Чек-лист по самопроверке №5

В этом документе описаны критерии, которым должна соответствовать работа. Перед отправкой работы на ревью, убедитесь, что она соответствует всем критериям.

Работа отклоняется от проверки

Если не соблюсти хотя бы один из критериев этого блока, ревьюеры не станут проверять работу.

- Пул-реквест не отправлен на проверку.
- Проект не собирается или не запускается.

Работа принимается

В этом блоке собраны требования к итоговой работе, по которым вы можете выявить и самостоятельно исправить частые ошибки:

Общее

- Функциональность, требуемая по заданию предыдущей проектной работы, реализована корректно и продолжает работать. Её описание вы можете найти **в чек-листе.**
- Пул-реквест создан из ветки `sprint-5/step-1` в `main`.
- Сборка и запуск проекта выполняются без ошибок.
- В проекте есть:
  - директория `components` с компонентами и стилями: `App`, `AppHeader`, `BurgerIngredients` , `BurgerConstructor`, `Modal`, `ModalOverlay`, `OrderDetails`, `IngredientDetails`;
  - если в проекте используются сторонние шрифты или изображения, то они хранятся в папках `fonts` и `images`;
  - единая директория `services` с вложенными `actions` и `reducers` со всей логикой приложения, либо `services` с `actions` и `reducers` для каждого отдельного компонента;
  - структура redux-части проекта для работы с `WebSocket`;
  - директория `pages` со всеми страницами, которые описаны в задании;
  - файл `README.md`;
  - файл `.gitignore`.
- Стили портированы как модули. Если есть общие стили, они портированы в глобальную область видимости.
- Код оформлен без ошибок:
  - имена переменных и функций написаны в camelCase;
  - имена классов и функциональных компонент — существительные с прописной буквы;
  - имена переменных — существительные;
  - имя функции отражает то, что она делает.

  Для именования запрещены:
  - транслит;
  - неуместные сокращения.
- Корректно выполняются запросы к API:
  - URL-адрес домена вынесен в отдельную константу;
  - есть проверка успешности выполнения запроса;
  - цепочка обработки промисов завершается блоком `catch`.

React

- Разметка портирована в JSX:
  - разметка заключена в `( )`;
  - разметка вынесена в соответствующие ей компоненты.
- Компоненты написаны корректно:
  - хуки не используются внутри условных блоков;
  - хуки вызываются в основной функции компонента;
  - при использовании классовых компонентов эффекты описаны внутри методов жизненного цикла компонента.
- Функциональность из брифа реализована корректно:

- Экран «Лента заказов»:
  - отображается по маршруту `/feed`;
  - доступен всем пользователям;
  - обновляется в режиме реального времени, когда авторизованный пользователь создаёт заказ;
  - содержит подсчёт стоимости каждого заказа, исходя из стоимости его ингредиентов;
  - отображает количество всех заказов, а также количество заказов за день;
  - содержит свёрстанные колонки «Готово» и «В работе» с номерами заказов. Разделение между колонками производится на основе значения поля `status` каждого заказа. Каждая из этих колонок содержит в себе не более 10 записей, иначе — создаётся дополнительная колонка.
- Экран «История заказов»:
  - отображается по маршруту `/profile/orders/`
  - доступен только авторизованным пользователям;
  - обновляется в режиме реального времени и отображает статусы заказов;
  - содержит подсчёт стоимости каждого заказа, исходя из стоимости его ингредиентов;
  - статусы «Отменён», «Готовится» и «Выполнен» формируются в зависимости от значения поля `status` каждого заказа.
    - При клике по заказу открывается модальное окно с информацией о заказе и происходит переход на маршрут `/feed/:id` или `/profile/orders/:id` в зависимости от того, на какой странице пользователь кликнул по заказу. При прямом переходе на эти маршруты открывается страница заказа.
    - Маршрут `/profile/orders/:id` доступен только авторизированным пользователям.
- Работа с WebSocket выполняется верно:
  - Сокет-соединение открывается, когда пользователь переходит на экраны «Лента заказов» или «История заказов», и закрывается, когда пользователь покидает их.
  - Логика подключения к сокет-соединению, получение сообщений с помощью него и обработка ошибок реализована с использованием библиотеки `Redux`.
  - Написан middleware, который принимает на вход параметр URL сокет-соединения и содержит набор обработчиков событий: `onmessage`, `onclose`, `onerror`, `send`.
  - Для каждого типа события описаны экшены.
  - Для работы с WebSocket описано базовое состояние Store и создан редьюсер.
  - Конфигурация Store и подключение middleware реализованы верно.
  - При подключении к URL добавляется `accessToken` в query-параметр: `` `?token=${accessToken}` ``.
- Типизация реализована верно:
  - Установлены все необходимые пакеты для `TypeScript`. Конфигурация `tsconfig.json` настроена верно.
  - `any` не используется без необходимости.
  - Инлайновое отключение линтера не используется без необходимости.
  - Для описания структуры объектов используется `interface` или `type`.
  - Описания одинаковых типов не дублируются.
  - Типизированы все утилитарные функции, например функции запросов к серверу. Все файлы с утилитарными функциями имеют расширение `.ts`.
  - Типизация компонентов реализована верно:
    - Все файлы с компонентами имеют расширение `.tsx`.
    - Для компонентов используется `TypeScript` вместо `PropTypes`.
    - Для функциональных компонентов типизируются пропсы и хуки.
    - Для классовых компонентов типизируются пропсы, параметры конструктора, состояние компонента, аргументы методов жизненного цикла.
  - Типизация хранилища реализована верно:
    - Все файлы, относящиеся к хранилищу, имеют расширение `.ts`.
    - Каждый экшен типизирован с помощью литерального типа.
    - Для объединения типов экшенов используется тип Union.
    - Типизированы начальные состояния и сами редьюсеры.

- Тип для хранилища описан с помощью состояния хранилища или вспомогательного типа `ReturnType`.
- Описана типизация для Redux Thunk. Для типизации метода `dispatch` используется `typeof`, либо тип `Dispatch` из пакета `redux`.
- Типизированы хуки `useDispatch` и `useSelector`.