парсеров.

В этой версии было добавлено множество новых тегов, некоторых из которых будут использоваться и в нашем проекте, в частности:

- `<header>` — для разметки верхней части сайта;
- `<footer>` — для разметки нижней части сайта;
- `<menu>` — для разметки меню;
- `<article>` — для разметки основного содержимого страницы;
- `<time>` — для разметки даты и времени;
- `<video>` — для разметки видеороликов.

3.7 Язык каскадных таблиц стилей

Разрабатывая сайт, мы заинтересованы сделать его не только функционально удобным, но и визуально комфортным. Задание шрифтов, оформление различных элементов — все это, как правило, не обходится без применения такого популярного языка описания внешнего вида как CSS (каскадные таблицы стилей) [27].

Однако, не будем ограничиваться одним лишь CSS и расширим возможности каскадных таблиц стилей использованием так называемого CSS-препроцессора, который позволит при написании стилей использовать свойственные языкам программирования приёмы и конструкции: переменные, вложенность, наследуемость, циклы, функции и математические операции.

Синтаксис препроцессоров похож на обычный CSS. Код, написанный на языке препроцессора, не используется прямо в браузере, а преобразуется в чистый CSS-код с помощью специальных библиотек.

Наиболее популярными препроцессорами на сегодняшний день являются:

- LESS [28];
- SASS [29];
- Stylus [30].

По большей части все они обладают полностью идентичной функциональностью, но отличающимся синтаксисом, поэтому как правило выбираются разработчиками по эстетическим соображениям.

По этой причине в нашем проекте будет использоваться LESS.

3.8 Картографический сервис

Поскольку проектом предусмотрено отображение пользовательской активности на интерактивных картах, возникает вопрос: какой же сервис использовать?

Сегодня наиболее популярными решениями являются:

- Яндекс.Карты [31];
- Google Maps [32];
- 2ГИС [33].

Для того, чтобы сделать выбор, составим сводную таблицу с наиболее значимыми критериями (табл. 3):

Таблица 3. Сравнение картографических сервисов.

Критерий	Яндекс.Карты	Google.Maps	2ГИС
Покрытие	Карта всего мира (но наиболее проработаны карты России, Украины, Белоруссии и Казахстана, а также Европы и Северной Америки)	Карта всего мира (хорошо прорисованы все крупные города)	Россия и несколько городов в 9 странах (всего около 350 городов)
Детализация карт, качество прорисовки	Хорошая или очень хорошая детализация в России, достаточная в других странах.	Приемлемый уровень детализации.	Очень хорошая детализация в городах присутствия.
Построение маршрутов	Построение нескольких вариантов маршрута на автомобиле (с учетом пробок), общественным транспортом, пешком. Расчёт предположительного времени в пути.	Построение нескольких вариантов маршрута на автомобиле (с учетом пробок), общественным транспортом, пешком, на велосипеде и даже самолетом. Расчёт предположительного времени в пути.	Построение нескольких маршрутов на автомобиле, общественном транспорте, пешком с расчётом времени на путь.

Продолжение таблицы 3

Условия использования API	Бесплатно для использования в открытых некоммерческих неигровых проектах, не предназначенных для мониторинга и диспетчеризации. Использование ключа и регистрация не обязательна.	Бесплатно для использования в открытых некоммерческих проектах, не предназначенных для мониторинга, диспетчеризации, ведения незаконной деятельности. Обязательна регистрация и получение ключа API.	Бесплатно для использования в открытых некоммерческих проектах, не направленных на построение маршрутов. Обязательна регистрация и получение ключа.
Ограничения количества запросов при бесплатном использовании API	Число запросов к сервисам геокодирования, маршрутизации и панорам Яндекса не должно превышать 25 000 в сутки.	Число загрузок карт не должно превышать превышает 25 000 в сутки.	Количество запросов к сервису ограничено предельной величиной 10 в секунду и (или) 10 000 в месяц
Средства для вывода большого количества данных	Кластеризация; Технология активных областей; Технологии ObjectManager, LoadingObjectManager, RemoteObjectManager	Кластеризция маркеров; Технология setTimeout для последовательного вывода маркеров на карту.	Кластеризация объектов

В результате, если 2ГИС перестал рассматриваться в качестве возможного решения сразу же (по причине хорошей детализации городов только России и некоторых стран СНГ), то выбор между Яндекс.Картами и Google Maps было сделать сложнее.

Ключевым фактором, повлиявшим на выбор, оказался лучший уровень покрытия иностранных государств и наличие бóльшего числа возможностей при построении маршрутов. По этим критериям сервис Google Maps показал более приемлемые результаты.

3.9 Система управления версиями

Даже несмотря на то, что в разработке проекта принимает участие только один человек, в процессе работы целесообразно использовать систему управления версиями. Это позволит вести подробные журналы истории файлов и при необходимости обращаться к их различным состояниям, а также даст возможность восстановить случайно удаленный код.

Существует два наиболее популярных решения, это системы это SVN [34] и GIT [35]. Сравним их:

Subversion (также известная как «SVN») — свободная централизованная система управления версиями, официально выпущенная в 2004 году компанией CollabNet. Ее основными преимуществами являются:

- Допускает атомарные операции;
- Операции с ветвлением кода менее затратны;
- Широкий выбор плагинов IDE;
- Не использует пиринговую модель.

Однако, есть и недостатки:

- Неудовлетворительный набор команд для работы с репозиторием;
- Сравнительно небольшая скорость.

GIT — система, созданная изначально для управления разработкой ядра Linux. В ней используется подход, который в корне отличается от CVS. На этом фоне преимуществами системы являются:

- Значительное увеличение быстродействия;
- Дешевые операции с ветками кода;
- Полная история разработки доступная оффлайн;
- Распределенная, пиринговая модель.

Недостатками, в свою очередь, являются:

- Высокий порог вхождения (обучения) для тех, кто ранее использовал SVN;
- Ограниченная поддержка Windows (по сравнению с Linux).

После сравнения было принято решение использовать GIT.

3.10 Сводная таблица средств реализации

Обобщим все раннее выбранные средства реализации в виде единой таблицы (табл. 4):

Таблица 4. Выбранные средства реализации.

Критерий	Выбранное средство реализации	Версия (если имеется)
Система управления содержимым	Symfony	3.4.9
Серверный язык программирования	PHP	7.2
Веб-сервер	Apache	2.4.33
Система управления базами данных	MySQL	5.7.21
Клиентский язык программирования	Javascript	1.7
	библиотека JQuery	3.2.1
Язык разметки	HTML	5
Язык каскадных таблиц стилей	LESS	2.6.1
Картографический сервис	Google Maps	3.32.10
Система управления версиями	GIT	2.17.0

4. Проектирование структуры программного обеспечения и базы данных

4.1 Диаграммы прецедентов

В зависимости от наличия той или иной пользовательской роли (гость/авторизованный пользователь/модератор/администратор) возможно несколько вариантов взаимодействия с сайтом. Рассмотрим каждый из них подробнее, а для наглядности построим диаграммы прецедентов.

Поскольку система ролей устроена иерархично (рис. 8), т.е. авторизованный пользователь имеет все те привилегии, которые имеет гость, а модератор — все те, которые имеет авторизованный пользователь и т.д., то будет удобно построить несколько последовательных диаграмм, а не одну общую.

Рисунок 8. Иерархия пользовательских ролей

Для начала приведем такую диаграмму для абстрактного посетителя сайта с правами, которые есть у каждой из ролей (рис. 9). Актер схематично изображен в виде человека, а прецедент – в виде эллипса.

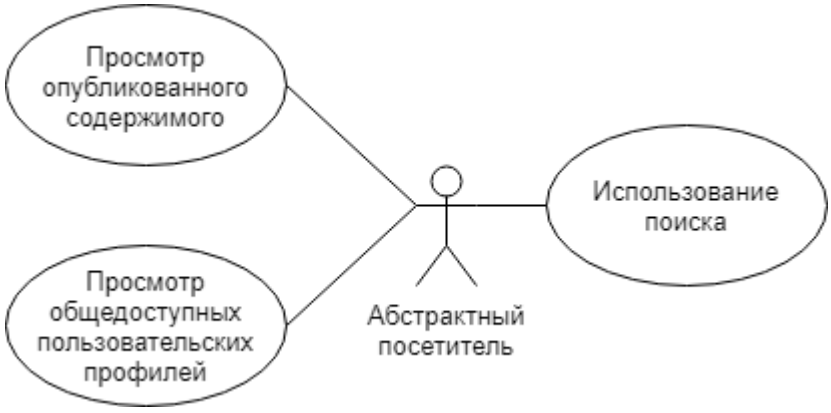

Рисунок 9. Диаграмма прецедентов роли «Абстрактный посетитель»

Теперь подобную диаграмму построим для гостей сайта (рис. 10).

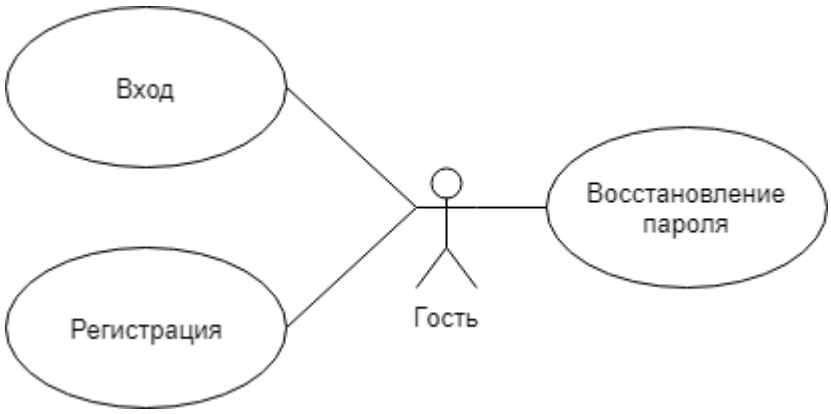

Рисунок 10. Диаграмма прецедентов роли «Гость»

Для авторизованных пользователей (рис. 11):

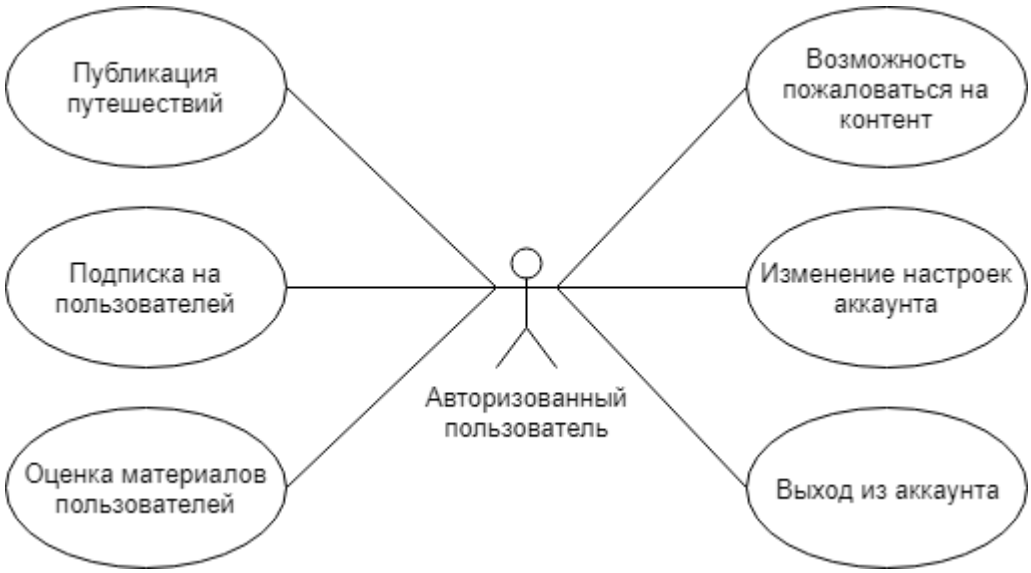

Рисунок 11. Диаграмма прецедентов роли «Авторизованный пользователь»

Для модераторов (рис. 12):

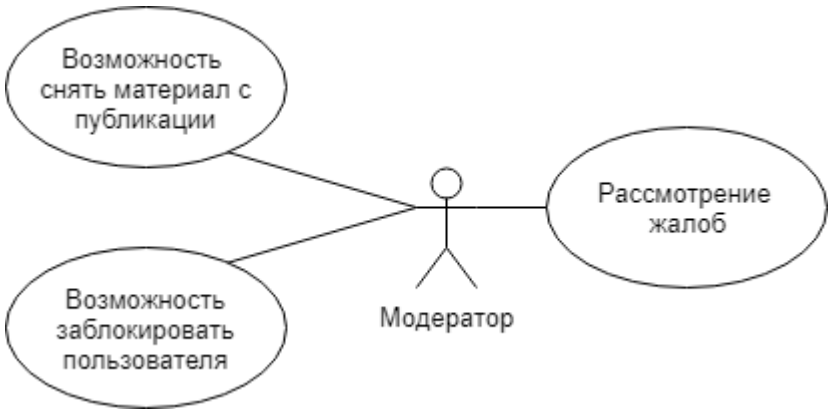

Рисунок 12. Диаграмма прецедентов роли «Модератор»

И для администраторов сайта (рис. 13):

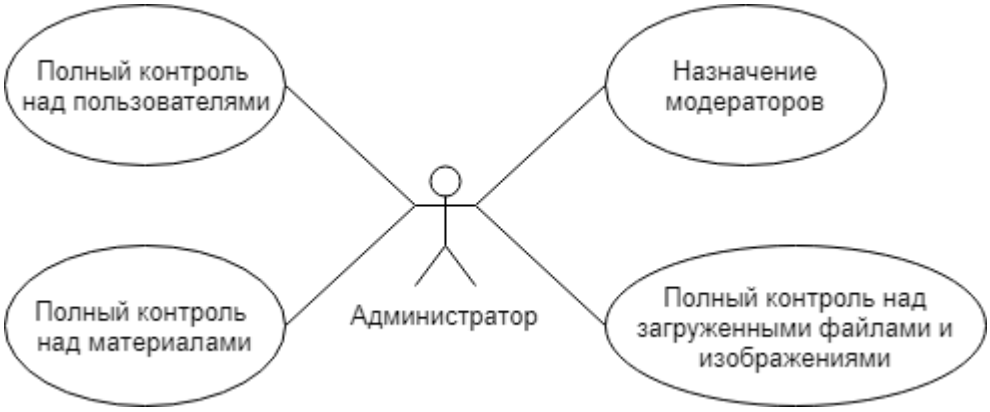

Рисунок 13. Диаграмма прецедентов роли «Администратор»

4.2 Логическая структура сайта

Система организации ссылок между страницами сайта устроена древовидно, и пользователь при заходе на главную страницу оказывается перед выбором, куда идти дальше. После перехода в нужный раздел, он подбирает необходимый подраздел и т. п.

На рис. 7 приведена логическая структура сайта для всех ролей, за исключением «модератора» и «администратора». Их функциональность недостижима с главной страницы, а содержится в отдельном разделе.

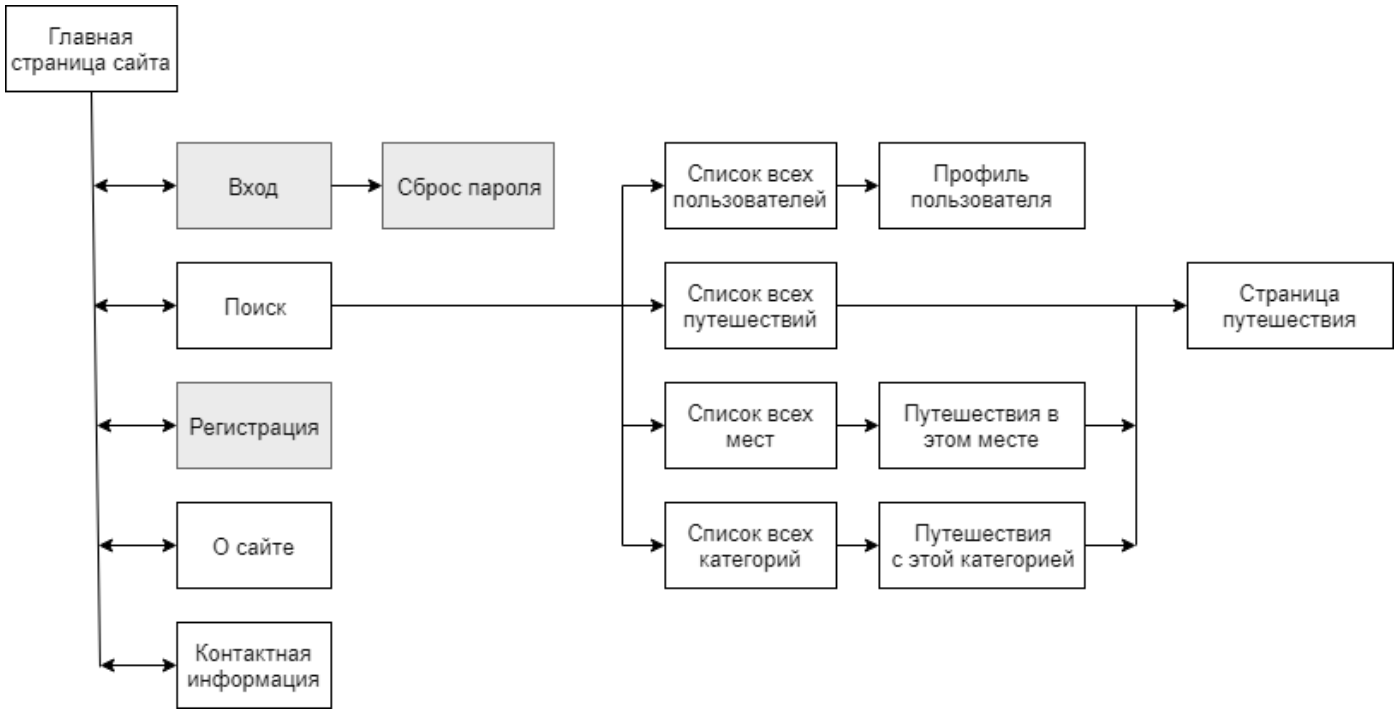

Рисунок 14. Логическая структура сайта

Серым цветом выделены страницы, которые доступные лишь гостям. Если любую из них откроет авторизованный пользователь/модератор/администратор, то будет переадресован на главную страницу сайта.

4.3 Физическая структура сайта

Используемая версия 3.4.9 фреймворка Symfony использует следующее дерево каталогов (рис. 8):

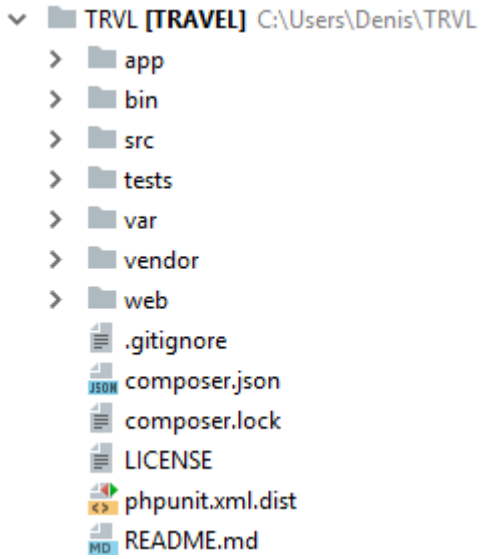

Рисунок 15. Структура каталогов Symfony 3.4.9

Корневой директорией сайта является директория /web (рис. 9). Именно в ней будут храниться все доступные через браузер файлы, включая файлы каскадных таблиц стилей, файлы скриптов, а также файлы и изображения, загруженные пользователями.

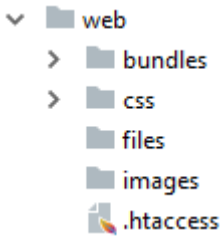

Рисунок 16. Содержимое директории /web

4.4 Проектирование базы данных

Осуществлять проектирование будем постепенно добавляя одну таблицу за другой. В процессе будем использовать следующие аббревиатуры:

- AI — auto increment. Используется как генератор уникального идентификатора для новых записей;
- PK — primary key. Первичный ключ;
- FK — foreign key. Внешний ключ.

Отметим, что непосредственное взаимодействие с базой данных будет осуществляться с использованием объектно-реляционного проектора Doctrine [36], который базируется на слое абстракции доступа к БД.

Наш сайт не может существовать без пользователей, поэтому начнем проектирование именно с этой таблицы (табл. 5):

Таблица 5. Структура таблицы Users (Пользователи).

Название	**Тип**	**Назначение**
ID	Int (PK, AI)	уникальный идентификатор
Username	Varchar (20)	псевдоним
Email	Varchar (255)	электронная почта
Password	Varchar (255)	пароль
Roles	Longtext	массив ролей пользователя
Timezone	Varchar (64)	временная зона
Locale	Varchar (8)	язык интерфейса
Enabled	Boolean	статус аккаунта (активен/отключен)
Firstname	Varchar (30)	имя
Lastname	Varchar (30)	фамилия
Gender	Varchar (1)	пол
DateOfBirth	Datetime	дата рождения
Location ID	Int (FK)	идентификатор местоположения

В таблице выше мы указали, что некоторые пользователи могут выбрать свое местоположение с помощью ссылки на соответствующую запись из другой таблицы. Объявим ее (табл. 6):

Таблица 6. Структура таблицы Locations (Местоположения).

Название	**Тип**	**Назначение**
ID	Int (PK, AI)	уникальный идентификатор
Country ID	Int (FK)	идентификатор страны
Name	Varchar (50)	название места
Type	Int (FK)	идентификатор типа места
Latitude	Varchar (50)	координаты широты
Longitude	Varchar (50)	координаты долготы

Поскольку мы ссылаемся на страну, то необходимо организовать соответствующий справочник (табл. 7).

Таблица 7. Структура таблицы Countries (Страны).

Название	**Тип**	**Назначение**
ID	Int (PK, AI)	уникальный идентификатор
Name	Varchar (50)	название

Кроме того, стоит завести справочник типов местоположений (город, остров, деревня и т.д.) поскольку хранить типы строками не очень рационально (табл. 8).

Таблица 8. Структура таблицы Location Types (Типы мест).

Название	**Тип**	**Назначение**
ID	Int (PK, AI)	уникальный идентификатор
Name	Varchar (50)	название (город, остров и др.)

Теперь займемся проектированием таблицы путешествий (табл. 9):

Таблица 9. Структура таблицы Trips (Путешествия).

Название	**Тип**	**Назначение**
ID	Int (PK, AI)	уникальный идентификатор
User ID	Int (FK)	идентификатор пользователя
Name	Varchar (50)	название поездки
Description	Varchar (255)	текстовое описание поездки

Предполагается, что пользователи будут иметь возможность указать категорию своего путешествия (семейный отдых, командировка и т.д.), поэтому необходимо завести такой справочник (табл. 10):

Таблица 10. Структура таблицы TripCategories (Категории путешествий).

Название	**Тип**	**Назначение**
ID	Int (PK, AI)	уникальный идентификатор
Name	Varchar (50)	название

Поскольку одно путешествие может относиться сразу к нескольким категориям, необходима таблица (табл. 11), которая бы устанавливала эти связи:

Таблица 11. Структура таблицы Trips_cross_Categories (Соответствие между путешествиями и категориями).

Название	**Тип**	**Назначение**
Trip ID	Int (FK)	идентификатор путешествия
Category ID	Int (FK)	идентификатор категории

Каждое путешествие будет состоять из набора дней, создадим под них необходимую таблицу (табл. 12):

Таблица 12. Структура таблицы Days (Дни путешествий).

Название	**Тип**	**Назначение**
Date	Datetime (PK)	дата
Trip ID	Int (FK)	идентификатор путешествия

В свою очередь, все заметки о путешествии будут сохранены как Моменты. Сделаем таблицу и для них (табл. 13), где уже сможем сослаться на созданные раннее День и Местоположение.

Таблица 13. Структура таблицы Moments (Моменты).

Название	**Тип**	**Назначение**
ID	Int (PK, AI)	уникальный идентификатор
Day ID	Int (FK)	идентификатор дня путешествия
Time	Datetime	время
Location ID	Int (FK)	местоположение
Text	Longtext	текст записи

Пользователи будут иметь возможность оставить комментарий к каждому из Моментов, для этого требуется организовать таблицу (табл. 14):

Таблица 14. Структура таблицы Comments (Комментарии).

Название	**Тип**	**Назначение**
ID	Int (PK, AI)	уникальный идентификатор
User ID	Int (FK)	идентификатор автора
Moment ID	Int (FK)	идентификатор момента
Datetime	Datetime	время публикации
Text	Longtext	текст
Enabled	Boolean	статус (опубликовн/не опубликован)

Также пользователи получат возможность подписаться на интересных им людей, для этого также нужна таблица (табл. 15):

Таблица 15. Структура таблицы Followers (Подписчики).

Название	**Тип**	**Назначение**
User ID	Int (FK)	идентификатор пользователя
Follower ID	Int (FK)	идентификатор его подписчика

К Путешествию, Дню или Моменту можно будет проявить симпатию, поставив лайк. Для этого нам тоже потребуется соответствующая таблица (табл. 16):

Таблица 16. Структура таблицы Likes (Лайки).

Название	**Тип**	**Назначение**
Entity ID	Int (FK)	идентификатор сущности
Entity Type	Varchar (50)	тип сущности (поездка/день/момент)
User ID	Int (FK)	идентификатор пользователя, поставившего лайк

Количество лайков к сущности рационально обновлять только при выставлении нового лайка, а не при каждом ее отображении, поэтому сделаем таблицу (табл. 17), которая будет содержать кэшированную информацию о количестве лайков:

Таблица 17. Структура таблицы LikesCache (Кэшированные лайки).

Название	**Тип**	**Назначение**
Entity ID	Int (FK)	идентификатор момента
Entity Type	Varchar (50)	тип сущности (поездка/день/момент)
Value	Int	количество лайков

Рассказывая о своем путешествии, пользователи возможно захотят поделиться с читателями какими-либо файлами, поэтому предусмотрим таблицу (табл. 18) и на этот случай:

Таблица 18. Структура таблицы Files (Файлы).

Название	**Тип**	**Назначение**
ID	Int (PK, AI)	уникальный идентификатор
Name	Varchar (255)	название
URI	Varchar (255)	путь к файлу
Entity ID	Int (FK)	идентификатор момента
Entity Type	Varchar (50)	тип сущности (поездка/день/момент)
Type	Varchar (50)	расширение файла
Size	Float	размер файла

Аналогичная ситуация и с изображениями (табл. 19):

Таблица 19. Структура таблицы Images (Изображения).

Название	**Тип**	**Назначение**
ID	Int (PK, AI)	уникальный идентификатор
Name	Varchar (255)	название
URI	Varchar (255)	путь к изображению
Entity ID	Int (FK)	идентификатор момента
Entity Type	Varchar (50)	тип сущности (поездка/день/момент)
Type	Varchar (50)	расширение изображения
Size	Float	размер изображения
Width	Int	ширина изображения
Height	Int	высота изображения

Проектирование можно считать завершенным. Еще раз перечислим все созданные таблицы (табл. 20):

Таблица 20. Структура базы данных.

Название таблицы	**Хранимые объекты**
Users	пользователи
Countries	страны
Locations	местоположения
LocationTypes	категории местоположений
Trips	путешествия
TripCategories	категории путешествий
Trips_cross_Categories	соответствие путешествий и категорий
Days	дни путешествия
Moments	моменты путешествия
Comments	комментарии к моментам
Followers	подписчики
Likes	лайки
LikesCache	кэшированные лайки
Files	файлы
Images	изображения

И изобразим их связи (рис. 10):

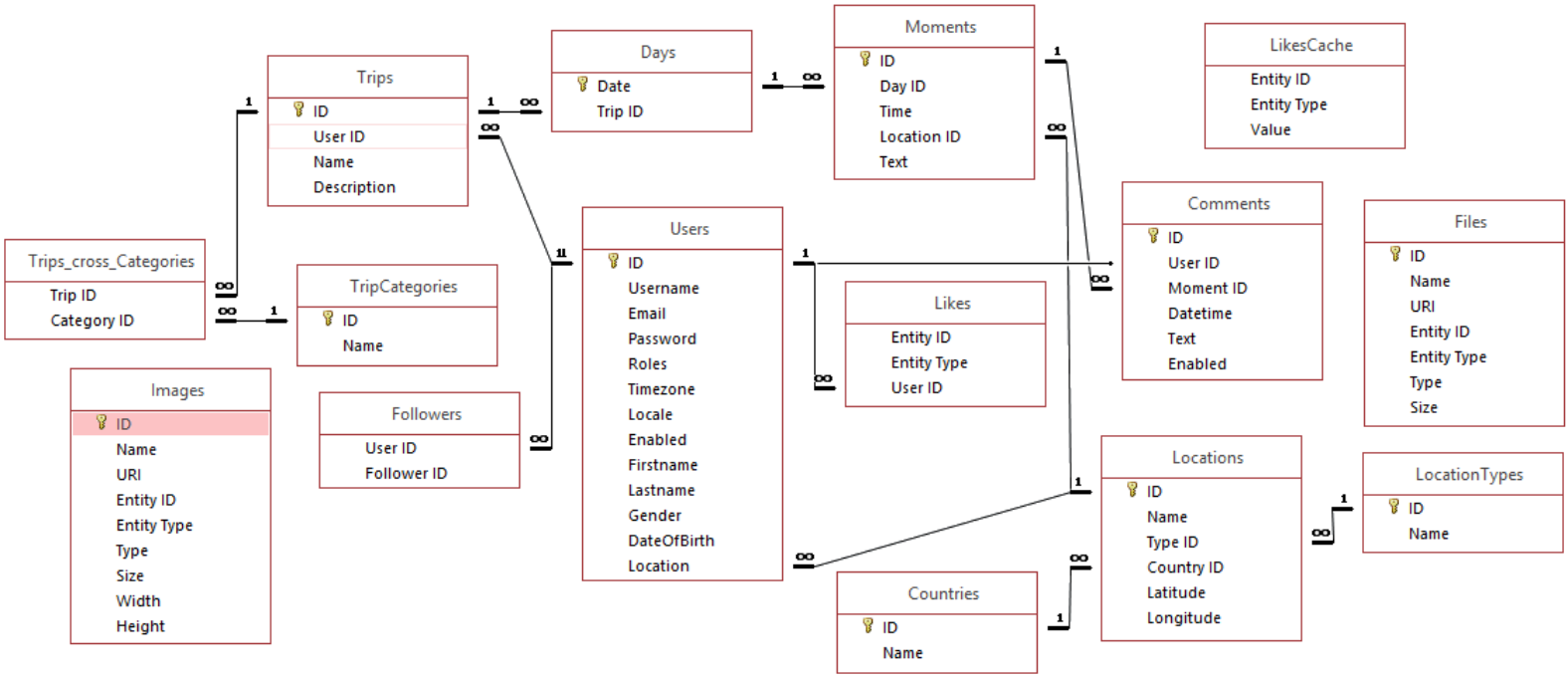

Рисунок 17. Связи между таблицами базы данных

5. Реализация и тестирование

На конкретных примерах рассмотрим, как с помощью выбранного нами фреймворка Symfony можно решать такие часто возникающие задачи, как:

- создание новой сущности;
- связывание сущностей друг с другом;
- запросы к базе данных;
- создание форм;
- создание контроллеров;
- проверка прав доступа.

5.1 Создание сущности

Сущность в Doctrine ORM — это класс, который имеет характерный для него набор свойств и может либо взаимодействовать с базой данных, для хранения информации о своих объектах, либо обходиться без неё, как, например в случае суперкласса [37].

Для примера возьмем описанную нами раннее сущность Trip (Путешествие), схема данных которой выглядит следующим образом (табл. 21):

Таблица 21. Структура таблицы Trips (Путешествия).

Название	**Тип**	**Назначение**
ID	Int (PK, AI)	уникальный идентификатор
User ID	Int (FK)	идентификатор пользователя
Name	Varchar (50)	название поездки
Description	Varchar (255)	текстовое описание поездки

Для создания такой сущности необходимо наличие класса, описывающего каждое из его свойств:

```
/**
 * УКАЗЫВАЕМ НАЗВАНИЕ ТАБЛИЦЫ В БАЗЕ ДАННЫХ
 * @ORM\Table(name="trips")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\TripRepository")
 */
class Trip
{
    /**
     * УКАЗЫВАЕМ СВОЙСТВА ПОЛЯ
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;



    /**
     * @ORM\Column(name="name", type="string", length=50)
     */
    private $name;

    /**
     * УСТАНАВЛИВАЕМ СВЯЗЬ С СУЩНОСТЬЮ USER
     * @ManyToOne(targetEntity="User")
     * @JoinColumn(name="user_id", referencedColumnName="id")
     */
    private $user;

    /**
     * @ORM\Column(name="name", type="string", length=255)
     */
    private $description;
```

Необходимо также объявить сеттеры и геттеры для сохранения и получения данных соответственно:

```
    /**
     * @return int
     */
    public function getId()
    {
        return $this->id;
    }

    /**
     * @return string
     */
    public function getName()
    {
        return $this->name;
    }
```

```
 /**
     * @param string $name
     */
    public function setName($name)
    {
        $this->name = $name;
    }

    /**
     * @return User
     */
    public function getUser()
    {
        return $this->user;
    }
}
```

Отдельно стоит отметить строку

```
@ORM\Entity(repositoryClass="AppBundle\Repository\TripRepository")
```

Она необходима для того, чтобы установить расположение репозитория этого класса. В разделе 5.3 нам это пригодится.

5.2 Виды связей сущностей

Существует множество способов, как обозначить связи между сущностями. Среди них:

- «один к одному» (односторонняя/двусторонняя) — в этом случае объект одной сущности сможет ссылаться не более чем на один объект другой или своей сущности;
- «многие к одному» (односторонняя) — в этом случае объект одной сущности сможет ссылаться на множество объектов другой или своей сущности;
- «многие ко многим» (одностороняя/двусторонняя) — в этом случае объекты одной сущности смогут ссылаться на множество объектов другой или своей сущности.

В разделе 5.1 уже было показано, как создается связь «многие к одному», когда один пользователь может иметь множество путешествий. Рассмотрим и остальные виды связей.

Одностороннюю связь «один к одному» можно описать так:

```
/** @Entity */
class Country
{
// ...
/**
* @OneToOne(targetEntity="Capital")
* @JoinColumn(name="capital_id", referencedColumnName="id")
*/
private $capital;
// ...
}

/** @Entity */
class Capital
{
// ...
}
```

В этом случае у нас одна страна (Country) будет иметь ровно одну столицу (Capital), при этом зная страну, мы сможем узнать и ее столицу, а вот в обратную сторону это сделать не удастся.

Двусторонняя связь «один к одному» описывается следующим образом:

```
/** @Entity */
class Country
{
    // ...
    /**
     * @OneToOne(targetEntity="Capital", mappedBy="country")
     */
    private $capital;
    // ...
}

/** @Entity */
class Capital
{
    // ...
    /**
     * @OneToOne(targetEntity="Capital", inversedBy="country")
     * @JoinColumn(name="country_id", referencedColumnName="id")
     */
    private $country;
    // ...
}
```

Суть осталась прежней: у каждой страны не может быть более одной столицы, однако теперь, зная столицу, мы сможем узнать ее страну.

Односторонняя связь «многие ко многим» описывается следующим образом:

```
/** @Entity */
class User
{
    // ...
    /**
     * @ManyToMany(targetEntity="Group")
     * @JoinTable(name="users_groups",
     *      joinColumns={@JoinColumn(name="user_id",
referencedColumnName="id")},
     *      inverseJoinColumns={@JoinColumn(name="group_id",
referencedColumnName="id")}
     *      )
     */
    private $groups;
    // ...
}

/** @Entity */
class Group
{
    // ...
}
```

В этом случае один пользователь может состоять в множестве групп, и об этих группах можно узнать, только имея пользователя. Если же мы будем иметь только информацию о группе, то не сможем узнать кто в ней состоит.

Чтобы исправить это, необходимо организовать двустороннюю связь «многие ко многим», которая описывается так:

```
/** @Entity */
class User
{
    // ...
    /**
     * @ManyToMany(targetEntity="Group", inversedBy="users")
     * @JoinTable(name="users_groups")
     */
    private $groups;

    public function __construct() {
        $this->groups = new \Doctrine\Common\Collections\ArrayCollection();
    }
}
```

```
/** @Entity */
class Group
{
    // ...
    /**
     * @ManyToMany(targetEntity="User", mappedBy="groups")
     */
    private $users;

    public function __construct() {
        $this->users = new \Doctrine\Common\Collections\ArrayCollection();
    }
}
```

5.3 Запросы к базе данных

Ниже приведен пример, как зная пользователя, можно получить информацию обо всех его путешествиях.

```
class TripsRepository extends \Doctrine\ORM\EntityRepository
{
    public function getTripsForUser(User $user)
    {
        $qb = $this->createQueryBuilder('t')
            ->select('t')
            ->where('t.user_id = :user_id')
            ->setParameter('user_id', $user->getId());

        return $qb->getQuery()
            ->getResult();
    }
}
```

Этот запрос вернет массив объектов Trip (Путешествие). Обратим внимание на то, что функция расположена в репозитории, описанном в разделе 5.1.

5.4 Создание формы отправки данных

Форму создания нового путешествия в общем случае можно описать так:

```
class TripType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('user')
            ->add('name')
            ->add('description')
    }
}
```

В ней мы отображаем созданный нами в разделе 5.1 набор полей сущности Trip (Путешествие).

5.5 Создание контроллера

Чтобы сохранить данные из формы, необходимо реализовать отдельный контроллер. В общем виде он будет выглядеть следующим образом:

```
public function createAction(Request $request)
{
    $trip = new Trip();
    $form = $this->createForm(TripType::class, $trip);
    $form->handleRequest($request);
    if ($form->isValid()) {
        $brewery = $form->getData();
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($trip);
        $entityManager->flush();
    }

    return $this->render(
        'AppBundle:Trip:add.html.twig',
        [
            'form' => $form->createView(),
        ]
    );
}
```

Этот контроллер сначала вызовет страницу с созданной раннее формой, а затем, если отправленные через форму данные успешно пройдут валидацию, создаст новый объект сущности Путешествия.

5.6 Проверка прав доступа

Как уже было отмечено в разделе 4.1, каждая пользовательская роль характеризуется индивидуальным набор разрешенных действий. Для разграничения доступа будем использовать 2 основных подхода: проверка наличия пользовательской роли и проверка наличия требуемых атрибутов.

В большинстве случаев роль пользователя является достаточным критерием для предоставления или запрета прав на совершение того или иного действия. Так, например, зная, что посетитель не имеет роли «гость», можно сразу же выполнить переадресацию со страницы авторизации, которая в данном случае будет не актуальна.

В этом случае удобно использовать проверку формата «адрес — роль», которая описывается в файле /app/config/security.yml:

```
access_control:
    - { path: ^/api/v1/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
```

Но существуют случаи, когда сведений о наличии или отсутствии роли недостаточно. Например, когда необходимо разрешить редактирование материала блога только ее автору. Для этого сначала необходимо создать класс избирателя (Voters), с указанием в каким именно случаях он должен возвращать разрешение на доступ:

```
if ($user->getId() === $moment->->getUser()->getId()) {
    return VoterInterface::ACCESS_GRANTED;
}
```

А затем использовать этот класс в требуемом контроллере:

```
    /**
     * @Route("/moment/{id}/edit")
     * @Security("is_granted('edit', moment)")
     */
    public function momentEdit(Brewery $brewery)
    {
            ...
    }
```

5.7 Текущее состояние сайта

Разрабатываемый веб-сайт размещен в сети Интернет и доступен по адресу http://travel.azarov.de [38], однако в настоящий момент он работает в режиме ограниченной функциональности и поддерживает лишь ряд возможностей, описанных в данной работе.

Заключение

Выпускная квалификационная работа на тему: «Разработка сервиса ведения блогов о путешествиях» выполнена в соответствии с поставленным заданием. В процессе выполнения был спроектирован, а впоследствии и реализован сервис, представляющий собой веб-сайт и программный интерфейс приложения (API), который в дальнейшем может быть использован для обмена данными с мобильными приложениями и иным программным обеспечением.

Работа состояла из следующих этапов:

1. анализа предметной области: в частности, были рассмотрены существующие решения, обозначена целевая аудитория;
2. написания технического задание, наиболее полно описывающего желаемый результат;
3. сравнения и последующего выбора средств реализации, которые наилучшим образом удовлетворяли бы функциональным потребностям сервиса.
4. непосредственно реализации сервиса и его публикации в сеть Интернет.

В настоящий момент сайт размещен в сети Интернет и общедоступен, однако он работает в режиме ограниченной функциональности и поддерживает лишь ряд возможностей, описанных в данной работе.

Среди направления для дальнейшего развития сервиса можно обозначить:

- разработку клиентских версий приложений для мобильных устройств;
- интеграцию с сервисами близкой тематики (бронирование гостиниц, покупка авиабилетов и т.д.);
- дальнейшая локализация: перевод сервиса на наиболее востребованные языки.

Список использованных источников

1. 2017 International Tourism Results: the highest in seven years [Электронный ресурс]. — Режим доступа: http://media.unwto.org/press-release/2018-01-15/2017-international-tourism-results-highest-seven-years/ — (Дата обращения: 01.04.2018)
2. Your Travel Blog centered on an Interactive Map — TravelMap: [Электронный ресурс]. — Режим доступа: https://travelmap.net/ — (Дата обращения: 03.04.2018)
3. Tripline [Электронный ресурс]. — Режим доступа: https://www.tripline.net/ — (Дата обращения: 03.04.2018)
4. Tripblan: The Travel Guides and Travel Journals Sharing App [Электронный ресурс]. — Режим доступа: https://tripblan.com/ — (Дата обращения: 03.04.2018)
5. Приложения в Google Play – Tripline [Электронный ресурс]. — Режим доступа: https://play.google.com/store/apps/details?id=net.tripline&hl=ru — (Дата обращения: 03.04.2018)
6. App Store: Tripline — iTunes — Apple [Электронный ресурс]. — Режим доступа: https://itunes.apple.com/ru/app/tripline/id417133912?mt=8 — (Дата обращения: 03.04.2018)
7. Ostrovok.ru - бронирование отелей [Электронный ресурс]. — Режим доступа: https://ostrovok.ru — (Дата обращения: 03.04.2018)
8. Тонкости туризма — энциклопедия курортов, описания отелей [Электронный ресурс]. — Режим доступа: https://tonkosti.ru/ — (Дата обращения: 03.04.2018)
9. LiveInternet: Рейтинг сайтов [Электронный ресурс]. — Режим доступа: https://www.liveinternet.ru/rating/ru/ — (Дата обращения: 03.04.2018)
10. Статистика сайта Тонкости туризма [Электронный ресурс]. — Режим доступа: https://www.liveinternet.ru/stat/tonkosti.ru/index.html — (Дата обращения: 03.04.2018)

11. Программный интерфейс приложения (API) [Электронный ресурс]. — Режим доступа: http://cccp.ifmo.ru/scorm/___api.html — (Дата обращения: 03.04.2018)
12. SQL-инъекции [Электронный ресурс]. — Режим доступа: http://php.net/manual/ru/security.database.sql-injection.php — (Дата обращения: 05.04.2018)
13. Cross-Site Request Forgeries [Электронный ресурс]. — Режим доступа: http://shiflett.org/articles/cross-site-request-forgeries — (Дата обращения: 05.04.2018)
14. Top Ten Internet Languages - World Internet Statistics [Электронный ресурс]. — Режим доступа: https://www.internetworldstats.com/stats7.htm — (Дата обращения: 06.04.2018)
15. Laravel - The PHP Framework For Web Artisans [Электронный ресурс]. — Режим доступа: https://laravel.com/ — (Дата обращения: 07.04.2018)
16. Symfony, High Performance PHP Framework for Web Development [Электронный ресурс]. — Режим доступа: https://symfony.com/ — (Дата обращения: 07.04.2018)
17. Drupal - Open Source CMS | Drupal.org [Электронный ресурс]. — Режим доступа: https://www.drupal.org/ — (Дата обращения: 07.04.2018)
18. The Apache HTTP Server Project [Электронный ресурс]. — Режим доступа: https://httpd.apache.org/ — (Дата обращения: 08.04.2018)
19. nginx [Электронный ресурс]. — Режим доступа: https://nginx.ru/ru/ — (Дата обращения: 08.04.2018)
20. December 2017 Web Server Survey [Электронный ресурс]. — Режим доступа: https://news.netcraft.com/archives/2017/12/26/december-2017-web-server-survey.html — Режим доступа: https://nginx.ru/ru/ — (Дата обращения: 08.04.2018)
21. MySQL [Электронный ресурс]. — Режим доступа: https://www.mysql.com/ — (Дата обращения: 08.04.2018)

22. PostgreSQL: The world's most advanced open source database [Электронный ресурс]. — Режим доступа: https://www.postgresql.org/ — (Дата обращения: 08.04.2018)
23. SQLite Home Page [Электронный ресурс]. — Режим доступа: https://www.sqlite.org/ — (Дата обращения: 08.04.2018)
24. JavaScript Tutorial [Электронный ресурс]. — Режим доступа: https://www.w3schools.com/js/ — (Дата обращения: 09.04.2018)
25. jQuery [Электронный ресурс]. — Режим доступа: https://jquery.com/ — (Дата обращения: -9.04.2018)
26. HTML5 | htmlbook.ru [Электронный ресурс]. — Режим доступа: http://htmlbook.ru/html5 — (Дата обращения: 09.04.2018)
27. Справочник CSS | htmlbook.ru [Электронный ресурс]. — Режим доступа: http://htmlbook.ru/css — (Дата обращения: 09.04.2018)
28. Less.js: Getting started [Электронный ресурс]. — Режим доступа: http://lesscss.org/ — (Дата обращения: 09.04.2018)
29. Sass: Syntactically Awesome Style Sheets [Электронный ресурс]. — Режим доступа: https://sass-lang.com/ — (Дата обращения: 09.04.2018)
30. Stylus [Электронный ресурс]. — Режим доступа: http://stylus-lang.com/ — (Дата обращения: 09.04.2018)
31. Яндекс.Карты [Электронный ресурс]. — Режим доступа: https://yandex.ru/maps — (Дата обращения: 11.04.2018)
32. Google Maps [Электронный ресурс]. — Режим доступа: https://www.google.ru/maps — (Дата обращения: 11.04.2018)
33. 2ГИС [Электронный ресурс]. — Режим доступа: https://2gis.ru/ — (Дата обращения: 11.04.2018)
34. Apache Subversion [Электронный ресурс]. — Режим доступа: https://subversion.apache.org/ — (Дата обращения: 12.04.2018)
35. Git [Электронный ресурс]. — Режим доступа: https://git-scm.com/ — (Дата обращения: 12.04.2018)

36. Object Relational Mapper - Doctrine - PHP Database Tools [Электронный ресурс]. — Режим доступа: https://www.doctrine-project.org/projects/orm.html — (Дата обращения: 01.05.2018)
37. Inheritance Mapping - Object Relational Mapper (ORM) - Doctrine [Электронный ресурс]. — Режим доступа: https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/inheritance-mapping.html — (Дата обращения: 05.05.2018)
38. TRVL [Электронный ресурс]. — Режим доступа: http://travel.azarov.de/ — (Дата обращения: 10.06.2018)