Оглавление

## Постановка задачи 2

## Сценарий работы приложения 3

## Разработка клиентской части 9

## Разработка серверной части 10

## Яндекс.Карты 10

## Оптимизация скорости загрузки и работы приложения 11

## Система управления версиями 12

## Сборка проекта 13

## Тестирование 14

## Структура API для взаимодействия с базой данных 15

## Терминология 19

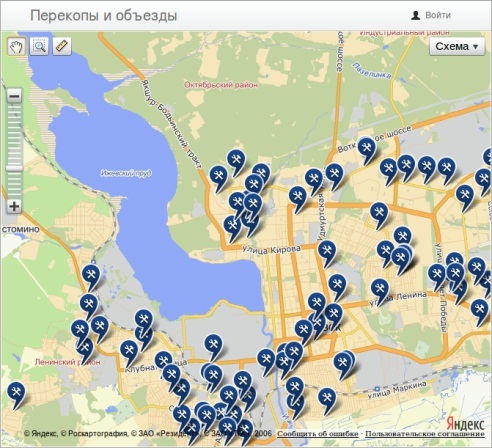
Постановка задачи

Данное веб-приложение предназначено для отображения на веб-странице карты перекопов и объездов по г. Ижевску. На карте всегда должны быть актуальные данные. Строго обговоренный круг лиц должны иметь возможность авторизоваться на данном ресурсе, после чего у них должна появиться возможность манипулирования объектами на карте: перемещение, удаление и редактирование информации существующих объектов, создание новых объектов.

Приложение разделено на клиентскую и серверную части. Клиентская часть является веб-страницей и отображается в браузере клиента, а серверная часть реализована на языке php и является промежуточным звеном между клиентской частью и базой данных FireBird. Серверная часть представляет собой готовый интерфейс программирования приложений (API), который дает определенный набор HTTP-запросов, а также определение структуры HTTP-ответов, для выражения которых используется JSON формат.

Также, необходимо разработать функционал для экспортирования актуальной информации из базы данных в XML-формат (YMapsML) для интеграции с ресурсом www.izh.ru, а для проверки выгружаемоего слоя для Яндекс.Карт, нужно создать тестовую страницу, где он будет отображаться.

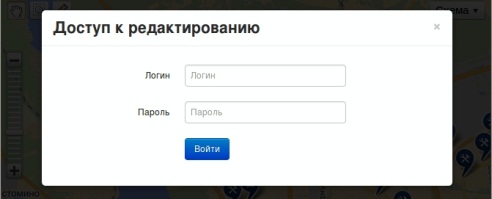
Сценарий работы приложения

**

*Рис. 1. Интерфейс приложения*

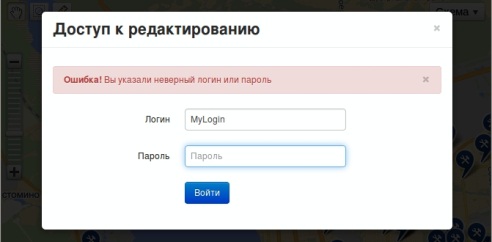
При открытии пользователем данной веб-страницы, происходит проверка на наличие в браузере пользователя cookie-файлов. Это необходимо, чтобы пользователю не нужно было бы вводить логин и пароль при каждом посещении ресурса. Если в cookies есть данные об авторизации пользователя, то к серверу делается AJAX-запрос и просходит попытка авторизации пользователя по этим данным. Если авторизация проходит успешно, то от сервера возвращается положительный ответ и данныими пользователя, которые записываются в глобальную переменную window.user, и кнопка "вход" на верхней панели страницы меняется на кнопку "выход".

Если пользователь не авторизован, то кнопка "вход" на верхней панели активна, а кнопка "выход" скрыта. При нажатии на кнопку "вход", на экране появлятся всплывающее окно с формой авторизации. Форма авторизации требует ввода логина и пароля.



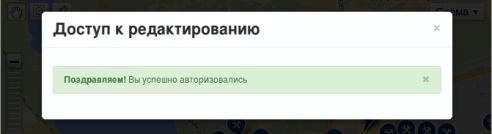
*Рис. 2. Форма авторизации пользователя*

При вводе данных и субмите формы делаются AJAX-запросы к серверу, где происходит валидация и проверка этих данных. Если пользователь вводит эти данные неправильно, то ему показывается сообщение об ошибке.



*Рис. 3. Сообщение при ошибке авторизации*

Если пользователь вводит данные правильно, то ему показывается сообщение об успехе операции и форма авторизации закрывается.



*Рис. 4. Сообщение при успешной авторизации*

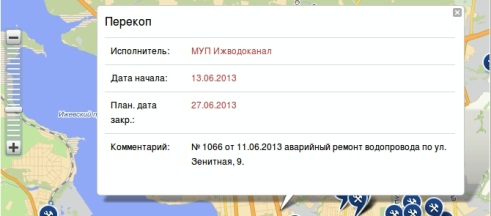
Кнопка "вход" соответственно меняется на кнопку "выход".



*Рис. 5. Интерфейс верхней панели после авторизации*

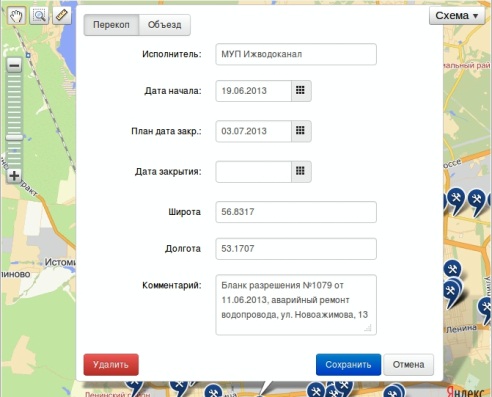
Основную часть интерфейса занимает карта, и почти весь функционал приложения так или иначе разработан для работы с объектами на этой карте. При открытии веб-страницы, сразу же загружается карта г.Ижевска. Параллельно с этим процессом, делается AJAX-запрос к серверу для получения всех объектов которые необходимо отобразить на карте. Для увеличения скорости данной операции и уменьшения трафика, в качестве ответа от сервера возвращаются только данные в JSON формате, и только те данные, которые необходимы для отображения данных объектов (в ответе нет детализации по объектам).

При нажатии левой кнопкой мыши на любую из меток, открывается балун и к серверу делается AJAX-запрос на получение детализации по данному объекту. Ответ спомощью шаблонизатора рендерится в HTML формат и отображается внутри открывшегося балуна.



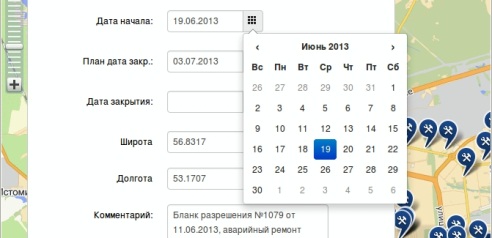
*Рис. 6. Отображение детализации объекта внутри балуна*

Если пользовать авторизован и данный объект создан им самим, то появляется возможность редактировать эти данные.



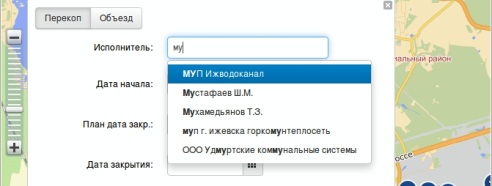
*Рис. 7. Интерфейс для редактирования данных*

При этом, при редактировании полей с датами, открывается календарь, чтобы пользователю было удобней сориентироваться и выбрать нужную дату в нужном формате.



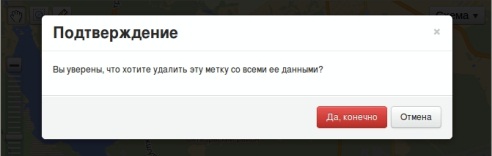
*Рис. 8. Календарь, который помагает выбрать нужную дату*

При редактировании поля "исполнитель", появляются подсказки, которые помогают пользователю выбрать нужного исполнителя.



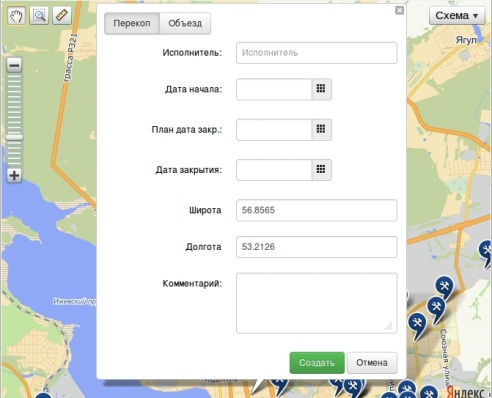
*Рис. 9. Подсказки при заполнении поля "исполнитель"*

Внизу формы есть кнопки "сохранить" и "удалить". При сохранении, данные формы отправляют на сервер, где они проходят валидацию и записываются в базу данных. При удалении метки, должно открыться окно с подтверждением данной операции, и если пользователь ответит положительно, то на сервер посылается команда для удаления данной метки из базы данных.



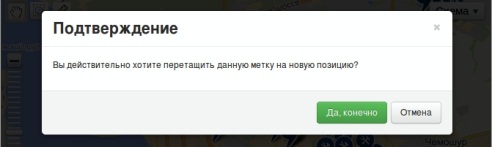
*Рис. 10. Окно с подтверждением при удалении метки*

Если пользователь хочет создать новую метку, то ему достаточно кликнуть в любую свободную часть карты и откроется окно для заполнения данными. Причем, в полях "широта" и "долгота" запишутся координаты точки, куда был произведен данный клик. При нажатии на кнопку "создать", на сервер отправятся данные для записи нового объекта в базе данных, а на карте в той точке, куда первоначально кликнул пользователь сразу же появится новая метка. Причем иконка метки будет зависеть от типа объекта, который выбрал пользователь.



*Рис. 11. Создание новой метки*

Авторизованный пользователь имеет возможность менять географические координаты объектов, которые он создал, путем их перетаскивания по карте. Для этого, ему нужно навести курсор мыши на нужную метку, зажать левую кнопку мыши, затем передвинуть метку на карте на новую позицию и отпустить левую кнопку мыши. При этом, чтобы пользователь не мог сделать это случайно, по окончании операции, на экране отображается окно с предложением подтвердить данное дейсвие, после чего на сервер отправляется запрос на изменение географических координат данной метки. Если пользователь не подтвердил данное действие, то метка возвращается на прежнюю позицию.



*Рис. 12. Окно с подтверждением при перемещении метки*

Разработка клиентской части

Клиентская сторона представляет из себя одностраничный сайт, т.е. все действия происходят на одной странице, перезагрузки страницы при каких либо действиях не происходит - связь с сервером осуществляется спомощью ajax-запросов.

Вся логика приложения написана на JavaScript'e. Для удобства работы с DOM элементами, используется JavaScript библиотека jQuery. Также она помогает проще обращаться к атрибутам и содержимому элементов DOM, и манипулировать ими. Также библиотека jQuery предоставляет удобный API для работы с AJAX. Для удобства написания JavaScript используется CoffeeScript, который компилируется в JavaScript на этапе разработки. В среднем для выполнения одинаковых действий на CoffeeScript требуется в 2 раза меньше строк, чем JavaScript. JavaScript код, который скомпилируется из CoffeeScript полностью проходит проверку JavaScript Lint.

Вся HTML разметка выполнена в виде Jade-шаблонов и также компилируется на этапе разработки. В качестве шаблонизатора на клиентской стороне использутеся EJS (Embedded JavaScript). Шаблонизатор на стороне браузера необходим для того, чтобы обрабатывать JSON ответы от сервера и осуществлять рендеринг данных в HTML формат (например, при открытии балуна метки сначала делается запрос на получение данных от сервера, а затем осуществляется рендеринг этих данных и их отображение внутри открывшегося балуна).

Дизайн клиентской части сделан спомощью css-фреймворка bootstrap. Это свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML и CSS шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейсов, включая JavaScript расширения.

Разработка серверной части

Серверная часть приложения реализована на минималистичном php-фреймворке sn-system от компании-разработчика Стандарт-Н. Логика работы этой части сведена к получению HTTP-запросов от клиентской стороны, их обработке и валидации, осуществлению SQL-запросов к базе данных и возвращению ответа в JSON-формате. Данный интерфейс реализован как API. Также реализована прозрачная поддержка JSONP-запросов для будующей интеграции данного приложения с другими проектами и для возможности использования функционала и данных этого приложения на сторонних веб-ресурсах.

Яндекс.Карты

В проекте используется API Yandex Maps 2.0 для отображения карты на странице, отображения на ней объектов и манипуляции данными объектами. Яндекс.Карты — это удобный инструмент, который поможет решить множество задач: от создания простой схемы проезда до разработки сложных геоинформационных сервисов.

Яндекс.Карты — это поисково-информационный сервис, предоставляющий пользователям карты крупных городов России, Украины, Белоруссии, Казахстана и других стран СНГ, актуальные данные о пробках, схемы метро и другие возможности. Пользователи имеют возможность размещать карты Яндекса или собственные карты на своих веб-страницах. Для этого существует бесплатный инструментарий — API Яндекс.Карт и соответствующая документация.

API Яндекс.Карт — это программный интерфейс, с помощью которого можно установить карту и весь необходимый для работы с ней инструментарий на сайт. На его основе работают Яндекс.Недвижимость, Яндекс.Фотки, а также более ста тысяч других сайтов. API предоставляет доступ ко всему содержимому Яндекс.Карт — это сотни подробных схем городов, карта мира, спутниковые снимки планеты и Народная карта. API Яндекс.Карт позволяет манипулировать любыми картами — то есть изображениями местности или любых объектов, каждой точке которых соответствуют определённые двумерные координаты.

При работе с картами часто используется такой приём как размещение на изображении местности различных графических объектов. Это могут быть метки, всплывающие подсказки, линии, многоугольники и другие элементы. API Яндекс.Карт содержит набор программных компонентов, реализующих отображение наиболее часто используемых графических объектов. Причём объекты могут размещаться как на интерактивных картах с помощью JavaScript, так и на статических изображениях местности с помощью обычных HTTP-запросов.

Интерактивность электронных карт подразумевает их реакцию на действия, совершаемые пользователем, например, на нажатие кнопки изменения масштаба. Когда пользователь совершает какие-либо действия в активной области карты, генерируются события. API Яндекс.Карт позволяет отслеживать эти события и программировать реакцию на их возникновение. События возникают не только как реакция на действия пользователя, но и во время программного обращения к электронной карте посредством API, например, при размещении на ней объектов или их удалении. Реализованный в API Яндекс.Карт механизм обработки событий является хорошей базой для создания интерактивных картографических веб-приложений.

Оптимизация скорости загрузки и работы приложения

#### Клиентская оптимизация

В проекте проведен целый ряд мер, направленный на увеличение скорости загрузки приложения в браузере клиента: - css файлы и JavaScript загружаются на страницу только через теги <link> и <script> с указанием атрибута src - весь JavaScript код подключается внизу тега <body> - css стили и js файлы сжимаются и минифицируются - объединение css в один - объединение js файлов - изображения объединены в спрайты и оптимизированы - сторонний контент загружается асинхронно - кэширование шаблонов

#### Серверная оптимизация

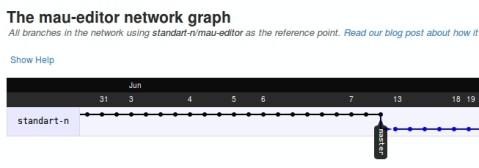
* GZIP сжатие
* кэширование статики

Система управления версиями

Система управления версиями — это программное обеспечение для облегчения работы с изменяющейся информацией. Она позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (Software Configuration Management Tools).

Для данного проекта используется Git - это наиболее популярная на сегодняшний день система управления версиями. Система спроектирована как набор программ, специально разработанных с учётом их использования в скриптах. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы. Например, Cogito является именно таким примером фронтенда к репозиториям Git, а StGit использует Git для управления коллекцией патчей. Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone, Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой. Удалённый доступ к репозиториям Git обеспечивается git-daemon, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использовать существующие конфигурации сетевых фильтров.



*Рис. 13. График истории изменений проекта*

Сборка проекта

Для автоматизации процесса преобразования файлов из одной формы в другую и компоновки других файлов, в том числе компиляции jade-разметки и СoffeeScript скриптов используется программа make, ситема сборки Grunt.js, серверная платформа Node.js и пакетный менеджер NPM.

* NPM — это пакетный менеджер node.js. С его помощью можно управлять модулями и зависимостями.
* Grunt — это инструмент для сборки javascript проектов из командной строки с использованием задач.
* Node или Node.js — серверная платформа, использующая язык программирования JavaScript, основанная на движке V8. Предназначена для создания масштабируемых распределённых сетевых приложений, таких как веб-сервер. В отличие от большинства программ JavaScript, этот каркас исполняется не в браузере клиента, а на стороне сервера.
* Make — утилита, автоматизирующая процесс преобразования файлов из одной формы в другую. Чаще всего это компиляция исходного кода в объектные файлы и последующая компоновка в исполняемые файлы или библиотеки. Утилита использует специальные make-файлы, в которых указаны зависимости файлов друг от друга и правила для их удовлетворения. На основе информации о времени последнего изменения каждого файла make определяет и запускает необходимые программы.

Для сборки проекта требуются следующие npm-пакеты:

"grunt-cli": "~0.1.9"

"grunt": "~0.4.1"

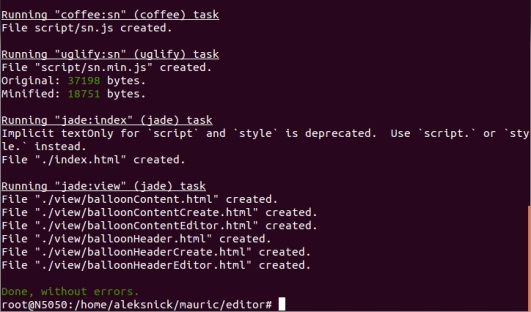
"grunt-contrib-uglify": "~0.2.2"

"grunt-contrib-coffee": "~0.7.0"

"grunt-contrib-concat": "~0.3.0"

"grunt-contrib-jade": "~0.7.0"

"grunt-recess": "~0.3.3"

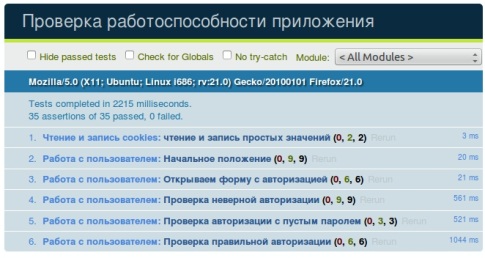


*Рис. 14. Процесс сборки проекта*

Тестирование

Тест — это процедура, которая позволяет либо подтвердить, либо опровергнуть работоспособность кода. Когда программист проверяет работоспособность разработанного им кода, он выполняет тестирование вручную. В данном контексте тест состоит из двух этапов: стимулирование кода и проверка результатов его работы. Автоматический тест выполняется иначе: вместо программиста стимулированием кода и проверкой результатов занимается компьютер, который отображает на экране результат выполнения теста: код работоспособен или код неработоспособен. При этом происходит «инверсия ответственности»: от логики тестов и их качества зависит, будет ли код соответствовать техническому заданию. Методика разработки через тестирование заключается главным образом именно в организации автоматических тестов и выработке определённых навыков тестирован

В данном проекте используется библиотека QUnit. QUnit — это библиотека от разработчиков jQuery, позволяющая писать unit-тесты для кода на JavaScript. В проекте тестируется интерфейс приложения и авторизация пользователя.



*Рис. 15. BDD тестирование проекта*

Структура API для взаимодействия с базой данных.

#### Запросы для работы с объектами, которые отображаются на карте

* **Получение всех точек**
  + Параметры запроса:
    - action: getPoints
  + Ответ: массив данных точек со следующими полями:
    - UUID, уникальный ID точки
    - POINT, географические координаты точки
    - USER\_ID, ID пользователя, который создал данную точку
    - VID\_ID, тип точки
    - SVID, название типа точки
    - PLAN\_PERIOD\_END, планируемая дата окончания
    - STREET, улица, на которой расположен данный объект
* **Получение данных для определенной точки**
  + Параметры запроса:
    - action: getBalloonContent
    - uuid: уникальный ID точки
    - userid: ID учетной записи (*опционально*)
    - login: логин (*опционально*)
    - hash: хэш пароля (*опционально*)
  + Ответ: массив данных по данному объекту
* **Запрос на изменение данных метки**
  + Параметры запроса:
    - action: saveMark
    - uuid: уникальный ID точки
    - agent: исполнитель
    - info: комментарий
    - date1: дата начала
    - date2: планируемая дата окончания
    - date3: дата окончания (*опционально*)
    - lat: широта
    - lon: долгота
    - vid: тип объекта (0 - перекоп, 1 - объезд)
    - userid: ID учетной записи
    - login: логин
    - hash: хэш пароля
  + Ответ: boolean
* **Запрос на создание новой метки**
  + Параметры запроса:
    - action: createMark
    - agent: исполнитель
    - info: комментарий
    - date1: дата начала
    - date2: планируемая дата окончания
    - date3: дата окончания (*опционально*)
    - lat: широта
    - lon: долгота
    - vid: тип объекта (0 - перекоп, 1 - объезд)
    - userid: ID учетной записи
    - login: логин
    - hash: хэш пароля
  + Ответ: boolean
* **Запрос на изменение координат метки**
  + Параметры запроса:
    - action: dragMark
    - uuid: уникальный ID точки
    - lat: широта
    - lon: долгота
    - userid: ID учетной записи
    - login: логин
    - hash: хэш пароля
  + Ответ: boolean
* **Запрос на удаление метки**
  + Параметры запроса:
    - action: removeMark
    - uuid: уникальный ID точки
    - userid: ID учетной записи
    - login: логин
    - hash: хэш пароля
  + Ответ: boolean
* **Получение списка исполнителей**
  + Параметры запроса:
    - action: getAgents
    - userid: ID учетной записи
    - login: логин
    - hash: хэш пароля
  + Ответ: массив исполнителей

#### Запросы для авторизации

* **Авторизация**
  + Параметры запроса:
    - action: signin
    - login: логин
    - password: пароль (*опционально*)
    - hash: хэш пароля (*если не указан пароль*)
  + Ответ: массив исполнителей

Терминология

[api](http://ru.wikipedia.org/wiki/API) - Интерфейс программирования приложений (иногда интерфейс прикладного программирования) - набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) для использования во внешних программных продуктах. Используется программистами для написания всевозможных приложений.

[html](http://ru.wikipedia.org/wiki/Html) - стандартный язык разметки документов во Всемирной паутине. Большинство веб-страниц создаются при помощи языка HTML (или XHTML). Язык HTML интерпретируется браузерами и отображается в виде документа в удобной для человека форме.

[DOM](http://ru.wikipedia.org/wiki/Document_Object_Model) - это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

[CSS](http://ru.wikipedia.org/wiki/Css) - формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

[php](http://ru.wikipedia.org/wiki/Php) - скриптовый язык программирования общего назначения, интенсивно применяемый для разработки веб-приложений.

[Node](http://ru.wikipedia.org/wiki/Node.js) - серверная платформа, использующая язык программирования JavaScript, основанная на движке V8. Предназначена для создания масштабируемых распределённых сетевых приложений, таких как веб-сервер.

[javascript](http://ru.wikipedia.org/wiki/Javascript) - прототипно-ориентированный сценарный язык программирования. Является диалектом языка ECMAScript. JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности веб-страницам.

[coffeescript](http://ru.wikipedia.org/wiki/CoffeeScript) - язык программирования, транслируемый в JavaScript. CoffeeScript добавляет синтаксический сахар в духе Ruby, Python, Haskell и Erlang для того, чтобы улучшить читаемость кода и уменьшить его размер.

[json](http://ru.wikipedia.org/wiki/JSON) - текстовый формат обмена данными, основанный на JavaScript и обычно используемый именно с этим языком. Как и многие другие текстовые форматы, JSON легко читается людьми.