Чеклист для самопроверки. 1 Спринт.

В этом документе описаны критерии, которым должна соответствовать работа. Убедитесь, что ваша работа соответствует им. И только после этого отправляйте на ревью.

Проверьте

1. Все прототипы нарисованы и добавлены в папку `ui` или есть ссылка на макет в Figma.
2. Подключён шаблонизатор.
3. Настроена сборка проекта при помощи Parcel.
4. Все страницы сделаны на шаблонах. Страница со списком чатов и лентой переписки сделана как страница-заглушка (реализуете её в следующем спринте).
5. Стили написаны с использованием препроцессора или подключён PostCSS.
6. Весь код в проекте разбит на модули (`import` или IIFE — неважно, но должно быть что-то одно).
7. Настроена локальная раздача статики через Express.
8. Проект выложен на Netlify, настроен автодеплой.
9. Из `README.md` удалён текст, сгенерированный по умолчанию при создании репозитория.
10. Репозиторий оформлен, то есть в `README.md` добавлено описание проекта и указаны команды, которые можно использовать (как минимум для сборки проекта).
11. В ветке `main` лежат только исходники.
12. Проект открывается на 3000 порту, команда для запуска проекта — `npm run start`.
13. Репозиторий с работой публичный.
14. Папки `dist` и `node_modules` добавлены в `.gitignore`.

Работа не принимается

1. Нет нарисованных прототипов.
2. Не подключён Parcel, проект не собирается или собирается с ошибками.
3. Стили написаны не на препроцессоре или не используется PostCSS.
4. Все стили внутри `*.html`.
5. Стили написаны с большим дублированием, без применения методологий (можно использовать, например, БЭМ). Именования классов в тегах не декларативны и непонятно, что описывают.
6. Не используются семантические теги:
   - основной контент страницы не обёрнут в тег `main`,
   - шапка и меню сделаны просто через `div`, без `nav`,
   - кнопки созданы через тег `a`,
   - списки также делаются не через `ul` и `li`,
   - заголовки не обёрнуты в нужные уровни.
7. Используется где-то IIFE, где-то модули ES6. Нужно выбрать что-то одно.
8. Неэффективная работа с DOM-деревом. Например, циклы, которые добавляют в DOM много элементов или вставка и удаление элементов там, где можно обойтись `display: block` и `display: none`.
9. Контент везде добавлен в элемент через `innerHTML`. Необходимо стараться использовать `textContent`.
10. Не настроена ветка `deploy` для автодеплоя в Netlify.
11. Express-сервер локально не раздаёт статику.
12. Не указана версия NodeJS в `.nvmrc` или `engine` в `package.json` либо указана, но ниже 12.

13. Не оформлен `README.md`. Нет описания проекта, не указаны команды для запуска проекта (как минимум для сборки).
14. Остался текст шаблонного `README.md`, который сгенерировался автоматически.
15. В ветке `main` лежит собранный проект, то есть папка со сборкой не в `.gitignore`. На проверку нужно отправлять только исходники.
16. Проект запускается не на 3000 порту, команда для запуска не `npm run start`.
17. Репозиторий с работой приватный.

Рекомендации, которые не стоит игнорировать

1. Самая сложная страница — это непосредственно чат. Советуем начать с него, а напоследок оставить что-то простое (например, страницу ошибки 404).
2. Если сами разрабатываете дизайн:
   - Старайтесь максимально переиспользовать элементы. Например, вы сэкономите время на написании стилей, если поля ввода на всех страницах будут выглядеть одинаково.
   - Не переусердствуйте с разнообразием размеров шрифта и цветов. Так вам будет проще их применять и не придется помнить множество вариантов, даже если вы используете препроцессор и вынесли их в отдельные переменные.
3. Не дублируйте стили. Если понимаете, что они повторяются, лучше вынести их в отдельный файл или использовать силу препроцессора.
4. Не экономьте буквы, когда называете переменную. Она должна быть понятна в первую очередь человеку, а не компьютеру. Последний всё «стерпит». Название должно максимально отражать суть, быть специфическим для конкретного случая и легко читаться.
5. Не оставляйте код «на будущее». Он может не пригодиться, а проект превратится в «свалку».
6. Не храните в коде комментарии с бесполезным текстом.
7. Оформляйте условия полностью. Да, запись в одну строчку сэкономит место, но ваш код будут смотреть и другие разработчики. От таких сокращений пострадает в первую очередь читаемость кода.

```
// Хорошо
if (name === 'Серёжа') {
  console.log(`${name} молодец!`);
}

//А так лучше не делать
if (name === 'Серёжа') console.log(`${name} молодец!`);
```

8. Старайтесь избегать излишней вложенности кода. Если есть возможность вынести что-то в отдельный блок, лучше это сделать. Особенно это касается функций со сложной логикой.
9. Единообразие. Единообразие в каждом файле. Ваш первый файл должен стать началом тому стилю, которого вы будете дальше придерживаться. Начали использовать двойные кавычки? Во всех файлах должны быть двойные кавычки. Начали ставить пробел после `if`? Значит, так должно быть везде. Когда в следующих спринтах добавите в проект линтеры (инструменты, которые будут следить за чистотой кода и стилем), придерживаться единообразия станет намного проще. Но и без них код должен выглядеть аккуратно.

```
// Пример того, как не надо делать. Представьте, что это два файла из одного проекта

//file1.js
const name = 'Ivan';
function sayHello(name) => {`Привет, ${name}`};

//file2.js
const nickName="DoMiNaTor98"

function sayGoodbye (nickname)=>{
  return `Пока, ${nickName}`
}
```

10. Будьте внимательны, если копируете код из решения в тренажёре или из урока. Убрали ли вы лишние комментарии? Исправили ли решение так, чтобы оно учитывало все случаи, которые могут у вас возникнуть? Добавили ли туда необходимый для вашего проекта функционал? Иногда в коде из уроков или задач есть рекомендации, как можно улучшить решение. Не игнорируйте их. Они помогут сделать проект ещё лучше.
11. Не используйте в проектной работе `var`. Совсем. После появления `let` и `const` его чаще всего можно встретить в legacy-коде (то есть старом, но ещё существующем по каким-то причинам в неизменённом виде). От каких ошибок вас защитят `let` и `const`, найдёте в уроке о переменных в первом спринте.
12. Ваш код будут читать другие люди. Иногда логика функции или класса не очевидна, но важна. Старайтесь в таких местах оставлять короткие комментарии с объяснениями. Но знайте меру. Не нужно описывать каждый шаг или общую логику, если они могут быть понятны с первого взгляда.
13. Обрабатывайте исключения. Не просто через `try ... catch` и общее сообщение об ошибке `Error` или `console.log`. Нужно генерировать максимально специфические исключения, насколько это возможно.
14. Старайтесь не мутировать данные, это ухудшает читаемость кода. Пример:

```
// Было
let someData = 'foo';
// Какая-то логика
someData = 'result'

//Стало
const someData = 'foo';
// Какая-то логика
const result = 'result'
```

15. Если много условий, их лучше инкапсулировать в одну переменную. Пример:

```
// Было
if (isActive && userAvailable() && staffOnly && getTime()) {
  // Какая-то логика
}

// Стало
const isProcessAvailable = isActive && userAvailable() && staffOnly && getTime();

if (isProcessAvailable) {
  // Какая-то логика
}
```

16. Читаемый код лучше, чем минималистический. Пара сэкономленных строчек могут стоить множества потраченных нервов. Как тем, кто будет этот код читать, так и вам через пару месяцев.
17. Если в проекте есть картинки, лучше брать их из фотостоков (например, Unsplash или Pixabay).
18. Вы не ограничены в выборе иконок и шрифтов. Добавить их можно как ссылками, так и напрямую в проект, скачав нужные файлы. Важно, чтоб они были бесплатными и не защищались авторскими правами. Иконки можно найти, например, здесь: Font Awesome, icon8, Flaticon, Material Design. Со шрифтами поможет Google Fonts.
19. Желательно хранить `index.html` в папке `static`. Это упростит настройку Netlify.