САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет
Лабораторная Работа №2
“Взаимодействие с внешним API”

Выполнил:
Стукалов Артем

Группа
K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

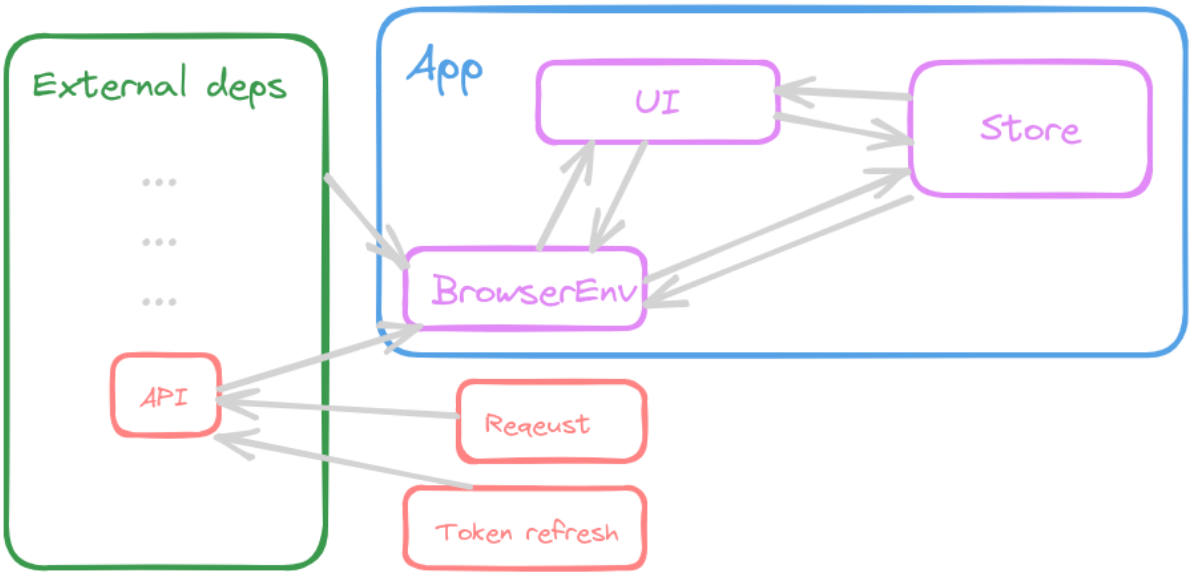
Задача

Реализовать взаимодействие с внешним API используя etch/axios/xhr.

Ход работы

В рамках данной работы будет продемонстрирована упрощенная схема глобального стора данных приложения wev.vk.me, включающая в себя интеграция с API Вконтакте.

Все приложение построено с расчетом на возможность внешней кастомизации через патерн “dependency injection”. В том числе и используемое апи. Передаваемые зависимости должны лишь реализовывать нужные интерфейсы. Также все базовые зависимости предоставляются в рамках пакета приложения и могут быть расширены по необходимости при интеграции приложения.



Работа API

Как можно видеть на схеме для API условно нужна функция request, которая будет выполнять запросы. Необязательно ходить в сеть, можно использовать моковые данные. Также нужна функция “token refresh”, которая будет перезагружать токен пользователя по запросу.

Внутри класса API есть очередь запросов. Каждый запрос помимо данных и метода, которые нужно использовать хранит в себе информация о кол-ве ретраев, интервале между ретраями и множитель данного интервала. После вызова метода fetch мы отсылаем запрос в функцию request внутри бесконечного цикла. Если удалось получить ответ, возвращаем его, иначе проверяем тип ошибки. Если это ошибка истекшего токена, то сбрасываем кол-во патраев и перезапрашиваем токен и ждем задержку. Если ошибка известная и это не токен, то просто ждем задержку. В противном случае выкидываем ошибку выше. Упрощенный код, реализующий данный алгоритм представлен ниже на скриншотах.

```ts
type Requests = {
  'messages.method': {
    params: {}
    response: {}
  }
}

type FetchOptions = {
  retries: number
  retryDelay: number
}

// Используем пустую функцию просто для примера
// eslint-disable-next-line @typescript-eslint/no-unused-vars
const externalDoRequest = async (...data: any): Promise<any> => {}

// Используем пустую функцию просто для примера
// eslint-disable-next-line @typescript-eslint/no-unused-vars
const getToken = (isExpired: boolean): Promise<string> => {
  return new Promise((resolve) => resolve(''))
}

class TokenError extends Error {
  constructor(message: string) {
    super(message)

    Object.setPrototypeOf(this, TokenError.prototype)
  }
}

class RequestError extends Error {
  constructor(message: string) {
    super(message)

    Object.setPrototypeOf(this, RequestError.prototype)
  }
}
```

```
const fetch = async <Method extends keyof Requests>(
  method: Method,
  params: Requests[Method]['params'],
  options: FetchOptions,
): Promise<Requests[Method]['response']> => {
  let fetchAttempts = 0
  let token = await getToken(false)

  while (true) {
    try {
      const res = await externalDoRequest(method, params, token)
      return res
    } catch (err: unknown) {
      fetchAttempts += 1
      if (fetchAttempts === options.retries + 1) {
        throw err
      }

      if (err instanceof TokenError) {
        fetchAttempts = 0
        token = await getToken(true)
        await sleep(options.retryDelay)
        continue
      }

      if (err instanceof RequestError) {
        await sleep(options.retryDelay)
        continue
      }
      throw err
    }
  }
}

```

Работа стора

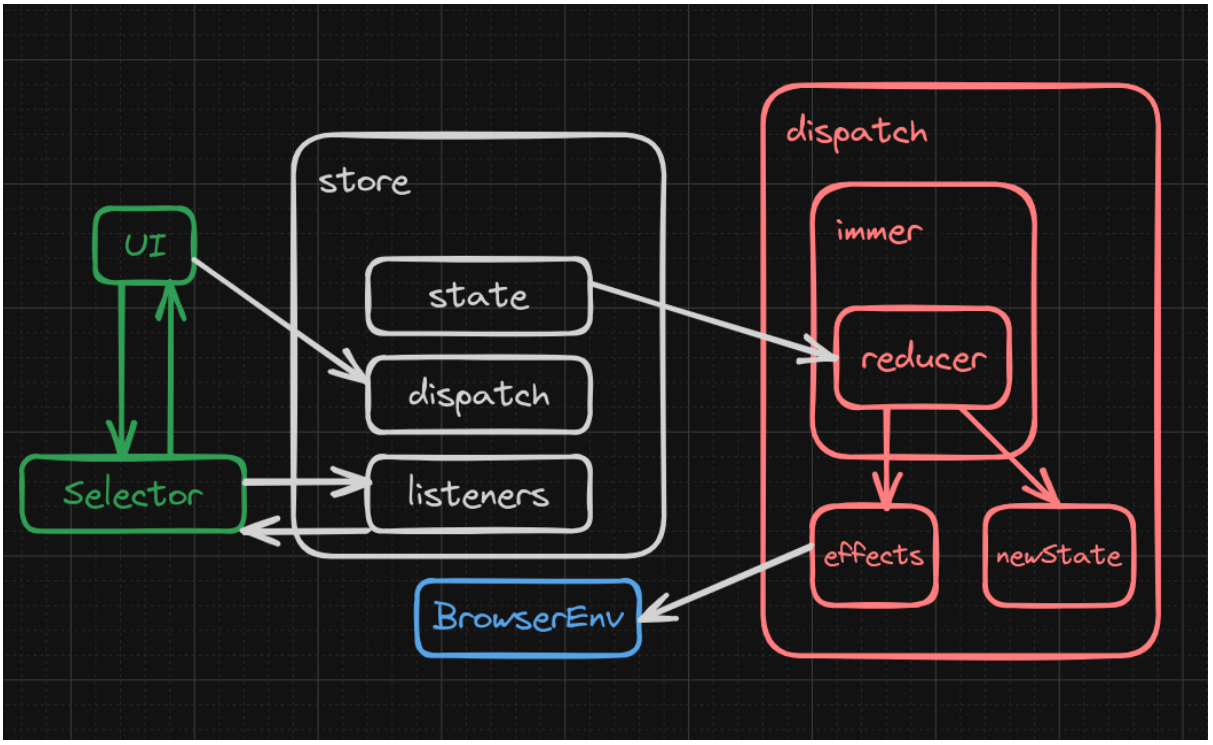
Далее рассмотрим алгоритм работы глобального стора приложения и его взаимодействие с ui. Данный стор является redux подобным. Флоу данных можно описать следующим образом:

1) Происходит подписка на стор и слушатели каждый раз получают обновленную версию состояния приложения после изменения. На

уровне UI это делается с использованием селекторов для оптимизации ререндеров.

2) Используя функцию dispatch можно вызвать выполнение к некоторого действия по изменению состояния.
3) После dispatch создается draft текущего состояния с использованием библиотеки immer для возможности мутирования данных.
4) draft попадает в функцию reducer, результатом работы которой является измененный стейт и некий массив асинхронных эфектов.
5) Оповещение слушателей об изменении ставится в очередь макро задач в event loop для оптимизации.
6) Эффекты имеет доступ к API через browserEnv. Результатом работы эффекта должен быть некий объект действия аналогичный передаваемому в функцию dispatch. В эффектах запрещено изменять состояние приложения.

Примерная схема работы стора представлена ниже:



Вывод

В рамках выполнения данной работы были освоены навыки создания функций взаимодействия с API, разработки глобальных сторов данных и их интеграции с API и UI приложения.