1. Качественные и некачественные клиентские приложения, как я это понимаю

Следующие приложения качественны с точки зрения:

1. пользователя:
– отзывчивые во взаимодействии с пользователем;
– высокопроизводительные;
– при надобности максимально простые;
– при надобности максимально информативные, понятные.

2. менеджера проекта:
– с максимально качественно прописанным кодом;
– максимально просто дорабатываемые;
– максимально быстро разрабатываемые;
– требующие наименьшее количество ресурсов для разработки.

3. дизайнера:
– с приятным, красивым внешним видом;
– позволяющие реализовать все дизайнерские приёмы.

4. верстальщика:
– разработанные с соблюдением общепринятых и локально-принятых правил вёрстки;
– с максимально «читаемой» вёрсткой.

5. серверного программиста:
– разработанные с соблюдением общепринятых и локально-принятых правил взаимодействия с серверами;
– с предсказуемым поведением при взаимодействии с сервером.

2. Особенности крупных модифицируемых разработок

Существенного опыта участия в подобных разработках не имею, но имею некоторое представление и теорию. Пишу согласно им.

1. Размер. Со временем программа может стать настолько большой, что становится сложно запомнить, какая часть кода для чего предназначена. Для решения этой проблемы создают справочники.

2. Несколько разработчиков. С ростом проекта растёт количество рабочих, обслуживающих код – при том по разным направлениям разработки. Эту проблему решают с привлечением централизованного управления разработкой, общих локальных стандартов, систем распределения файлов по сотрудникам.

3. Модификации. При модификации крупного приложения полезны справочники по приложению, стандарты, автоматические тесты для гарантии правильной работы задействованных частей модифицируемого приложения.

3. Presentational Components и Container Components

С рассматриваемыми понятиями знаком по официальной документации к Redux. Отвечаю так, как знаю по этой документации.

Redux выступает как хранилище состояния приложения.
Разделение на данные виды компонентов улучшает понимание прописанного кода, упрощает взаимодействие с компонентами. О недостатках на данный момент не знаю.

Presentational Components:
– Предназначены для хранения визуализирующего кода (разметка, стили).
– Не взаимодействует с Redux напрямую.
– Прописываемы вручную.

Container Components:
– Предназначены для взаимодействия с данными (извлечение данных, обновление state).
– Взаимодействует с Redux напрямую.
– Обычно их генерирует сам Redux.

4. Наследование в JS

В JS реализовано наследование через прототипы. Прототипом какого-либо объекта может быть какой-либо другой объект. При этом у одного объекта не может быть более одного прототипа. Эту проблему решают через mixin-ы (примеси), копируя данные из них в прототип, дополняя его этими данными.

Далее для обозначения сущностей в коде: Child/child – объект-наследник, Parent/parent - объект-прототип.
Присвоение прототипа происходит через записывание в свойство __proto__ объекта-наследника ссылки на объект-прототип. Для поддержки старых браузеров можно создавать объект с заранее присвоенным прототипом через Object.create(prototype). В том числе, можно задать объект без прототипа, передав в качестве аргумента null.

У функций-конструкторов (классов) для реализации принципов прототипного наследования есть специальное статическое свойство ”prototype” (Constructor.prototype = …) – в качестве значения принимает объект. В это свойство записывают свойства и методы, предназначенные для всех объектов данного класса. Для того чтобы назначить класс в качестве прототипа другому классу, классу-наследнику в ”prototype.__proto__” записывают ссылку на свойство ”prototype” наследуемого класса. (Способы записи рассмотрены выше.)
Изначально у функций в свойстве ”prototype” – объект со свойством ”constructor”, ссылающееся на саму функцию. Это важно знать, если важно не потерять это свойство при перезаписи через prototype.

Чтобы не дублировать код в конструкторе-наследнике, можно вызвать Parent.apply(this, arguments) со всеми переданными конструктору аргументами.
Функция с одинаковым именем в prototype наследника перекроет аналогичную функцию наследуемого класса. Функцию наследуемого класса можно вызвать в контексте наследника через Parent.prototype.function.apply(this, arguments).

В ES6 реализован дополнительный синтаксис классов поверх существующих принципов

JS. Со своими особенностями. Среди прочего добавлена конструкция ”super” для обращения к наследуемому классу из методов. (class className [extends parentClass] { конструктор... методы... }).

Для проверки принадлежности к цепи прототипов существует instanceof (object instenceof Constructor).

Цикл for...in перебирает в том числе свойства прототипа объекта. Чтобы узнать, относится ли свойство к объекту, используют child.hasOwnProperty(property).

5. End-to-end тестирование

Опыта тестирования веб-приложений на данный момент не имею.

С end-to-end тестированием знаком отдалённо. Для такого тестирования во фронтенде, полагаю, можно использовать Selenium WebDriver, Puppeter.

6. Форма без описания

Не уверен, что понял, о чём именно вопрос. В подобных описанной ситуациях планирую вести себя в соответствии с описанием задания, указаниями, стандартами. Не потратить лишний раз время на реализацию ненужного функционала и не пренебречь реализацией необходимого функционала.

Предположительно, описанной статичной вёрстки было бы достаточно в рамках требований к проекту. Некоторую динамичность, вероятно, задали бы стандартные механизмы браузеров (как они, например, предупреждают, если поле формы в соответствии с вёрсткой требует заполнения).

7. Инструменты для экономии времени

До сих пор использовал некоторые стандартные инструменты разработчика в браузере Firefox. Консоль иногда могла помочь моментально проверить некоторые особенности JS, работоспособность некоторой тактики написания кода. При поиске ошибок помогал debugger, который я вызывал из кода.

Я использовал локальный сервер через webpack, с автоматической перезагрузкой, для отображения сайта в актуальном состоянии в браузере. Скрипты, прописанные в package.json тоже экономят время. Webpack совместно с loader-ми позволяет за раз перевести весь специфический код в понятный браузеру. Некоторые модули ускоряют написание кода (например, sass для css может упростить написание и повысить читаемость, как и JSX в связке с babel).

Если бы я писал код не в блокноте, и среда разработки подсказывала бы варианты написания кода и выделяла бы переменную цветом во всех местах кода при наведении на неё – мне вроде было бы удобнее.

8. Информационные ресурсы

Для изучения основ HTML, CSS, JS я использовал документацию MDN (developer.mozilla.org) на английском языке. Прочёл там JS guide. Иногда обращаюсь к reference. MDN бывает полезен мне по вопросам JS, DOM.

Закреплял понимание JS за счёт сайта learn.javascript.ru. Изучал DOM в основном на этом сайте.

React, Redux, React Redux изучал в основном строго по официальной документации.
Для точной информации о sass, webpack, babel, git использую официальную документацию. Для того, чтобы узнать что-то быстро при нехватке времени, использую некоторые сторонние статьи.

Для того, чтобы быстро получить какую-то структурированную информацию по интересующим вопросам, иногда помогают (например) статьи на habr.com, xakep.ru и готовые ответы на stackoverflow.com.
По вопросам HTML и CSS – приглядел сайт html5book.ru. Если не он, порою помогают некоторые прилично выглядящие справочники с первой страницы поисковой выдачи.

[Какие области знаний мне интересны: например, биология и тому подобное, ”высокие” технологии (в том числе компьютерные).]

9. Обо мне

Полгода работал программистом 1С, но JS по многим параметрам привлёк меня в большей степени. Последние два месяца я (по большому счёту) целыми днями занимался самообучением, чтобы поступить на должность React разработчика. Это привело меня в FunBox.

Помимо данного тестового задания у меня есть единственный, недоделанный проект на ”чистых” HTML, CSS, JS. Я задействовал аудио-DOM event-ы – в результате получил относительно полноценный плеер (хотя не успел сделать настройку уровня громкости и полноценный список для музыкальных композиций – но при этом есть секундомеры и drag полоска, связанная со временем текущей музыкальной композиции). В коде там беспорядок и в публичном доступе проект не находится.