Введение

Итак, всем привет. Это поясняющий текст к проекту MakeYouHero.

В этом проекте реализован REST API для изменения и получения состояния сервеного объекта `hero`. Объект `hero` умеет сохранять свое состояние между перезапусками в конфигурационном файле.

Средствами `Express` содается http сервер, который обрабатывает запросы клиента. Настроенны обработчики 3х роутов:

`/heroStats` и `/heroImage` - в зависимости от типа запроса `get` или `post` клиент получает свойство или устанавливает его. `/` - методом `get` клиент может получить все маршруты обрабатываемые сервером и поддерживаемые ими методы доступа.

Соглашение возврата результата клиенту:

Результат возвращается в формате json, с обязательным полем `error` - которое содержит ошибку или `null` (признак отсутствия ошибки). И необязательное поле `result`, которое может быть заполненно,если `error` = `null`.

```
{
    error: null,
    result: 'some result'
}
```

Запуск и тестирование

Перейдите в папку `makeYouHero` и выполните две команды:

```
npm install
node .
```

Это запустит сервер.

Если вы хотите также выполнить тесты:

```
npm test
```

Для тестирования я настаиваю на использовании вами утилиты: httpie
Это такой curl удобно форматирующий вывод, в зависимости от mime-type и предоставляющий более удобный CLI.

Команды для тестирования при помощи `httpie` вы найдете в файле `test with http util.txt`
Команды для тестирования при помощи `curl` вы найдете в файле `test with curl.txt`

Hero:

Класс `Hero` предоставляет 4 публичных метода для взаимодействия с 2 внутренними свойствами: `stats` и `heroImage`,

2 метода геттера и 2 метода сеттера соответственно.

Конструктор класса `Hero` может быть инициирован начальными состояниями явно. Приняв исходные `stats` и `heroImage`. И объект конфига, в который будут сохранятся изменения внутренних свойств.

А может быть инициирован объектом класса SimpleConfig. Из которого конструктор извлечет исходные значения и в средствами
которого будет обновлять конфугурационный файл при изменении внутренних свойств.

Конструктор:

```
var hero = new Hero(stats, heroImage, [simpleConfig, accessHeroImageSize,
accessHeroImageFormats]);
...

var hero = new Hero(simpleConfig, [accessHeroImageSize, accessHeroImageFormats]);
...
```

Здесь и далее в прямоугольных скобках указанны необязательные параметры.

- `stats` - обект со следующим шаблоном:

```
statsPattern = {
    name: 'string',
    strength: 'int',
    dexterity: 'int',
    intellect: 'int',
    isInvincible: 'boolean'
};
```

- `heroImage` - пуст к изображению, размер которого не привышает `accessHeroImageSize`, а расширение соответствует `accessHeroImageFormats`.
- `simpleConfig` - объект конфигурации, из которого можно извлечь хранимые поля, и обновить для использования после перезапуска программы.
- `accessHeroImageSize` - максимальный размер `heroImage` в байтах.
  `default`: `1024*1024`.
- `accessHeroImageFormats` - массив с разрешенными расширениями для `heroImage`.
  `default`: `['.png', '.jpg']`.

Методы:

- `setStats` - принимает объект `stats`, верифицирует его, и изменет внутренне состояние объекта.
- `getStats` - возвращает свойство `stats`.
- `setHeroImage` - принимает путь к файлу, верифицирует его, и изменяет внутренне состояние объекта.

- `getHeroImage` - возвращает путь к файлу.

Что возможно стоило бы сделать по другому, но уже нет времени толком обдумать:

Убрать зависимость класса `Hero` от конфига:

- Извлекать исходное состояние `stats` и `heroImage` из конфига.
- Передавать в конструктор эти исходные состояния.
- Выбрасывать из объекта класса `Hero` события `newStats` и `newHeroImage` при изменении соответствующих полей.
- Навесить слушателей вне класса `Hero` которые обновляли бы соотсветствующие поля в конфиге.

HeroProvider

Объект этого класса хранит в себе ссылку на объект `hero` доступ к которому он обеспечивает.
Имена и суть методов этого класса повторяет имена методов класса `Hero`.
С той лишь разницей, что они принимают аргументы: `request` и `response`.

Внутри методов происходит обработка `request`, вызов метода объекта класса `Hero`. И возврат результата в `json` формате средствами объекта `response`.

`json` ответ придерживается следующего шаблона:

```
{
    error: null,
    result: 'some result'
}
```

по сути повторяя стандарт который принят для каллбеков.

Конструктор:

```
var heroProvider = new HeroProvider(hero, uploadsDir);
...
```

- `hero` - ссылка на объект к которому провайдер дает доступ.
- `uploadsDir` - папка в которую будет загруженны `heroImage`.

Что возможно стоило бы сделать по другому, но уже нет времени толком обдумать:

- Сейчас файл для `heroImage` передается от клиента на сервер в поле "формы", у меня есть подозрение, что файл можно передать методо POST явно,
  не оборачивая его в "форму".
  Примечание имя этого поля: `avatar`.
- Стоит сделать более проработанную обработку некорректных запросов от клиента.
  Сейчас, если с запросом, что-то не так, то сервер просто отвечает как должно было быть и

что прислал клиент.
- Очищать папку для загрузок `heroImage` от невалидных устаревших файлов.

SimpleConfig

Это обертка над популярным модулем `nconf`, он очень мощный, но в данном проекте мне потребовалась только десятая часть его функционала,
поэтому сделана обертка, конструктор которой принимает путь к конфигурационному файлу.

И предоставляет два метода:

- `get` - для извлечения поля.
- `set` - для установки значения поля и последующего сохранения изменения в конфигурационный файл.

```
var someConfig = new SimpleConfig('/path/to/someConfig');
...
var otherConfig = new SimpleConfig('/path/to/OtherConfig.json')
...
```

Примечание: отсутствие `.json` это не опечатка, а сахар который добавляет обертка.