Разработка front-end: Use case

(слайд 1)
Здравствуйте, меня зовут Чеботарев Александр, я работаю фронт-енд разработчиком в компании Фогсофт.
Сегодня мы поговорим о том, как происходит процесс фронт-енд разработки в реальной жизни. В начале я в общих словах расскажу о настройке проекта и используемых инструментах, остальное время - сверстаем макет и на практике разберем часто встречающиеся задачи.

(слайд 2)
Разработка клиентской части должна начинаться с макетов.

(слайд 3)
В идеальном мире в зависимости от проекта это должны быть макеты нескольких страниц, на которых представлены все возможные элементы, которые только могут встретиться на страницах. В идеальном мире должно быть по несколько вариантов каждого макета - для разных размеров экрана. На макетах должны быть указаны размеры элементов, отступов, кегль шрифта, цвета и пр.
В ещё более идеальном мире ещё и описано поведение элементов.
Разработчик должен только реализовать макет, который ему предоставили.

(слайд 4)
В реальной жизни такое происходит не часто, и зачастую разработчик кроме тз получает 1-2 макета, на основании чего и строится клиентская часть веб-приложения. Далее верстальщик просто подгоняет элементы так, чтобы они более-менее соответствовали макету.

Все любят котиков, поэтому сегодня мы будем делать сайт про них. Это будет небольшой лэндинг из нескольких страниц. У нас есть вот такой макет:
awesome-mockup.png

В простейшем виде, клиентская часть - это набор .html, .css и .js файлов. Для небольших статичных сайтов этого достаточно и можно обойтись без дополнительных инструментов. Но с увеличением сложности проекта, потребность в них возрастает.

В многостраничном сайте обязательно присутствуют общие элементы: хедер, футер и т.д. Их можно, конечно, копипастить, но как только где-то что-то поменялось, это приходится изменять на всех страницах. От этой проблемы нас избавляет использование шаблонизатора.

(слайд 5)
В большинстве случаев шаблонизаторы собирают страницу из разных кусков на сервере и отправляют готовую страницу на клиент.

Шаблонизаторы позволяют:

- задавать лейаут - общую часть всех страниц. Например, head-элемент со всем содержимым, часть body: хедер, футер и пр. Сам контент страницы будет подставляться в указанное место в лейауте;
- переиспользовать элементы. Часто используемые блоки можно выносить в шаблоны и вставлять их в нужное место страницы;
- подставлять данные сразу на сервере (в случае, если страницы собираются на сервере). Помогает избежать лишних запросов на сервер за данными.

Сейчас адаптивность - это стандартное требование для сайта. Можно написать свой набор стилей для адаптивности, но проще подыскать готовое решение - библиотек и фреймворков для решения этой задачи много.

(слайд 6)
Наверное, самым популярным из них является Bootstrap. Это достаточно объемный фреймворк, который включает в себя практически всё необходимое для быстрой стилизации сайта - стили для адаптивности, стили для различных элементов и контролов, несколько js библиотек для интерактивных элементов (для них нужно будет дополнительно подключить jquery).
Т.к. не нужно тащить на сайт всё подряд, нам нужно сделать свою сборку Bootstrap-а.

Переходим на страницу кастомизации.
Здесь можно отключить ненужные компоненты, а в блоке “Less variables” у нас есть возможность изменить переменные, определяющие цвета, размеры элементов, настройки шрифтов, настройки адаптивности и другие.
В Bootstrap-е стили разбиты по разным файлам для каждого компонента, плюс файл с переменными, все они имеют расширение .less. Результирующий .css файл собирается из них с помощью препроцессоров и постпроцессоров css.

(слайд 7)
Препроцессоры и постпроцессоры css призваны упростить жизнь разработчика.

(слайд 8)
Препроцессоры css собирают .css файлы из файлов специального формата - .less или .sass. Они позволяют:
- использовать в стилях переменные;
- объединять часто повторяющиеся группы правил в примеси (mixins), и переиспользовать их;
- использовать функции и циклы.

(слайд 9)
Постпроцессоры уже работают с .css файлами.
Как пример постпроцессора можно рассмотреть autoprefixer. Он решает проблему поддержки старых браузеров, в которых некоторые современные свойства реализованы с помощью вендорных префиксов или через другие свойства. В его настройках мы указываем, какие браузеры хотим поддерживать, и просто пишем стили для последней версии, всё остальное он делает за нас.

Конкатенация и минификация файлов - тоже может быть реализована с помощью постпроцессоров.

Кроме всего этого, в проекте может понадобиться изменение .js файлов - те же конкатенация и минификация.

Все эти задачи можно запускать вручную, когда будет необходимо. Если у нас будет развернуто несколько версий проекта (например, отдельно для разработки, для тестирования и продакшн версия), то для них может понадобится выполнять разные действия. Но всё это можно автоматизировать с помощью сборщиков проектов.

(слайд 10)
В конфигурационном файле сборщика проектов указываются задачи, которые он должен выполнить. Для разных версий можно задать разные задачи, например, минифицировать файлы только для продакшн версии. Сборщик можно запускать вручную, либо настроить его запуск при билде всего проекта, либо повесить отслеживание изменения в определенных файлов, и тогда он будет запускаться автоматически.

(слайд 11)
В этом примере я буду использовать gulp. Другие наиболее известные сбощики - это grunt и webpack.

Для его установки нам потребуется также установить node package manager.
Менеджер пакетов используется для подключения к нашему проекту различных библиотек.
node package manager устанавливается вместе с node.js, на котором в этом проекте сделана серверная часть.

С настройкой проекта закончено…

С учетом того, что над проектом может работать несколько человек, со своим стилем написания кода, и для упрощения дальнейшей поддержки проекта, нужно ввести определенные правила, которым должны следовать разработчики.

Существует множество методологий организации верстки. Все они призваны тем или иным образом стандартизировать написание стилей и html-кода, упростить его поддержку в дальнейшем и предотвратить превращение проекта в помойку.
В основном методологии устанавливают правила именования классов, использования различных селекторов и разбиения страницы на блоки.
Подробную информацию вы можете найти в интернете.

верстка сайта по макету
(слайд 12)