Во время интервью стоит записывать и помечать как кандидат отвечал на те или иные вопросы. Где у кандидата возникали проблемы, а где он справился хорошо.

Вопросы стараются определить крепкие базовые знания у К .

1. **Ревью Pull-Request на React, Typescript.**

Даем К на проверку pull-request. Pull-request небольшой и достаточно простой для понимания. Цель К найти как можно больше ошибок. Хорошо если К будет анализировать вслух, это поможет понять его ход мысли, и как он подходит к анализу.

**Pull-request который нужно давать К:**

<https://gitlab.com/TheNear/test-review/-/merge_requests/1/diffs>

**Pull-request для самопроверки с комментариями:**

<https://github.com/TheNear/review-test/pull/1/files>

**На что К должен обратить внимание**

- Мемоизация компонента в необходимом месте

- Утечка памяти из-за слушателя

- Порядок вызова хуков

- Использования индекса в качестве ключа

- Обработка ошибок запроса

- Аборт запроса при анмаунте компонента

- Валидация инпута

- Доступность и семантика

**Что мы проверяем этим вопросом и почему это нам важно:**

Этим вопросом мы проверяем базовые знания по React и Javascript одновременно. Немного задеваем Typescript. Нам это дает понимание, знает ли кандидат как работать с React и его опыт. Нам это важно, потому-что большинство задач К будет состоять в написании компонентов, наличие этих знаний скажет нам о том, сможет он быстро и самостоятельно делать типовые задачи.

1. **Простая и быстрая проверка знаний Typescript**

Даём такой кусок кода, можно использовать<https://www.typescriptlang.org/play> . Нужно подготовить для этого комнату.

const obj1 = { name: "Alice", age: 25 };

const obj2 = { job: "Developer", country: "Canada" };

const mergeObjects = (obj1, obj2) => {

return {};

}

const result = mergeObjects(obj1, obj2);

Просим реализовать типизированный вариант функции поверхностного мержа двух объектов. Это даст нам понимание о знании К дженериков.

Ожидаем такого примерно такого решения:

const obj1 = { name: "Alice", age: 25 };

const obj2 = { job: "Developer", country: "Canada" };

const mergeObjects = <T extends Record<string, any>, U extends Record<string, any>>(obj1: T, obj2: U): T & U => {

return {...obj1, ...obj2};

}

const result = mergeObjects(obj1, obj2);

**Маркеры вопросов и замечаний:**

* На вход приходят только объекты, нужно чтобы тайпскриптом это проверялась.
* Объекты могут имплементировать любой интерфейс, нам важно чтобы на выходе было объединение по типам, вне зависимости о того что приходит на вход, функция должна быть универсальной.
* Если кандидат очень легко справился, можно спросить дополнительно, будет ли работать эта функция с объектами, которые имеют вложенную структуру, и будет ли работать вывод типов.

**Что мы проверяем этим вопросом и почему это нам важно:**

Эти вопросом мы немного углубляем знания по TypeScript, но все равно пробегаем всего лишь поверхностно. Это базовые знания, кандидат должен понимать как работают Дженерики в TypeScript, иначе он будет тратить очень много времени на типовые задачи.

1. **Проверка знаний оптимизации Веб-приложений и проблем с производительностью.**

**Вопрос:** Пользователь жалуется что долго грузиться сайт, что делал бы в таком случае? Куда бы смотрел?

Из-за чего могут быть такие проблемы?

**Что хотим услышать:**

* **Анализ проблемы:** К должен рассказать про инструменты профайлинга, такие как Chrome DevTools, Lighthouse, и WebPageTest, для проведения начального анализа производительности. Он должен посмотреть на обращение к API, посмотреть как они проходят, посмотреть размеры файлов JS, Картинок и CSS. Они могут быть большие. API может отвечать долго, что он в этом случае бы делал? Хотим услышать от него рассуждение по этому вопросу. Если все эти аспекты в норм, куда бы еще он мог посмотреть. Может какой-то клиентский код выполняется долго?
* **Пути решения:**
* **a)** Оптимизация ресурсов - Минификация CSS, JavaScript и HTML файлов, а также их объединение, где это возможно. Splitchunking кода с помощью сборщиков. Знает ли К про способы минификации CSS, Javascript ( Babel и плагины ). Lazy Loading компонентов и ленивый импорт, чтобы не загружать сразу весь код приложения, а только тот, который нам необходим для отрисовки страницы. Может можно спросить про модульность в JS, в чем отличие ES modules от Common JS. Что такое threeshaking и тип модулей влияет на это.В части изображение можно спросить про использование современных форматов, таких как WebP, и подходов, включая "ленивую" загрузку и адаптивные изображения.
* **b)** Кеширование ресурсов - Нужно чтобы рассказал про Браузерное кеширование. Какие есть способы браузерного кеширования. Знает ли К про заголовки Cache-Control, Expires, ETag. Что он может рассказать про Cache-Control? Как установить время жизни кеша (max-age), как отключить кеш (no-store). Про заголовок Expires, для чего он (Можно задать точную дату инвалидации кеша)? Как еще можно инвалидировать кеш, какие способы К знает (Например версионирование статика, вследствии чего меняется ссылка на файл)? Для чего заголовок ETag (Позволяет инвалидировать кеш при изменении содержимого, обеспечивается проверкой Хэша в заголовке ETag)?
* **b)** Медленная работа API - больше разговорная часть? Чтобы проверил, и что делал бы в таком случае

**Что мы проверяем этим вопросом и почему это нам важно:**

Этим вопросом мы проверяем базовые знание по тому как устроена сборка веб-приложения, как она отдается пользователю, и как устранить возможные проблемы плохой оптимизации. Нам это важно в качестве базового понимания К устройства веб-приложений, как он находит анализирует и решает проблемы во время их возникновения. Его знания и опыт работы со сборщиками и подобными инструментами.

1. **Небольшая задача на проектирование RESTful API**

**Вопрос:** ”Представь что ты вместе с бекенд разработчиком делаете простое приложение. Приложение представляет собой обычный Task tracker, по другому тудулист. Ты можешь получить весь список задачь, добавить новую задачу, удалить задачу, изменить одну из задач. Допустим это всё что ты можешь сделать. Как бы ты реализовывал API или по другому контракт с бекенд разработчиком.”

Попросить K просто описать как на его взгляд выглядели бы ручки/эндпоинты, чем подробнее тем лучше. В формате:

**GET**: /tasks

**POST**: /task

**DELETE**: /task/:id

**PUT**: /task/:id

Прямо примерно в таком формате и просим описать. Потом просим, чтобы он описал, какие ручки какие данные принимали бы, и какие отдавали бы.

Например

**GET**: /tasks?limit=0&offset=15 => Array<Задач>

**POST** /task в body отсылает допустим {title: ‘new task’, content: ‘task content’ } => Возвращает созданную таску

**DELETE** /task/:id => Возвращает OK 200

**PUT** /task/:id в body отсылает {title: ‘new title’} => Возвращает измененную таску

Тут можно какой опыт кандидат имеет при взаимодействии с RESTAPI.

Просим его рассказать чтобы он еще добавил. Какие-то может оптимизации бы добавил. Например для ручки получения тасок он упомянет пагинацию.

Можем обратить внимание на используемые методы, и спросить почему он решил использовать именно эти методы для этих задач. Спросить про Кеширование, если на прошлом вопросе мы не спросили.

Можем задать вопрос “А как приложение поймет какие задачи именно отдавать, если пользователей несколько”. Этот вопрос должен подтолкнуть К на аутентификацию и идентификацию пользователя сервером. Как бы изменились его запросы? Может он добавить заголовок Authorization. Можно спросить какие методы авторизации он знает.(Basic когда логин и пароль передаются в заголовке в base64 с каждым запросом с использованием https или может с помощью JWT токенов, когда на основе данных формируется токен, а в более продвинутой версии минимум 2 токена Access и Refresh, или например OAuth )

**Что мы проверяем этим вопросом и почему это нам важно:**

В целом это очень маленькая задача, которая поможет понять насколько К знаком с работой и может с проектированием API в базовом его понимании. Это поможет в дальнейшем К при взаимодействии с другими коллегами кросс функциональной команды. Очень часто К на самом деле сильно тупят на таких простых вопросах, что позволяет увидеть очень сильные пробелы в понимании базовых вещей.

1. **Проверка знаний протокола HTTP и безопасности Веб-приложений.**

**Первая часть вопроса:** Можешь рассказать про HTTP методы, какие знаешь и для чего нужны, в чем отличие?

**Ожидаем примерный ответ:** Ожидаем что К знает основные методы GET, POST, PUT, PATCH, DELETE, OPTIONS, HEAD. Будет плюсом если К знает про идемпотентность методов и какие из этих методов являются “безопасными”, а какие нет. ( GET, HEAD, PUT, DELETE, OPTIONS - являются идемпотентными, так как при многократном вызове возвращают тот же ответ. POST, PATCH - в большинстве могут быть не идемпотентными, зависит от реализации ). Спрашиваем у кандидата про метод OPTIONS, знает ли он для чего используется данный метод (Для определения возможностей сервера, и важный инструмент в контексте CORS )? В каких случаях клиент посылает запрос OPTIONS?( Если используется методы отличный от GET, HEAD, POST. При POST так же делает автоматический префлайт запрос OPTIONS, если content-type выставлен отличный от application/x-www-form-urlencoded, multipart/form-data, или text/plain. Так же автоматический будет если в запросе присутствуют любые пользовательские заголовки, которые не разрешает CORS по умолчанию ) Здесь мы можем перейти ко второй части вопроса.

**Вторая часть вопроса:** Можешь рассказать про CORS, и про другие возможные уязвимости веб-приложение, и какие способы их избежать.

**Ожидаем примерный ответ:** CORS защищает от междоменного доступа к данным.( Пример уязвимости: Атакующий создает вредоносную веб-страницу evil.com, которая выполняет запрос к example.com/api/private с использованием учетных данных пользователя. Поскольку CORS неправильно настроен на example.com, браузер пользователя разрешит этот кросс-доменный запрос, потенциально раскрывая чувствительные данные атакующему. ) Как работает CORS? ( Браузер делает префлайт запрос OPTIONS, чтобы узнать у сервера какие методы доступны на сервере, другая информация, и в том числе какая политика CORS, доступ с каких ресурсов разрешен. Если в ответе придет заголовок Access-Control-Allow-Origin, в котором не будет указан домен, с которого выполняется запрос, то браузер автоматически заблокирует запрос. Важно! Понимает ли К, что CORS это не про защиту API и Сервера. Никто не мешает подделать запрос злоумышленником или отключить проверку CORS в браузере). Какие заголовки вообще используются для CORS (Access-Control-Allow-Origin, - какие доменны доступны, Access-Control-Allow-Methods - Какие методы доступны, Access-Control-Allow-Headers - Какие заголовки доступны, Access-Control-Allow-Credentials - Передавать ли куки в кросдоменном запросе ). **ВАЖНО: Нам не нужен дословный ответ наизусть, нам просто важно понимание К данного вопроса.** XSS уязвимости - Пример когда ты вставляешь себе на сайт какой-то сторонний кусок кода, может из библиотеки, а он пересылает твои креденшелы на сторонний ресурс. Можно задать CSP, и указать куда именно можно делать запросы с своего ресурса. Еще важно делать санитайз пользовательского ввода например, если используется какой-нибудь dangerInnerHTML. Если уж совсем хочется углубиться, можно спросить про SQL инъекции, или как обеспечивает безопасность HTTPS

**Что мы проверяем этим вопросом и почему это нам важно:**

Эти вопросы проверять самое базовое представление о взаимодействии клиента и сервера например. Самые базовые вопросы безопасности. То что человек не будет весь день сидеть и думать почему у него CORS блокирует. Ну и то, что он осознает когда пишет код, чтобы не наплодить уязвимостей.

1. **Дополнительные вопросы по стекам технологий:**

**Вопросы:** Нам желательно чтобы К умел работать с Redux, Redux-saga в частности, это ускорит его как выполнителя задач. Работал ли он с Webpack. Может К настраивал Webpack или писал какие-то плагины. Писал ли К тесты, с помощью чего и какие тесты писал К.

**Что мы проверяем этим вопросом и почему это нам важно:**

Эти вопросы проверят какой опыт имеет К прим работе с технологиями, которые используются у нас. Чем больше пересечений, тем быстрее К начнет приносить пользу команде, и тем меньше придется его менторить. В этих вопрос нужно смотреть на реакцию К, и задавать вопросы немного в глубь. Так как в таким абстрактных вопросах К ничего не мешает приврать и приувеличить.

1. **Дополнительные вопросы на определение сложности:**

Простой быстрый тест на определение сложности алгоритмов, без конкретных задач.

const users = [{id: 1, name: 'Vova', admin: true }, {id: 2, name: 'Pasha'}, {id: 3, name: 'Georgiy'}];

const userMap = {

1: {id: 1, name: 'Vova'},

2: {id: 2, name: 'Pasha'},

3: {id: 3, name: 'Georgiy'}

}

users.sort((a,b) => a.id - b.id); // Сложность // 1

users.find((user) => user.id === 3); // Сложность // 2

userMap[3] // Сложность // 3

users.reduce((acc, cur) => {

if (cur.admin) {

acc = [...acc, cur];

}

return acc;

}, []); // Сложность // 4

Просим К определить сложность.

Ожидаем услышать:

1. O(n\*logn) - Линейно-логарифмическая сложность. Такую сложность имеет метод быстрой сортировки в JS
2. O(n) - Линейная сложность. Так как придется пройти по всем массиву для поиска.
3. O(1) - Константная сложность.
4. O(n2) - Квадратичная сложность. Так как в цикле используется еще один цикл. Спред оператор, это цикл. Чтобы развернуть массив в массив, он по одному элементы разворачивает каждый элемент массива.

**Что мы проверяем этим вопросом и почему это нам важно:**

Проверяем самые базовые понимания алгоритмов и сложностей. Это покажет нам, что К осознает сложность и как выполняется его код, и не напишет неоптимизированный код, который приведет к лагами и задержка в интерфейсе.