Фронтендеры: джуниоры, мидлы и сеньоры

В программировании сложилась практика разделения разработчиков на уровни Junior, Middle и Senior. Иногда возникают дискуссии про необходимость деления из-за размытости границ, субъективности и т.п. Но в целом это уже сложившаяся общепринятая модель, которая удобна многим — её понимают и требуют заказчики, её придерживаются разработчики, про неё спрашивают кандидаты при устройстве. Это общий язык, который более-менее понимают все. Однако понимают по-разному. В этом документе мы фиксируем наш взгляд.

Джуниор

Джуниору достаточно знаний конкретных технических инструментов (языка, фреймворка и библиотек, IDE), которые позволяют решать локальные задачи, не затрагивающие сразу несколько компонент системы. Если джуниору уже доверяют работу на реальном проекте и платят за это, но он ещё не всеми навыками обладает, то полученная позиция — больше везение, чем заслуга, надо интенсивно подтягиваться и оправдывать выданный кредит доверия.

- Умеет решать задачи с чётко обозначенными границами в рамках одного компонента разрабатываемого продукта, если задачи хорошо декомпозированы, и укладываются в принятые соглашения и архитектурные принципы.
- Хорошо знает основной инструментарий и умеет его применять для стандартных задач: HTML, CSS, JavaScript, DevTools браузеров.
- Уверенно владеет используемыми в команде библиотеками (например, React) и фреймворками (например, Angular) на таком уровне, который позволяет решать абсолютное большинство стандартных задач по созданию интерфейсов (включая настройку сборки, верстку, хождение до API и обратно).
- Умеет следить за качеством — тестировать свою работу, понимать и вносить правки после код-ревью.
- Понимает идею поддерживаемости кода. Может ответить на вопросы: «Что значит код читаемый? Что значит гибкий? Зачем нужны стандарты? И какие особенности по

стандартам в команде?» При этом не обязательно, чтоб он мог сам суметь написать такой код без помощи.

Что покажет на собеседовании классный джуниор?

- Грамотно и гибко верстает, не хардкодит отступы где попало.
- Понимает, что такое замыкания и как ими пользоваться в JavaScript.
- Понимает, как работать с прототипами в JavaScript, в чем отличие от ООП на классах, какие способы отнаследоваться существуют и в чем отличия, и почему их знание необходимо даже с реализацией новых стандартов.
- Владеет применяемым в команде VCS, например, Git, понимает разные механизмы ветвления, пушит осмысленные коммиты, понимает основы git flow.
- Умеет писать unit-тесты, понимает преимущества TDD.

Мидл

Четыре навыка ниже — определяющие, без любого из них мидл — это ещё всё-таки джуниор.

- Умеет использовать инструменты из списка на джуниора даже в нестандартных ситуациях, то есть он их отлично знает.
- Умеет эффективно решать довольно большие задачи в рамках заданной архитектуры, например, следуя парадигме от React+Redux, даже если задачи затрагивают сразу несколько слоёв проекта. Можно смело доверять разработку интерфейсов для проекта до ±100k строк с командой до ±10 участников.
- Проектирует гибкие и легко читаемые решения локальных проблем, под которые нет готовой архитектуры. Например, не просто интерфейс по макетам к API прикрутить, а, написать развесистый модуль по работе с пуш-уведомлениями. Тут React не поможет.
- Понимает ценность своей работы для бизнеса: спрашивает “зачем?”, говорит “это не нужно”, когда это действительно не нужно, и аргументирует возможными альтернативами или даже доказывает низкий приоритет.

Что покажет на собеседовании классный мидл?

- Понимает, что такое event loop и какие ограничения однопоточный рантайм JavaScript-а накладывает на проектируемую программу.
- Владеет ООП хотя бы на базовом уровне: может смоделировать иерархию объектов, декомпозировать сложный стейт, развязать зависимости и разбить ответственность, определить поведение для каждой подчасти, понимает, когда использовать наследование (практически никогда) и почему. Знает SOLID и применяет его.

- Владеет функциональным программированием хотя бы на базовом уровне: умеет работать с иммутабельными данными, декларативно описывать желаемые процессы, использовать функции высшего порядка для полиморфных частей программы.
- Понимает DDD, может создать базовую модель предметной области, необязательно сразу корректную и удобную.
- Знает паттерны от банды четырёх, видит их даже там, где они неявно использованы, умеет проводить аналогии между текущим решением и идеальной моделью из паттерна. Уже преодолел стадию игрищ с паттернами, когда непреодолимо желание использовать их где попало.
- Уверенно владеет unit и интеграционным тестированием, умеет проектировать и писать тестируемый код, знает и понимает пирамиду тестирования.

Сеньор

Расслабьтесь и доверьте продукт сеньору, если он:

- Понимает контекст всего продукта (обширней, чем просто проекта), включая пользователей, заказчиков, команду разработки и тд. Если проект новый для сеньора, то у него уже есть за плечами опыт погружения в прошлые проекты, а на текущем ему можно выдать кредит доверия на быстрое и эффективное изучение.
- Может сам чётко сформулировать задачу даже при смутных вводных либо затребовать конкретные недостающие данные. Разговаривает с нетехнарями доступным языком, донося ценности разработки и тонко улавливая нужды других команд.
- Умеет спроектировать архитектуру больших решений, заложив максимальную гибкость, которая позволит в дальнейшем проводить глубокий рефакторинг без полного паралича разработки. Примеры задач, где это необходимо: сложная библиотека графиков или комплекс из нескольких взаимосвязанных проектов.
- Сам строит себе карту развития. Умеет быстро изучать темы, необходимые для решения задач. Например, может изучить, как парсить и анализировать текст какого-нибудь языка разметки, и написать такой парсер.

Что покажет на собеседовании сеньор?

- Владеет ООП на продвинутом уровне: умеет уверенно смоделировать расширяемую и поддерживаемую иерархию сущностей, выделять контракты и ядро компонент.
- Владеет ФП на продвинутом уровне: знает слабые и сильные стороны использования монад, может писать элегантный и читаемый код, не злоупотребляя возможностями языка и парадигмы, может создать удобный интерфейс модуля в комбинаторном стиле.

- Уверенно владеет DDD, знает (и скорее всего прочувствовал) грабли, наподобие anemic domain models, умеет вытаскивать ценную информацию из экспертов предметной области.
- Знает множество паттернов, далеко не только от банды четырёх, тонко чувствует их схожести и различия, видит проект как иерархию паттернов.
- Умеет задизайнить архитектуру большого и сложного проекта, используя сильные стороны и избегая слабые у уже известных архитектур и принципов разделения ответственности: Порты и Адаптеры, Луковая Архитектура, основанная на сообщениях асинхронная архитектура (asynchronous message-based architecture), иммутабельная реактивная парадигма и т.п.
- Знает, как спроектировать систему автоматического тестирования и доставки оттестированного кода до продакшена, как быстро детектить и логировать ошибки на продакшене, чтобы можно было быстро пофиксить даже сложные кейсы с конкурентностью, например.

Об авторе

Меня зовут Сергей Черепанов, я руковожу командой фронтенд-аутсорсеров Fullstack Development с ноября 2014-го, за это время я обучил десятки ребят, разработал масштабируемую систему изучения фронтенда, создал карту развития, которая подробно описывает, как вырасти из джуна до сеньора (у нас это очень долго и больно).

Также я CTO сервиса срочной курьерской доставки Ptichka.moscow. Я активно прокачал знания бекенда от зачаточного до сносного уровня и изучил ещё два языка: Elixir и Haskell, помимо уже известных мне JavaScript и Python.

Заключение

Лучшие выделяются среди хороших тем, что:

- Умеют проектировать решение, где нет стандартных подходов (куда React не прикрутишь, например).
- Знают и применяют практики, которые известны отрасли уже 50 лет, но которые фронтенд отрицает или переоткрывает для себя заново.
- Закапываются в бизнес по-максимуму. Часто лучшие фронтендеры знают продукт лучше всех в команде (конкурируя тут разве что с тестировщиками).

У всех хороших есть потенциал стать лучшими. Развивайтесь и развивайте других.

Как развивать документ дальше?

Многие пункты остались не освещенными, многие пункты раскрыты, но недостаточно, где-то мы установили чётких критериев. Расти ещё есть куда, и здесь я был бы рад увидеть вашу помощь. Ниже список хотелок, которые я буду встраивать в документ, активно встраивайте туда свои пункты (у Google Docs замечательная функция Suggest Changes) или комментируйте текущие.

- Внести описание требований к soft skills. Я не уделял им внимание в документе не потому что мы их не ценим, а потому что я и так вижу у многих явное смещение в оценку этих навыков, а не hard skills. Когда я помогал нанимать или просто расспрашивал о текущих системах найма в разных компаниях, все 100% указывали, что оценивают soft skills, и почти все это делают более-менее одинаково. На что почти всегда и так смотрят:
    - коммуникации: умеет донести свою мысль, умеет не перебивать,
    - планирование: умеет держать свой график под контролем, контролирует выполнение плана.
    - работа в команде: уточнять у тимлида, может ли он помочь другой команде, если его попросили, например, может сам запросить подобную помощь.
    - эмпатия: умеет выслушать и понять, ловко выходить из конфликтов, не плодя новых.
    - обучаемость: умеет самостоятельно разобраться в материале для своего уровня, не останавливается в развитии, даже когда всё уже вроде хорошо.
    - Взаимодействие с остальной командой: с бекендерами, мобильщиками, ПМ, аналитиками и тд.
- Понимание статической типизации, её плюсов и минусов, если всё-таки есть приверженность к динамической, то она должна быть осознанной.
- Ненужность паттернов. Паттерны — очень противоречивая тема в software engineering. Я придерживаюсь позиции, которая хорошо себя показала в нашей практике: знать паттерны стоит, применять их нужно с осознанием, пихать их где попало не стоит. Знание помогает при проектировании нетривиальных частей программы, поэтому оно необходимо уже для мидлов (которые должны локальные жирные решения тоже уметь с нуля делать), причем их не обязательно активно применять, помогает сам факт их знания, умение видеть даже в мешанине проглядывающие структуры, которые можно выделять и упрощать для себя модель программы. Теоретическое знание паттернов — это путь на плечи гигантов, тут еще до конца сообщество не выбрало, стоит ли туда лезть.