УНИВЕРСИТЕТ СИНЕРГИЯ

ДОМАШНЕЕ ЗАДАНИЕ

Тема 4. Использование TypeScript с современными фреймворками и библиотеками

Цель работы: Отработать на практике навыки написания React-компонента на Typescript и тестов

План работы:

1. На основе кода урока “Написание Unit-тестов” создать страницу регистрации, содержащую следующие поля: Login, Email, Password
2. Создать API-роут методом POST для эндпоинта `/api/auth/register`, который принимает следующие поля в теле запроса `{login, email, password }`, и отправляет данное тело обратно на фронтенд со статусом 200. Серверную логику регистрации можно не писать.
3. Страница регистрации должна переиспользовать layout и компоненты совместно со страницей логина
4. Добавить ссылку на страницу регистрации в Header приложения
5. Сделать все компоненты и хуки типизированными
6. Написать тест к странице регистрации, в котором производится snapshot-тестирование страницы
7. [Задача со звездочкой *] Написать валидацию к странице регистрации с помощью библиотеки simple-react-validator, где
   a. поле login - required (обязательное)
   b. поле email - required|email (обязательное и валидный емэйл)
   c. поле password - required|min:8 (обязательное и минимум 8 символов)

   Сообщения под полями ввода должны появляться по событию onBlur. Состояние кнопки должно быть disabled (серый цвет, не кликабельная) до тех пор, пока все поля не будут корректно заполнены.

8. [Задача со звездочкой *] Написать тест к странице регистрации, в котором проводится тестирование функционала валидации: Берется Login input и эмулируется собитие Blur, проверяется что текст ошибки присутствует в документе.

```
expect(screen.getByText('The login field is
required.')).toBeInTheDocument()
```

Затем эмулируется ввод любого текста в Login input и проверяется, что текст ошибки более не присутствует в документе

```
expect(screen.queryByText('The login field is
required.')).not.toBeInTheDocument()
```

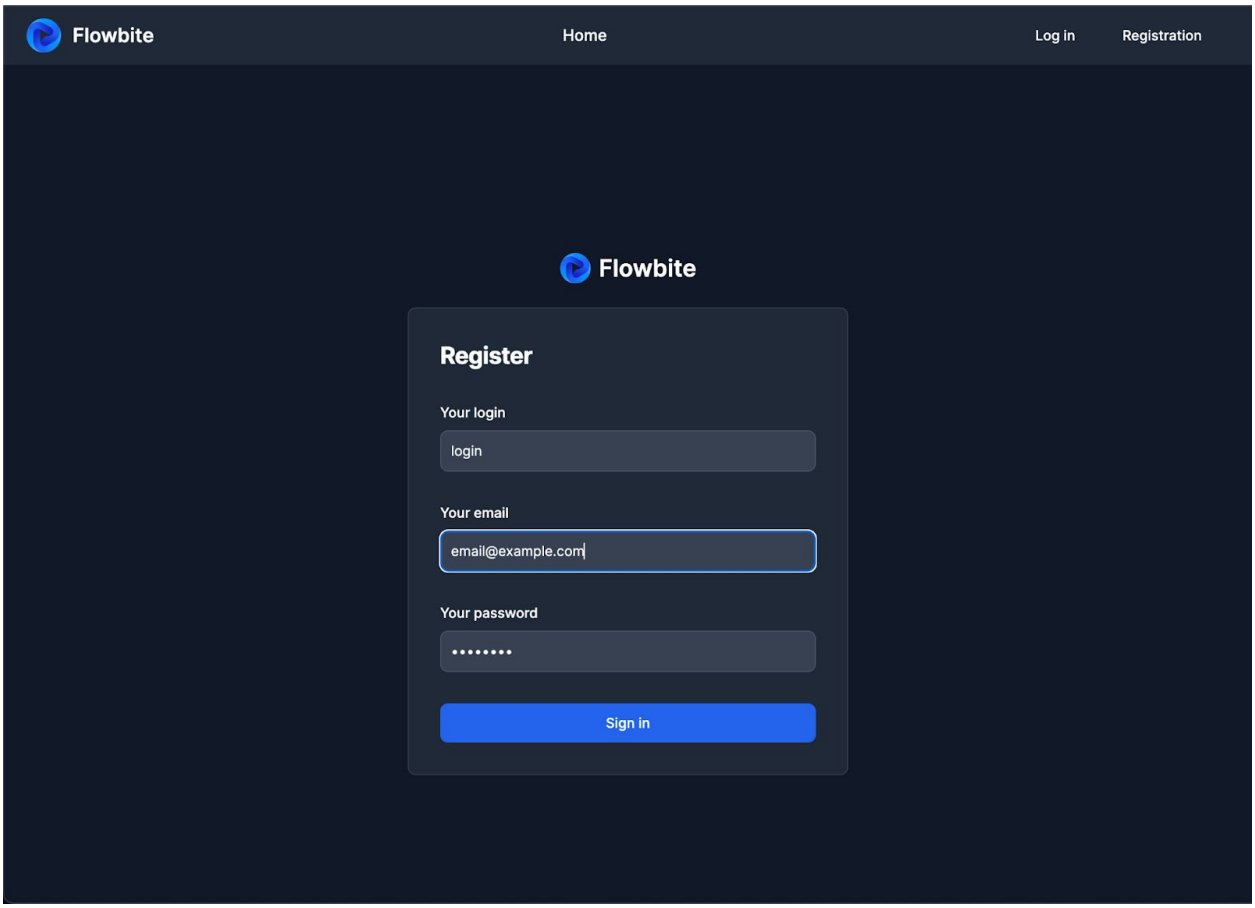

Рисунок 1. Страница регистрации

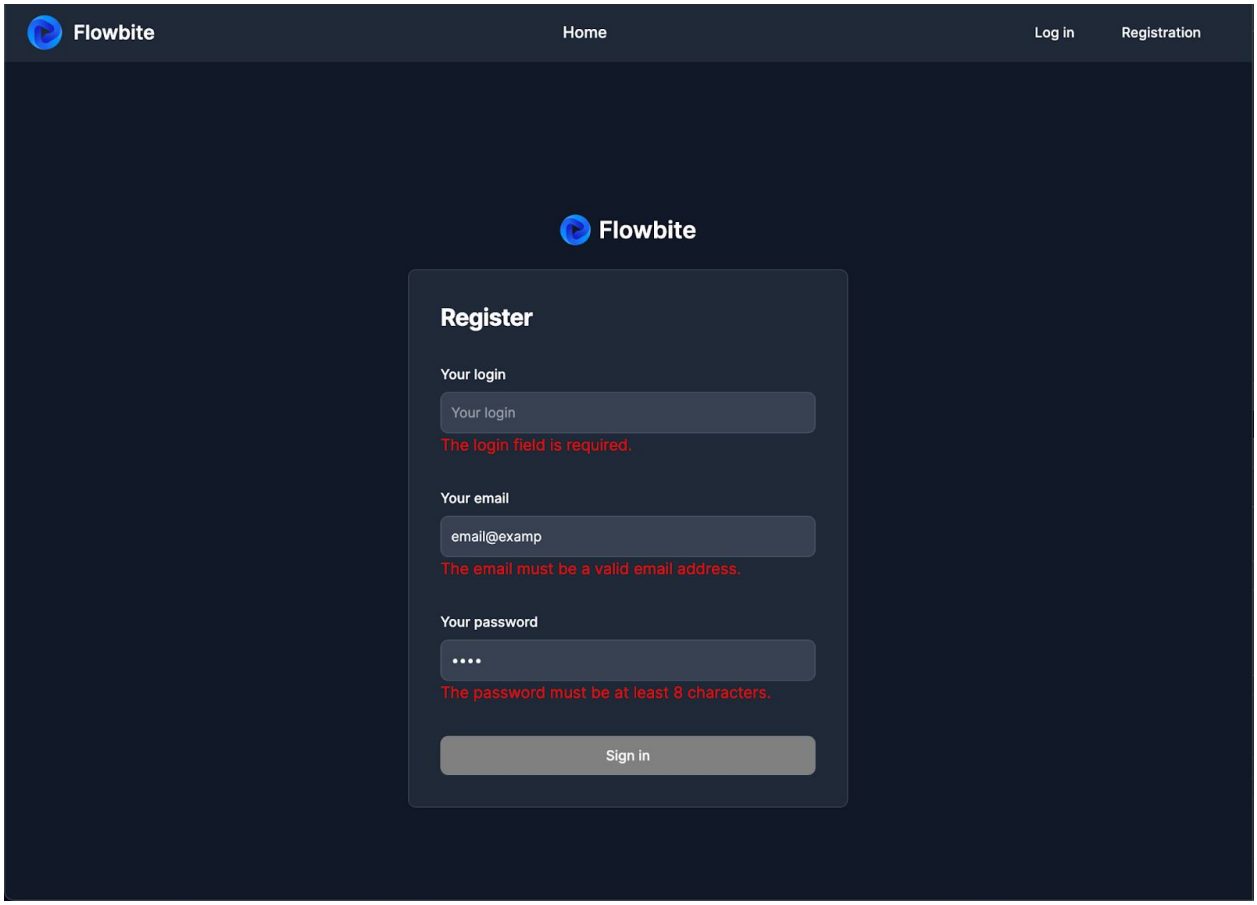

Рисунок 2. Страница регистрации с ошибками валидации

Рекомендации для выполнения:

1. Для правильного подключения валидатора к функциональному компоненту, нужно будет немного переделать примеры из официальной документации, использующие классовые компоненты и контекст классовых компонентов this
2. Подключайте экземпляр валидатора в общий компонент страницы Registration и проверяйте что все поля валидны там. Передавайте в компонент кнопки сразу true/false значение disabled, полученное из валидатора
   `<Button disabled={!validator.allValid()} text="Sign in" />`
3. Передавайте экземпляр валидатора в каждый компонент Input для того, чтобы выводить ошибку валидации в обработчике события onBlur внутри Input-компонента
   `validator={validator}`

4. Для того, чтобы убирать последнюю ошибку сразу же, как только кнопка разблокируется и не дожидаться события onBlur можно использовать следующий хук внутри Input component

```
useEffect(() => {
    if (validator?.allValid()) {
        setError(null)
    }
}, [validator])
```

Критерии оценивания:

К1 — Написана страница регистрации (1 балл).
К2 — Написан API-роут согласно требованиям (1 балл).
К3 — Страница правильно и подробно типизирована (1 балл).
К4 — Написан Snapshot-тест к компоненту страницы (1 балл).
К5 — Выполнена полностью задача со звездочкой 7 (3 балла).
К6 — Выполнена полностью задача со звездочкой 8 (3 балла)

Итого: 10 баллов
Минимальное количество баллов, чтобы преподаватель смог зачесть вашу работу — 4