Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

**ОТЧЕТ**

по производственной практике

Студент группы 350531: Козяков А.И.

Руководитель практики

от университета: Бурый Я.А.

Руководитель практики

от предприятия: Дудин Д.В.

МИНСК 2017

**СОДЕРЖАНИЕ**

1. ЗАДАНИЕ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ 3
2. СТРУКТУРА ПРЕДПРИЯТИЯ 4
3. Обзор стека технологий, используемых на проекте 8
   1. JavaScript 8
      1. Структура языка 8
      2. Область применения 10
   2. TypeScript 11
      1. Структура языка 8
      2. Область применения 10
      3. Область применения 10
   3. NodeJS 11
4. Знакомство с системой контроля версий - Git 39
5. Выполнение технического задания 21
   1. Определение требований 21
6. Заключение 41
7. ЗАДАНИЕ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Заданием по производственной практике является:

1. Ознакомление со структурой предприятия. Организация управления
2. Изучение продукции, выпускаемой предприятием
3. Изучение среды разработки, средств программирования и фреймворков
4. Изучение работы со средствами контроля версия и трекинга задач
5. Изучение методов подготовки среды для проекта

Место прохождения технологической практики — ООО «ИксБи Софтваре». Руководитель практики от предприятия — Кострома Н.И. B соответствии с указаниями руководителя технологической практики от вуза Бурый Я.А. были определены цели и сформулированы основные задачи.

Дата прибытия на практику – 08.06.2013.

Дата окончания практики – 29.06.2013.

Результатом прохождения практики является оценка руководителя практики, характеристика, а также отзыв руководителей практики от вуза.

1. СТРУКТУРА ПРЕДПРИЯТИЯ

Компания XB Software предоставляет услуги полного цикла по разработке программного обеспечения под заказ, а также разрабатывает и реализует собственные ИТ продукты. Компания основана в 2008 году. Офис располагается в Минске. В компании работают сертифицированные ИТ-специалисты: разработчики веб- и мобильных приложений, дизайнеры, бизнес-аналитики, менеджеры проектов, QA инженеры, маркетологи и менеджеры по продажам, и другие.

Компания создала собственные продукты, в том числе JavaScript-библиотеку виджетов Webix UI library, которая позволяет создавать передовые веб-приложения на базе HTML5.

Технологическая экспертиза XB Software включает в себя:

* + - 1. React.js;
      2. Marionette.js;
      3. Webix;
      4. Node.js;
      5. PHP / Laravel;
      6. .NET;
      7. Java.

В цикле разработки используются следующие базы данных:

* 1. Oracle;
  2. MS SQL;
  3. MySQL;
  4. PostgreSQL;
  5. MongoDB;
  6. Microsoft Access;
  7. SQLite;
  8. Redis

Также широко используются облачные решения, такие как: Amazon, JBoss, WebLogic, WebSphere и другие.

В работе также используются кроссплатформенные технологии, такие как PhoneGap, Webix JavaScript UI library, которые позволяют создавать пользовательские и бизнес-приложения для любых отраслей.

Кроме того, команда XB Software имеет существенный опыт работы с геоданными:

* 1. Google;
  2. Bing;
  3. OpenStreetMap;
  4. Yandex;
  5. Nokia.

Специалисты компании также разрабатывают решения, обеспечивающие поддержку актуальных задач заказчика, в числе которых:

1. Привлечение клиентов и укрепление лояльности;
2. Электронная коммерция;
3. Управление жизненным циклом продукции;
4. Взаимодействие с партнерами;
5. Формирование отчетности и бизнес-аналитика;
6. Бюджетирование и планирование;
7. Управление проектами;
8. Ведение учета и управление ресурсами (ERP) и другие.

За 9+ лет работы компания успешно выполнила более 150 проектов для своих заказчиков, а также успешно реализовала собственные продукты для клиентов из более чем 65 стран мира. Большинство клиентов XB Software находятся в США, Канаде, Великобритании и Европейском Союзе и представляют различные отрасли экономики:

* 1. Промышленность;
  2. Финансы;
  3. Медицина;
  4. Образование;
  5. Социальные сети.

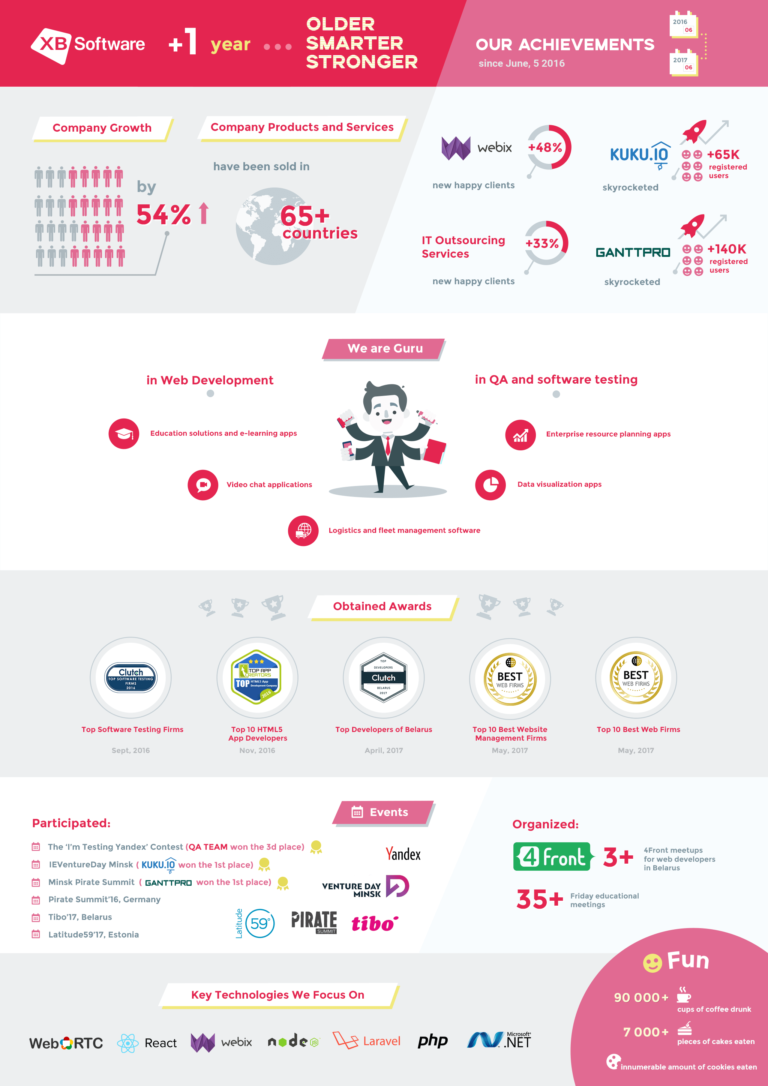
 Компания постоянно растет и развивается. Об это свидетельствует отчет за последний год работы:

Рисунок 2.1. Отчет о работе компании за последние год

Необходимо отметить, что численность сотрудников постоянно увеличивается — это вызвано расширением компании. Регулярно организуются курсы по целенаправленному подбору персонала: как для сотрудников компании, так и для внешних кандидатов. Также огромное значение руководство компании уделяет вопросам повышения квалификации своих сотрудников. Это достигается путём организации тематических и проблемных курсов и семинаров для обмена опытом, а также пятничных лекция на которых всегда рассказывают новую информация из мира IT.

Рис. 2.2 Управленческая структура предприятия

Компания получила следующие награды в своей отрасли:

* 1. Clutch.co - Топ компаний-разработчиков из Беларуси, 2017;
  2. Топ 10 лучших компаний-разработчиков сайтов, 2017;
  3. Топ 10 лучших веб-компаний, 2017;
  4. Clutch.co - Топ компаний по тестированию ПО, 2016;
  5. Топ 10 компаний по разработке приложений на HTML5, 2016;
  6. Clutch.co - Топ 15 компаний-разработчиков из Беларуси, 2015.

1. Обзор стека используемых технологий

## JavaScript

Прототипно-ориентированный [сценарный](http://ru.wikipedia.org/wiki/%25D0%25A1%25D1%2586%25D0%25B5%25D0%25BD%25D0%25B0%25D1%2580%25D0%25BD%25D1%258B%25D0%25B9_%25D1%258F%25D0%25B7%25D1%258B%25D0%25BA) [язык программирования](http://ru.wikipedia.org/wiki/%25D0%25AF%25D0%25B7%25D1%258B%25D0%25BA_%25D0%25BF%25D1%2580%25D0%25BE%25D0%25B3%25D1%2580%25D0%25B0%25D0%25BC%25D0%25BC%25D0%25B8%25D1%2580%25D0%25BE%25D0%25B2%25D0%25B0%25D0%25BD%25D0%25B8%25D1%258F). Является [диалектом](http://ru.wikipedia.org/wiki/%25D0%2594%25D0%25B8%25D0%25B0%25D0%25BB%25D0%25B5%25D0%25BA%25D1%2582_(%25D0%25BF%25D1%2580%25D0%25BE%25D0%25B3%25D1%2580%25D0%25B0%25D0%25BC%25D0%25BC%25D0%25B8%25D1%2580%25D0%25BE%25D0%25B2%25D0%25B0%25D0%25BD%25D0%25B8%25D0%25B5)) языка [ECMAScript](http://ru.wikipedia.org/wiki/ECMAScript).

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам [приложений](http://ru.wikipedia.org/wiki/%25D0%259A%25D0%25BE%25D0%25BC%25D0%25BF%25D1%258C%25D1%258E%25D1%2582%25D0%25B5%25D1%2580%25D0%25BD%25D0%25B0%25D1%258F_%25D0%25BF%25D1%2580%25D0%25BE%25D0%25B3%25D1%2580%25D0%25B0%25D0%25BC%25D0%25BC%25D0%25B0). Наиболее широкое применение находит в [браузерах](http://ru.wikipedia.org/wiki/%25D0%2591%25D1%2580%25D0%25B0%25D1%2583%25D0%25B7%25D0%25B5%25D1%2580) как язык сценариев для придания [интерактивности](http://ru.wikipedia.org/wiki/%25D0%2598%25D0%25BD%25D1%2582%25D0%25B5%25D1%2580%25D0%25B0%25D0%25BA%25D1%2582%25D0%25B8%25D0%25B2%25D0%25BD%25D0%25BE%25D1%2581%25D1%2582%25D1%258C) [веб-страницам](http://ru.wikipedia.org/wiki/%25D0%2592%25D0%25B5%25D0%25B1-%25D1%2581%25D1%2582%25D1%2580%25D0%25B0%25D0%25BD%25D0%25B8%25D1%2586%25D0%25B0).

Основные архитектурные черты: [динамическая типизация](http://ru.wikipedia.org/wiki/%25D0%2594%25D0%25B8%25D0%25BD%25D0%25B0%25D0%25BC%25D0%25B8%25D1%2587%25D0%25B5%25D1%2581%25D0%25BA%25D0%25B0%25D1%258F_%25D1%2582%25D0%25B8%25D0%25BF%25D0%25B8%25D0%25B7%25D0%25B0%25D1%2586%25D0%25B8%25D1%258F), слабая типизация, автоматическое управление памятью, [прототипное программирование](http://ru.wikipedia.org/wiki/%25D0%259F%25D1%2580%25D0%25BE%25D1%2582%25D0%25BE%25D1%2582%25D0%25B8%25D0%25BF%25D0%25BD%25D0%25BE%25D0%25B5_%25D0%25BF%25D1%2580%25D0%25BE%25D0%25B3%25D1%2580%25D0%25B0%25D0%25BC%25D0%25BC%25D0%25B8%25D1%2580%25D0%25BE%25D0%25B2%25D0%25B0%25D0%25BD%25D0%25B8%25D0%25B5), функции как [объекты первого класса](http://ru.wikipedia.org/wiki/%25D0%259E%25D0%25B1%25D1%258A%25D0%25B5%25D0%25BA%25D1%2582_%25D0%25BF%25D0%25B5%25D1%2580%25D0%25B2%25D0%25BE%25D0%25B3%25D0%25BE_%25D0%25BA%25D0%25BB%25D0%25B0%25D1%2581%25D1%2581%25D0%25B0).

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java, но при этом лёгким для использования [непрограммистами](http://ru.wiktionary.org/wiki/%25D0%25BF%25D0%25BE%25D0%25BB%25D1%258C%25D0%25B7%25D0%25BE%25D0%25B2%25D0%25B0%25D1%2582%25D0%25B5%25D0%25BB%25D1%258C-%25D0%25BD%25D0%25B5%25D0%25BF%25D1%2580%25D0%25BE%25D0%25B3%25D1%2580%25D0%25B0%25D0%25BC%25D0%25BC%25D0%25B8%25D1%2581%25D1%2582). Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.

Название «JavaScript» является зарегистрированным [товарным знаком](http://ru.wikipedia.org/wiki/%25D0%25A2%25D0%25BE%25D0%25B2%25D0%25B0%25D1%2580%25D0%25BD%25D1%258B%25D0%25B9_%25D0%25B7%25D0%25BD%25D0%25B0%25D0%25BA) компании [Oracle Corporation](http://ru.wikipedia.org/wiki/Oracle_Corporation).

### Структура языка

Структурно JavaScript можно представить в виде объединения трёх чётко различимых друг от друга частейю

1. ядро (ECMAScript),
2. объектная модель браузера ([Browser Object Model или BOM](http://ru.wikipedia.org/w/index.php?title=Browser_Object_Model&action=edit&redlink=1) ),
3. объектная модель документа ([Document Object Model или DOM](http://ru.wikipedia.org/wiki/Document_Object_Model)).

Если рассматривать JavaScript в отличных от браузера окружениях, то объектная модель браузера и объектная модель документа могут не поддерживаться.

Объектную модель документа иногда рассматривают как отдельную от JavaScript сущность, что согласуется с определением DOM как независимого от языка интерфейса документа. В противоположность этому ряд авторов находят BOM и DOM тесно взаимосвязанными.

ECMAScript не является браузерным языком и в нём не определяются методы ввода и вывода информации. Это, скорее, основа для построения скриптовых языков. Спецификация ECMAScript описывает типы данных, инструкции, ключевые и [зарезервированные](http://ru.wikipedia.org/wiki/%25D0%2597%25D0%25B0%25D1%2580%25D0%25B5%25D0%25B7%25D0%25B5%25D1%2580%25D0%25B2%25D0%25B8%25D1%2580%25D0%25BE%25D0%25B2%25D0%25B0%25D0%25BD%25D0%25BD%25D0%25BE%25D0%25B5_%25D1%2581%25D0%25BB%25D0%25BE%25D0%25B2%25D0%25BE) слова, [операторы](http://ru.wikipedia.org/wiki/%25D0%259E%25D0%25BF%25D0%25B5%25D1%2580%25D0%25B0%25D1%2582%25D0%25BE%25D1%2580_(%25D0%25BF%25D1%2580%25D0%25BE%25D0%25B3%25D1%2580%25D0%25B0%25D0%25BC%25D0%25BC%25D0%25B8%25D1%2580%25D0%25BE%25D0%25B2%25D0%25B0%25D0%25BD%25D0%25B8%25D0%25B5)), объекты, [регулярные выражения](http://ru.wikipedia.org/wiki/%25D0%25A0%25D0%25B5%25D0%25B3%25D1%2583%25D0%25BB%25D1%258F%25D1%2580%25D0%25BD%25D1%258B%25D0%25B5_%25D0%25B2%25D1%258B%25D1%2580%25D0%25B0%25D0%25B6%25D0%25B5%25D0%25BD%25D0%25B8%25D1%258F), не ограничивая авторов производных языков в расширении их новыми составляющими.

Объектная модель браузера — браузер-специфичная часть языка[[30]](http://ru.wikipedia.org/wiki/JavaScript#cite_note-chapmanBOM-35), являющаяся прослойкой между ядром и объектной моделью документа. Основное предназначение объектной модели браузера — управление окнами браузера и обеспечение их взаимодействия. Каждое из окон браузера представляется объектом window, центральным объектом DOM. Объектная модель браузера на данный момент не стандартизирована, однако спецификация находится в разработке [WHATWG](http://ru.wikipedia.org/wiki/WHATWG).

Помимо управления окнами, в рамках объектной модели браузера, браузерами обычно обеспечивается поддержка следующих сущностей.

1. управление [фреймами](http://ru.wikipedia.org/wiki/%25D0%25A4%25D1%2580%25D0%25B5%25D0%25B9%25D0%25BC_(HTML)),
2. поддержка задержки в исполнении кода и зацикливания с задержкой,
3. системные диалоги,
4. управление адресом открытой страницы,
5. управление информацией о браузере,
6. управление информацией о параметрах [монитора](http://ru.wikipedia.org/wiki/%25D0%259C%25D0%25BE%25D0%25BD%25D0%25B8%25D1%2582%25D0%25BE%25D1%2580_(%25D1%2583%25D1%2581%25D1%2582%25D1%2580%25D0%25BE%25D0%25B9%25D1%2581%25D1%2582%25D0%25B2%25D0%25BE)),
7. ограниченное управление историей просмотра страниц,
8. поддержка работы с [HTTP cookie](http://ru.wikipedia.org/wiki/HTTP_cookie).
9. Объектная модель документа
10. Основная статья: [Document Object Model](http://ru.wikipedia.org/wiki/Document_Object_Model)

Объектная модель документа — [интерфейс программирования приложений](http://ru.wikipedia.org/wiki/%25D0%2598%25D0%25BD%25D1%2582%25D0%25B5%25D1%2580%25D1%2584%25D0%25B5%25D0%25B9%25D1%2581_%25D0%25BF%25D1%2580%25D0%25BE%25D0%25B3%25D1%2580%25D0%25B0%25D0%25BC%25D0%25BC%25D0%25B8%25D1%2580%25D0%25BE%25D0%25B2%25D0%25B0%25D0%25BD%25D0%25B8%25D1%258F_%25D0%25BF%25D1%2580%25D0%25B8%25D0%25BB%25D0%25BE%25D0%25B6%25D0%25B5%25D0%25BD%25D0%25B8%25D0%25B9) для HTML и [XML](http://ru.wikipedia.org/wiki/XML)-документов. Согласно DOM, документ (например, веб-страница) может быть представлен в виде дерева объектов, обладающих рядом свойств, которые позволяют производить с ним различные манипуляции:

1. генерация и добавление узлов,
2. получение узлов,
3. изменение узлов,
4. изменение связей между узлами,
5. удаление узлов.

### Область применения

JavaScript используется в клиентской части веб-приложений: клиент-серверных программ, в котором клиентом является браузер, а сервером — веб-сервер, имеющих распределённую между сервером и клиентом логику. Обмен информацией в веб-приложениях происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются кроссплатформенными сервисами.

JavaScript используется в [AJAX](http://ru.wikipedia.org/wiki/AJAX), популярном подходе к построению интерактивных пользовательских интерфейсов веб-приложений, заключающемся в «фоновом» асинхронном обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью и интерфейс веб-приложения становится быстрее, чем это происходит при традиционном подходе (без применения AJAX).

JavaScript широко используется в [браузерных операционных системах](http://ru.wikipedia.org/wiki/WebOS). Так, например, исходный код [IndraDesktop WebOS](http://ru.wikipedia.org/w/index.php?title=IndraDesktop_WebOS&action=edit&redlink=1) на 75 % состоит из JavaScript, код браузерной операционной системы [IntOS](http://ru.wikipedia.org/w/index.php?title=IntOS&action=edit&redlink=1) — на 70 %. Доля JavaScript в исходном коде [eyeOS](http://ru.wikipedia.org/wiki/EyeOS) — 5 %, однако и в рамках этой операционной системы JavaScript играет важную роль, участвуя в визуализации на клиенте и являясь необходимым механизмом для коммуницирования клиента и сервера.

Пользовательские скрипты в браузере — это программы, написанные на JavaScript, выполняемые в браузере пользователя при загрузке страницы. Они позволяют автоматически заполнять формы, переформатировать страницы, скрывать нежелательное содержимое и встраивать желательное для отображения содержимое, изменять поведение клиентской части веб-приложений, добавлять элементы управления на страницу и т. д.

Для управления пользовательскими скриптами в [Mozilla Firefox](http://ru.wikipedia.org/wiki/Mozilla_Firefox) используется расширение [Greasemonkey](http://ru.wikipedia.org/wiki/Greasemonkey); [Opera](http://ru.wikipedia.org/wiki/Opera) и [Google Chrome](http://ru.wikipedia.org/wiki/Google_Chrome) предоставляют средства поддержки пользовательских скриптов и возможности для выполнения ряда скриптов Greasemonkey.

Приложения, написанные на JavaScript, могут исполняться на серверах, использующих [Java](http://ru.wikipedia.org/wiki/Java) 6 и более поздних версий. Это обстоятельство используется для построения серверных приложений, позволяющих обрабатывать JavaScript на стороне сервера.

## TypeScript

Несомненно, JavaScript хорош, но в нем есть ряд недостатков, особенно когда дело касается разработки крупного проекта. Именно этой целью и задали разработчики Microsoft - и вот уже в 2014 году миру был представлена первая версия TypeScript (далее TS).

В чем особенность данного языка? TS по сути является тем же JavaScript-ом, но, в отличии от последнего, в нем есть:

1. Строгая типизация;
2. Интерфейсы;
3. Возможность компиляции в любую из версий JS.

3.2.1 Строгая типизация

Строгая типизация важна. Она позволяет сократить количество ошибок в коде еще на этапе разработки. Всегда приятно знать параметры какого типа нужно передавать в функцию. Мы значительно сокращаем время разработки, потому что нам не нужно заходить внутрь функции, чтобы понять какие же параметры в нее нужно передавать. Это также полезно, когда в команде появляется новые разработчик. Пирог вхождения в проект для него значительно сокращается, потому читаемость кода великолепна. Да и вероятность ошибки сделанной новичком значительно сокращается, что не может не радовать.

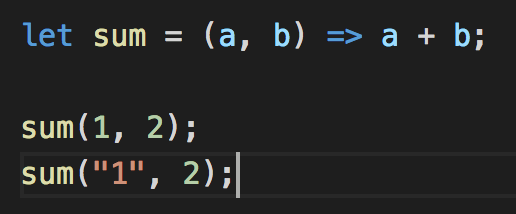
 Напишем функцию сложения двух числе на JS:

Рисунок 3.2.1.1. Функция сложения двух чисел на JS

Как видно из примера она довольно проста, принимает два параметра и отдает их сумму. Чуть ниже на рисунке представлены вызовы данной функции. Как видно, в качестве аргумента функции мы можем передать переменную любого типа. Но если представить, что функция будет иметь больший размер и точно зависеть от типа данных, переданных на ее вход, то вероятность ошибки при вызове данной функции довольно велика.

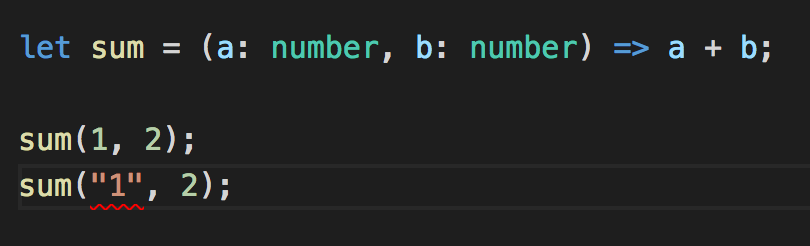
 Рассмотрим пример написанный на TS:

Рисунок 3.2.1.1 Функция сложения двух чисел на TS

Как видно, к предыдущему примеру только дописали типы входных переменных. Но даже сейчас не компилирую данный файл, файловый редактор подчеркнул второй вызов функции, помечая не валидный вызов функции. По-моему, это здорово. Когда часть работы при написании кода можно переложить на рабочую машину и сосредоточится над более важными вещами.

3.2.2 Интерфейсы

Второй важной характеристикой TS является наличие интерфейсов. Для чего же они нужны? В разработке очень часто работает с некоторыми структурами данных, а когда эту структуру данных можно описать и использовать во всем проекте, то легче становиться дышать всей команде.

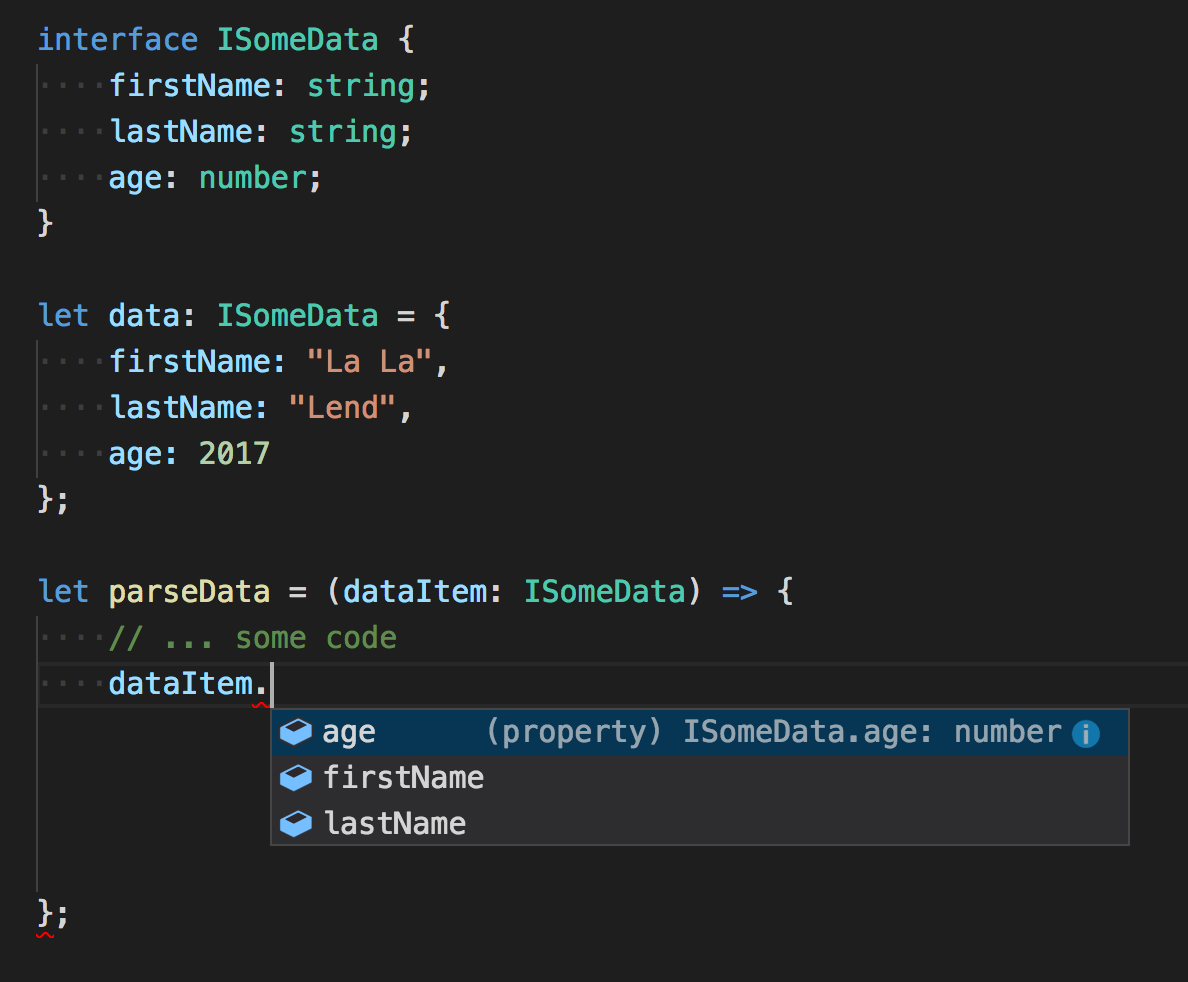
Приведем пример:

Рисунок 3.2.2.2. Пример использования интерфейса

Как видно из рисунка выше, создание и использование интерфейсов ничем не отличается от использования обычных типов, ведь они, по сути, и являются пользовательским типом данных.

Их можно свободно использовать в качестве типа для переменной или для типа аргумента функции. Самым большим плюсом является то, что, используя интерфейсы, мы помогаем текстовому редактору подсказывать нам все его свойства. На примере видно всплывающее окно текстового редактора, которое показывает все возможные свойства данного объекта.

3.2.2 Компиляция в JS

Как рассказывалось выше, браузер не понимает TS, но он понимает JS. Именно поэтому мы вынуждены компилировать один язык в другой. В последние два года идет прогрессивное обновление версий JavaScript. В него добавляются новые возможности, которые улучшают и повышают его функциональность. Но в тоже время, реальность такова, что мы вынуждены поддерживать поддержку новых возможностей JS в старых браузерах.

В данном вопросе TS умеет компилировать даже самую последнюю версию JS в более ранние версии. Это здорово, ведь как только проект перестает поддерживать старый браузер, мы можем безболезненно, поменяв один параметр компиляции, получить наш код в более современной версии.

## NodeJS

Node или Node.js — программная платформа, основанная на движке V8 (транслирующем JavaScript в машинный код), превращающая JavaScript из узкоспециализированного языка в язык общего назначения.

Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API (написанный на C++), подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода. Node.js применяется преимущественно на сервере, выполняя роль веб-сервера, но есть возможность разрабатывать на Node.js и десктопные оконные приложения (при помощи NW.js, AppJS или Electron для Linux, Windows и Mac OS) и даже программировать микроконтроллеры (например, tessel и espruino). В основе Node.js лежит событийно-ориентированное и асинхронное (или реактивное) программирование с неблокирующим вводом/выводом.

В 1996 году в компании Netscape были попытки создания серверного JavaScript (Server-side JavaScript — SSJS), однако технология не получила распространения.

Node.js разработал Райан Даль (англ. Ryan Dahl) в 2009 году после двух лет экспериментирования над созданием серверных веб-компонентов. В ходе своих исследований он пришёл к выводу, что вместо традиционной модели параллелизма на основе потоков следует обратиться к событийно-ориентированным системам. Эта модель была выбрана из-за простоты, низких накладных расходов (по сравнению с идеологией «один поток на каждое соединение») и быстродействия. Целью Node является предложить «простой способ построения масштабируемых сетевых серверов».

Разработка Node.js спонсируется компанией Joyent (англ.).

В декабре 2014 года был создан форк io.js.

В мае 2015 года было принято решение о слиянии io.js и Node.js и дальнейшем развитии под эгидой Node.js Foundation.

8 сентября 2015 года вышел Node.js v4.0.0 как результат слияния Node.js v0.12.7 и io.js v3.3.0.

На данный момент проект все также стремительно развивается и совершенствуется. Текущая версия платформы – 8.1.2. И с каждый годом она становиться все популярнее среди разработчиков.

1. Знакомство с системой контроля версий - Git

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Для примеров в этой книге мы будем использовать исходные коды программ, но на самом деле под версионный контроль можно поместить файлы практически любого типа.

Если вы графический или веб-дизайнер и хотели бы хранить каждую версию изображения или макета — а этого вам наверняка хочется — то пользоваться системой контроля версий будет очень мудрым решением. СКВ даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. Вообще, если, пользуясь СКВ, вы всё испортите или потеряете файлы, всё можно будет легко восстановить. Вдобавок, накладные расходы за всё, что вы получаете, будут очень маленькими.

Как и многие замечательные вещи, Git начинался с, в некотором роде, разрушения во имя созидания и жарких споров. Ядро Linux — действительно очень большой открытый проект. Бо́льшую часть существования ядра Linux (1991-2002) изменения к нему распространялись в виде патчей и заархивированных файлов. В 2002 году проект перешёл на проприетарную РСКВ BitKeeper.

В 2005 году отношения между сообществом разработчиков ядра Linux и компанией, разрабатывавшей BitKeeper, испортились, и право бесплатного пользования продуктом было отменено. Это подтолкнуло разработчиков Linux (и в частности Линуса Торвальдса, создателя Linux) разработать собственную систему, основываясь на опыте, полученном за время использования BitKeeper. Основные требования к новой системе были следующими:

1. Скорость;
2. Простота дизайна;
3. Поддержка нелинейной разработки (тысячи параллельных веток);
4. Полная распределённость;
5. Возможность эффективной работы с такими большими проектами, как ядро Linux.

С момента рождения в 2005 году Git развивался и эволюционировал, становясь проще и удобнее в использовании, сохраняя при этом свои первоначальные качества. Он невероятно быстр, очень эффективен для больших проектов, а также обладает превосходной системой ветвления для нелинейной разработки.

## Локальные системы контроля версий

Многие предпочитают контролировать версии, просто копируя файлы в другой каталог (как правило, добавляя текущую дату к названию каталога). Такой подход очень распространён, потому что прост, но он и чаще даёт сбои. Очень легко забыть, что ты не в том каталоге, и случайно изменить не тот файл, либо скопировать файлы не туда, куда хотел, и затереть нужные файлы.

Чтобы решить эту проблему, программисты уже давно разработали локальные СКВ с простой базой данных, в которой хранятся все изменения нужных файлов (см. рисунок 4.1.1).



Рисунок 4.1.1. Схема локальной СКВ.

Одной из наиболее популярных СКВ такого типа является rcs, которая до сих пор устанавливается на многие компьютеры. Даже в современной операционной системе Mac OS X утилита rcs устанавливается вместе с Developer Tools. Эта утилита основана на работе с наборами патчей между парами версий (патч — файл, описывающий различие между файлами), которые хранятся в специальном формате на диске. Это позволяет пересоздать любой файл на любой момент времени, последовательно накладывая патчи.

## Централизованные системы контроля версий

Следующей основной проблемой оказалась необходимость сотрудничать с разработчиками за другими компьютерами. Чтобы решить её, были созданы централизованные системы контроля версий (ЦСКВ). В таких системах, например, CVS, Subversion и Perforce, есть центральный сервер, на котором хранятся все файлы под версионным контролем, и ряд клиентов, которые получают копии файлов из него. Много лет это было стандартом для систем контроля версий (рисунок 4.2.1).



Рисунок 4.2.1. Схема централизованного контроля версий

Такой подход имеет множество преимуществ, особенно над локальными СКВ. К примеру, все знают, кто и чем занимается в проекте. У администраторов есть чёткий контроль над тем, кто и что может делать, и, конечно, администрировать ЦСКВ намного легче, чем локальные базы на каждом клиенте.

Однако при таком подходе есть и несколько серьёзных недостатков. Наиболее очевидный — централизованный сервер является уязвимым местом всей системы. Если сервер выключается на час, то в течение часа разработчики не могут взаимодействовать, и никто не может сохранить новой версии своей работы. Если же повреждается диск с центральной базой данных и нет резервной копии, вы теряете абсолютно всё — всю историю проекта, разве что за исключением нескольких рабочих версий, сохранившихся на рабочих машинах пользователей. Локальные системы контроля версий подвержены той же проблеме: если вся история проекта хранится в одном месте, вы рискуете потерять всё.

## Распределённые системы контроля версий

И в этой ситуации в игру вступают распределённые системы контроля версий (РСКВ). В таких системах как Git, Mercurial, Bazaar или Darcs клиенты не просто выгружают последние версии файлов, а полностью копируют весь репозиторий. Поэтому в случае, когда "умирает" сервер, через который шла работа, любой клиентский репозиторий может быть скопирован обратно на сервер, чтобы восстановить базу данных. Каждый раз, когда клиент забирает свежую версию файлов, он создаёт себе полную копию всех данных (рисунок 4.3.1).



Рисунок 4.3.1. Схема распределённой системы контроля версий.

Кроме того, в большей части этих систем можно работать с несколькими удалёнными репозиториями, таким образом, можно одновременно работать по-разному с разными группами людей в рамках одного проекта. Так, в одном проекте можно одновременно вести несколько типов рабочих процессов, что невозможно в централизованных системах.

Для совершения большинства операций в Git'е необходимы только локальные файлы и ресурсы, т.е. обычно информация с других компьютеров в сети не нужна. Если вы пользовались централизованными системами, где практически на каждую операцию накладывается сетевая задержка, вы, возможно, подумаете, что боги наделили Git неземной силой. Поскольку вся история проекта хранится локально у вас на диске, большинство операций кажутся практически мгновенными.

К примеру, чтобы показать историю проекта, Git'у не нужно скачивать её с сервера, он просто читает её прямо из вашего локального репозитория. Поэтому историю вы увидите практически мгновенно. Если вам нужно просмотреть изменения между текущей версией файла и версией, сделанной месяц назад, Git может взять файл месячной давности и вычислить разницу на месте, вместо того чтобы запрашивать разницу у СКВ-сервера или качать с него старую версию файла и делать локальное сравнение.

Из всего вышесказанного становиться можно сделать вывод, что на сегодняшний день распределенные системы контроля версия являются самыми удобными в использовании. Именно поэтому в задании по практике использовалась система контроля версия Git.

1. Выполнение технического задания

## Определение требований

Основной задачей является создание командной среды для нового проекта:

1. Ознакомиться со стеком технологий необходимым для создания данной среды;
2. Среда должна уметь запустить локальный сервер на заданном порту;
3. В проекте будут использованы такие языки как: Less и TypeScript. Для каждого из языков должны быть команды для их компиляции в CSS и JavaScript соответственно. Компиляция должны быть гибкой и настраиваемой;
4. Для удобства написания кода нужны команды для автоматической компиляции Less и TS после каждого изменения файла;
5. В качестве среды исполнения использовать NodeJS и его экосистему пакетов;
6. Все команды для работы с проектом должны быть доступны из командной строки. Должен присутствовать список всех доступных команд(help);
7. Весь процесс разработки среды должен быть зафиксирован в системе контроля версий Git, который впоследствии будет использован для дальнейшей работы с проектом.

## Выполнение технического задания

На первом этапе нужно было установить систему контроля версий Git. Это сделать довольно просто. Переходим по ссылке: <https://git-scm.com/downloads> и скачиваем самую последнюю версию Git для своей операционной системы.

Далее необходимо зарегистрировать аккаунт на GitHub. Он доступен по ссылке: <https://github.com/>.

После регистрации нужно войти в аккаунт и создать новый проект. Для этого переходим по ссылке: <https://github.com/new> и заполняем необходимые поля (рисунок 5.2.1).

После заполнения всех полей жмем кнопку создания репозитория и ожидаем выполнения операции. После успешного выполнения будет выведено сообщение о том, как можно скопировать созданный репозиторий на свою рабочую машину (рисунок 5.2.2). В командную строку своей операционной системы вводим описанные команды и получаем рабочий репозиторий. Для безопасной работы с репозиторием нужно соединить рабочую машину с git-аккаунтом посредством SSH-ключа. Это делается по ссылке: https://help.github.com/articles/connecting-to-github-with-ssh/

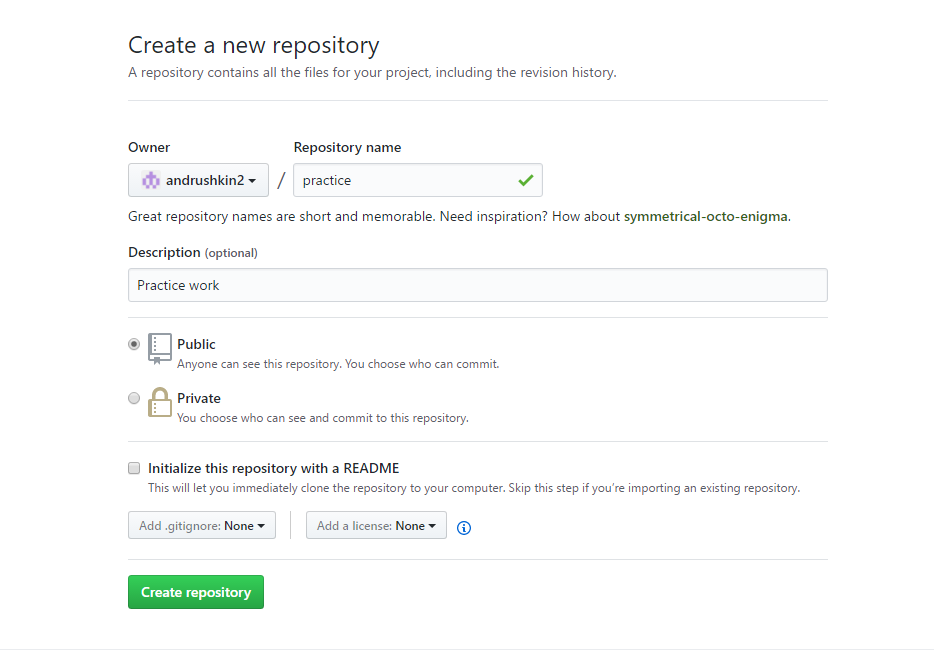


Рисунок 5.2.1. Создание нового репозитория

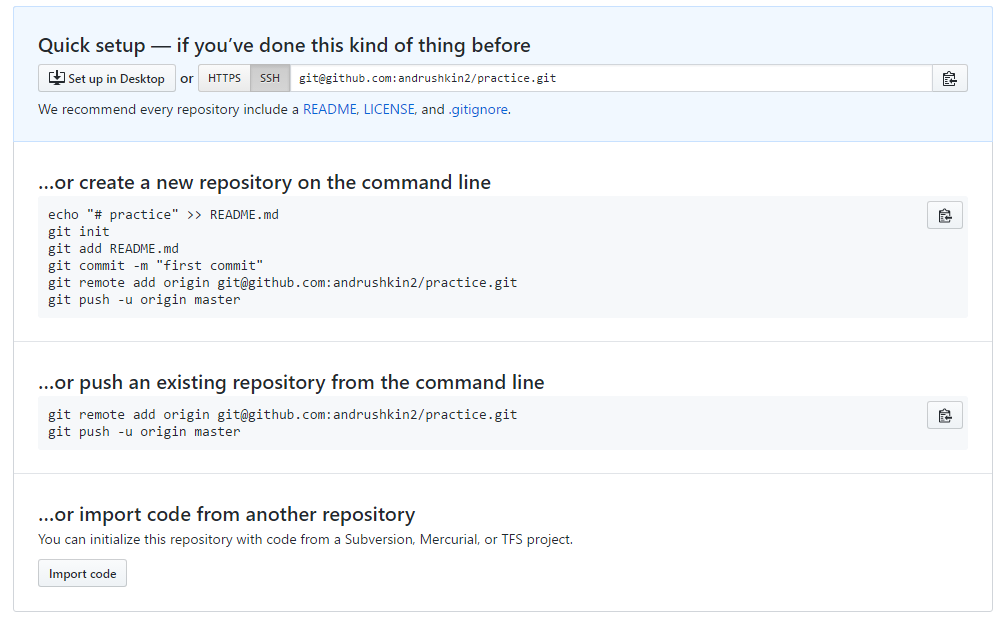


Рисунок 5.2.2. Создание репозитория на рабочей машине

Далее необходимо установить на рабочую машину NodeJS. Его можно скачать по ссылке: <https://nodejs.org/en/>. После установки приложения можно проверить его работу при помощи команды: node –v (рисунок 5.2.3).

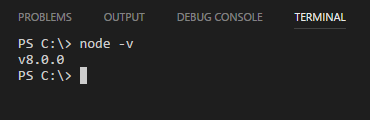


Рисунок 5.2.3. Версия NodeJS

Итак, можно приступать написанию команд для работы с проектом. NodeJS имеет огромную систему пакетов, с помощью которых можно сделать практически все. Но сначала, мы создадим конфигурационный файл для NodeJS-проекта. Для этого можно воспользоваться пакетным менеджером NPM, который идет в комплекте с NodeJS.

Вводим команду: «npm init» и заполняем запрашиваемые поля. После заполнения последнего можно еще раз взглянуть на создаваемый файл и подтвердить его создание (рисунок 5.2.4)

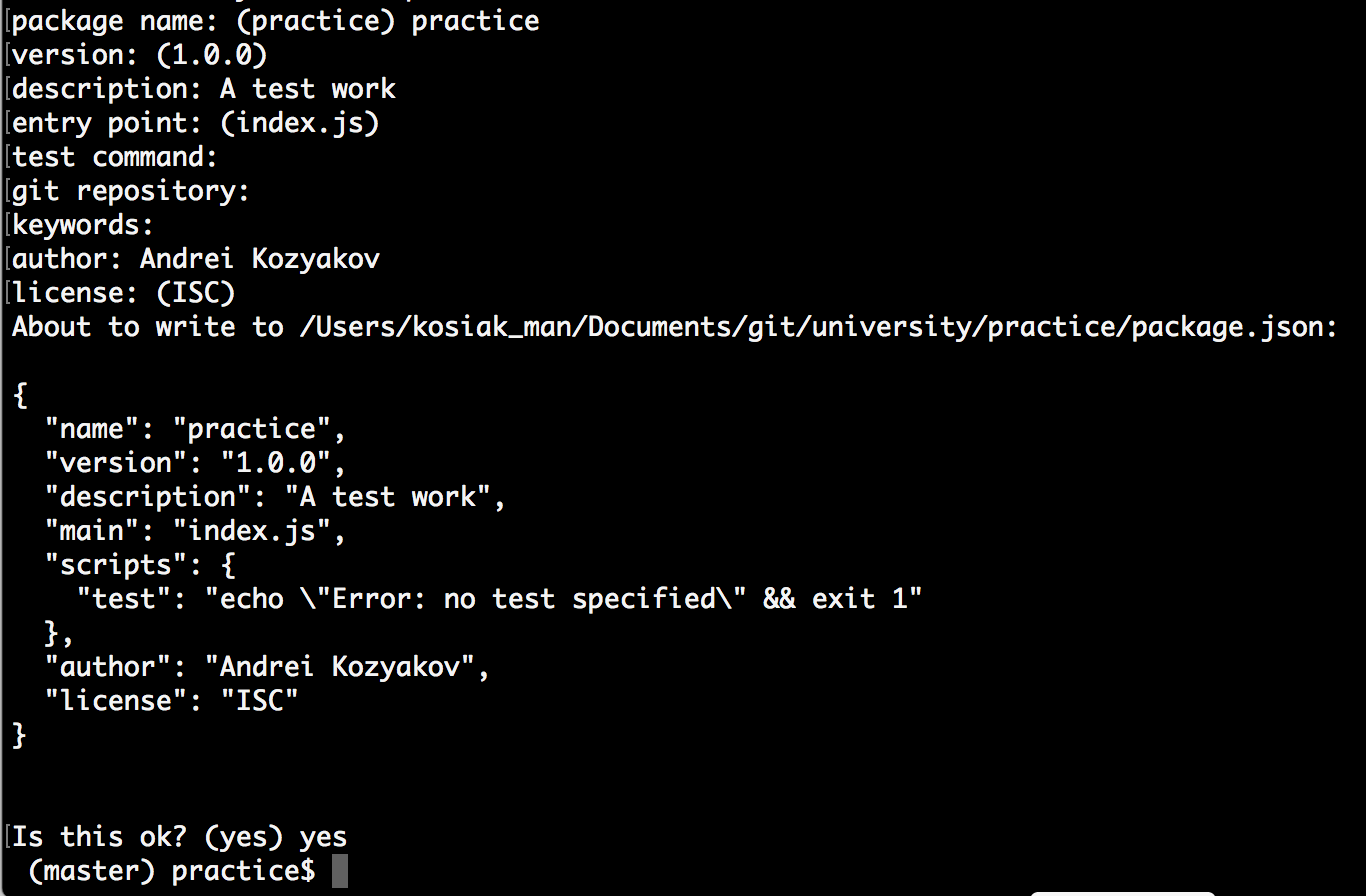


Рисунок 5.2.4. Конфигурационный файл проекта

Проделанную работу нужно сохранить в Git-репозиторий. Для этого последовательно вызввает две команды:

1. «git add .» – для добавления всех измененных файлов в закрепленное состояние;
2. «git commit –m “Add config file”» - для закрепления добавленных файлов в системе контроля версий.

При желании, можно сохранить изменения на сервере, использую команду «git push», но на данном этапе это не критично.

Теперь начинаем установку необходимых пакетов. NodeJS содержит очень большое их количество. Нам же нужны: пакет для сервера, для компиляции LESS в CSS и TypeScritp в JavaScript, а также менеджер, работающий с задачами.

Для сервера будем использовать ImpressJS. Для установки данного пакета в командной строке нужно ввести строку: «npm install impressjs –-save-dev». «Save-dev» флаг используется для того, чтобы дописать данный пакет в конфигурационный файл, как необходимый. В результате получим сообщение о скаченном пакете(рисунок 5.2.5)

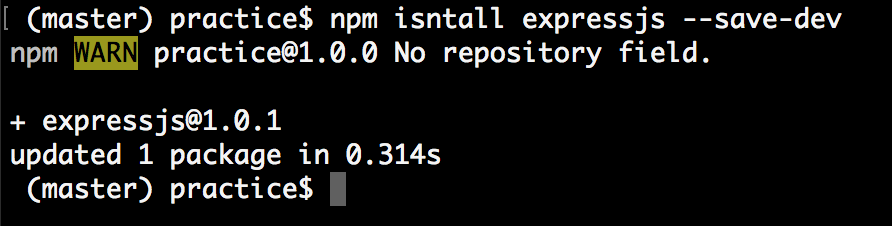


Рисунок 5.2.5. Загрузка пакета ExpressJS

Казалось бы, можно продолжать работать. Но если ввести команду: «git status», то станет понятно, что у нас добавилась новая папка «node-modules» (рисунок 5.2.6), в которую записываются все пакеты для проекта.

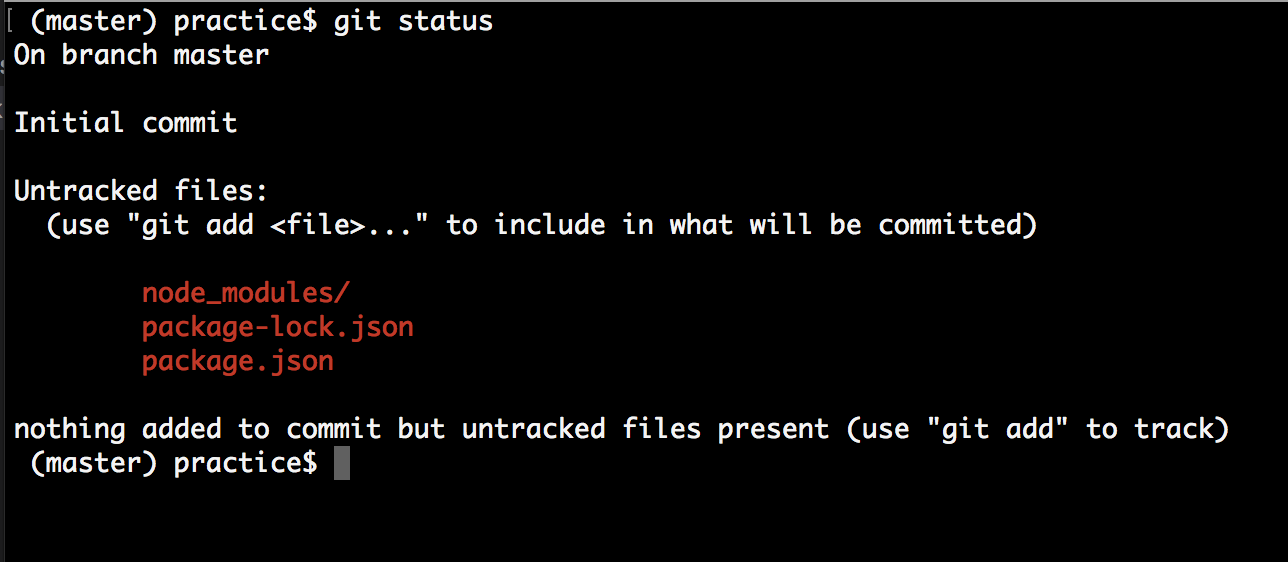


Рисунок 5.2.6. Состояние git-репозитория

Если подумать, то нам не нужно хранить все эти пакеты на сервере репозитория. Каждый установит их себе локально и git-репозиторий будет значительно меньше заниматься места. Для этого создается «.gitignore” файл, в котором указываются файлы и папки, которые не будут видны для git. После внесения ненужной папки в файл она пропадает из зоны видимости git, как это вижно из рисунка 5.2.7.

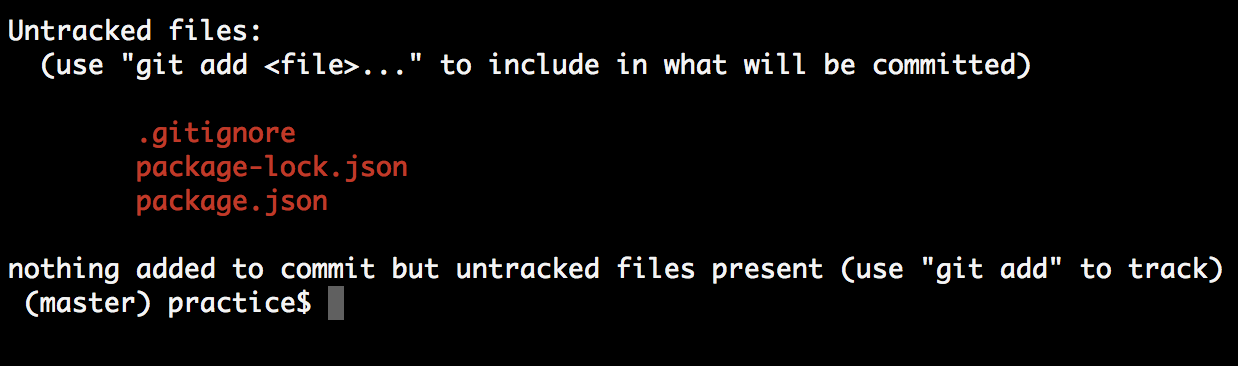


Рисунок 5.2.7. Репозиторий после скрытия папки node\_modules

По аналогии скачиваем оставшиеся пакеты в репозиторий.

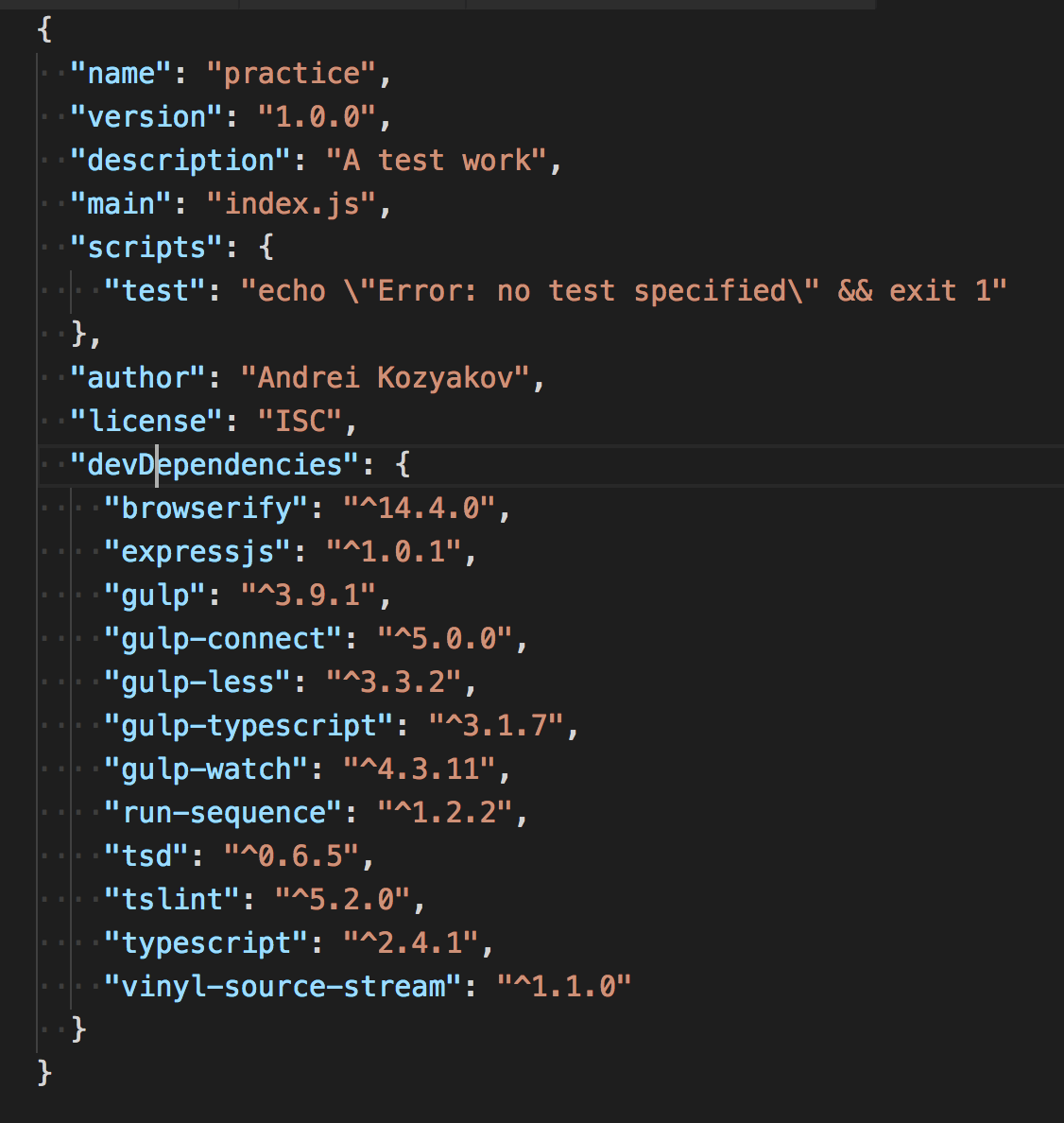


Рисунок 5.2.8. Конфигурация проекта после установки пакетов

После установки всех пакетов все установленные пакеты будут отражены в файле конфигурации. Любой другой человек имея этот файл сможет одной командой: «npm install» установить сразу все пакеты необходимые для работы с командной средой.

В качестве менеджера задач будет использоваться Gulp – он также входит в пакет NPM. В чем его суть? Имеется gulp-файл в котором записаны всевозможные задачи, для запуска той или иной задачи требуется написать в командной строке: «gulp имя\_задачи».

Нечем с сервера. Сначала создадим совсем примитивный сервер, который будет запускаться как задача.



Рисунок 5.2.9. Сервер на основе пакета ExpressJS

Запускаем команду из консоли и сервер запущен:

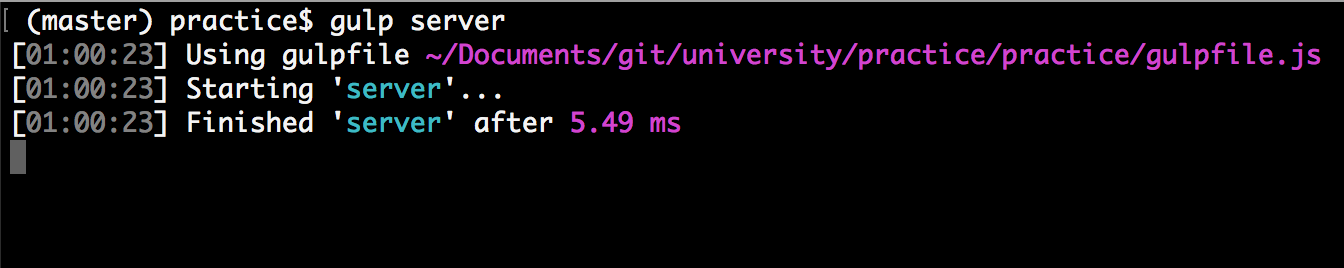


Рисунок 5.2.10. Запущенный сервер из gulp-команды

Для компиляции Less в CSS будем использовать пакет gulp-less. Подключим пакет к исполняемому файлу и создаем новую задачу:



Рисунок 5.2.11. Код компиляции Less в CSS

После написания кода выполняем тестовый запуск:

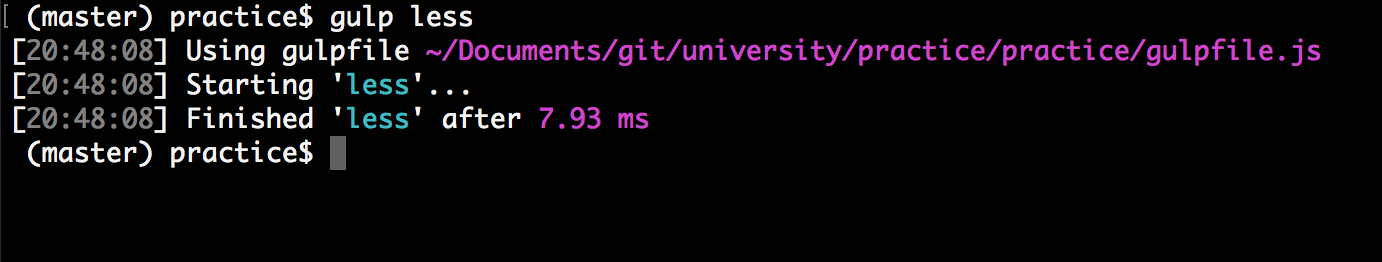


Рисунок 5.2.12. Запуск компиляции Less в CSS

Теперь преступим к компиляции TS в JS. Проделаем те же действия что и в примере выше:



Рисунок 5.2.13. Код для компиляции TS в JS

Теперь можно притупить написанию задачи для автоматического обновления LESS и TS файлов после редактирования, для этого используется функция Gulp.watch, которая наблюдает за наобходимыми файли. При изменении любого файла будет вызываться функцию, в которой можно будет выполнять любые действия. На рисунке 5.2.14 представлен код задачи для выполнения данной операции.



Рисунок 5.2.15. Код задачи для автоматической компиляции файлов

Как видно из кода, после изменения какого-либо файла вызывается соответствующая задача, которая знает, что с ним делать.

Создадим также команду для запуска всего проекта. Она будет сначала компилировать все необходимые файлы, а затем запускать сервер:

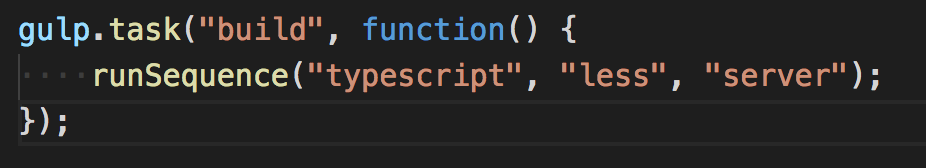


Рисунок 5.2.16. Команда для сборки всего проекта

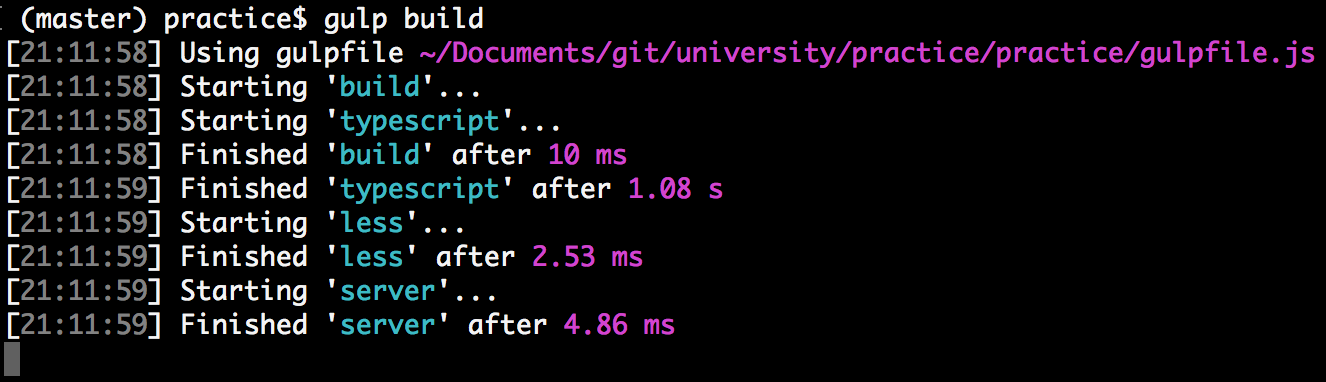


Рисунок 5.2.17. Результат сборки всего проекта

Добавим задачу для вывода справки о доступных командах:



Рисунок 5.2.18. Задача для вывода справки

Все выполненные изменения фиксируем в системе контроля версий.

1. Заключение

В результате прохождения технологической практики были решены задачи, поставленные руководителем практики от предприятия ООО «ИксБи Софтваре»: изучение стека технологий, а также разработка командной среды для нового проекта. Было проведено тестирование модуля.

Была изучена необходимая техническая документация, патентные и литературные источники, действующие стандарты, технические условия, положения и инструкции по разработке, наладке и эксплуатации, методы и технологии проектирования.

В соответствии с программой практики и индивидуальным заданием был осуществлен сбор, анализ, систематизация информации по теме задания и его непосредственное выполнение.

Также был подготовлен настоящий отчет и дневник с характеристикой руководителя от предприятия.

В ходе работы были получены навыки работы на предприятии, командной работы, общения с заказчиком. В ходе прохождения практики была изучена система контроля версий Git.