

Стажировка

Задание № 2

По теме

«Сетевые запросы и регулярные выражения»

Выполнил: Ноздряков Богдан Валериевич

Проверил: Пармузин Александр Игоревич

Санкт-Петербург

2024

Примечание:

Основной файл - ./main.js

Модули - ./src/

Итоговый JSON-файл - ./storage/storage.json

Файлы ошибок - ./logs/28.05.2024 19-01-39.txt и ./logs/28.05.2024 19-02-38.txt

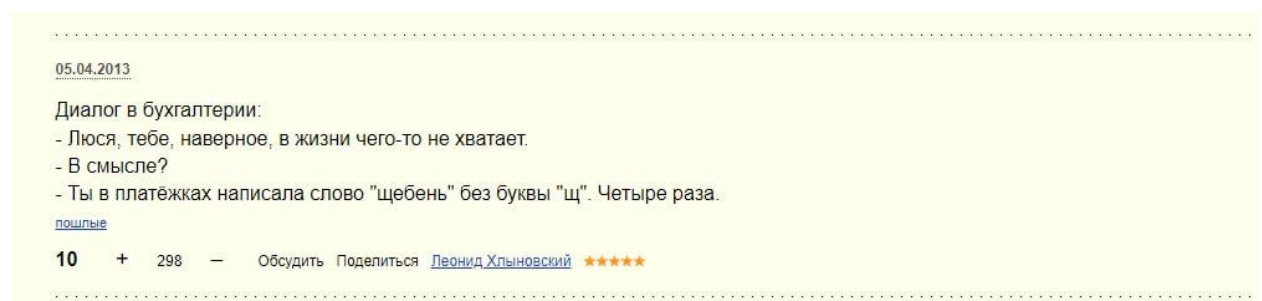
Отчет состоит из условия задачи, анализа заданной веб-страницы, решения (описывается код) и тестов (провожу запуск кода и показываю результаты)

Условие:

Задание связано с сетевыми запросами и регулярками. Необходимо получить HTML код страницы сайта и с помощью регулярных выражений вырезать из этого кода нужные данные и представить их в виде JSON.

Пациент - сайт отечественных шуток и анекдотов. Вот по этому URL <https://www.anekdot.ru/random/anekdot/> откроется страница, где будут выведены 20 случайных анекдотов вперемешку с рекламой и прочими визуальными элементами сайта. Необходимо проанализировать код страницы, средствами Node.js получить этот самый HTML код, и с помощью регулярок вычленив из него тексты анекдотов и прочие полезные данные, и сохранить их в JSON файл. Сохранение осуществлять по одному анекдоту. Так как анекдотов 20, то в JSON файле должен появиться массив из 20 объектов.

Вот пример анекдота с сайта:



Для него должен появиться такой JSON объект:

```
{  
  "id": 632599,
```

```
"text": "Диалог в бухгалтерии:\n - Люся, тебе, наверное, в жизни чего-то не хватает.\n - В смысле?\n - Ты в платёжках написала слово \"щебень\" без буквы \"щ\". Четыре раза.",
```

```
"date": "2013-04-05T00:00:00.000Z",
```

```
"rating": 298,
```

```
"tags": ["пошлые"],
```

```
"author": "Леонид Хлыновский"
```

```
}
```

У некоторых анекдотов может не быть автора, тогда поле `author` должно быть равно `null`, у некоторых может не быть тэгов, тогда поле `tags` будет пустым массивом, все остальные поля должны быть у любого анекдота.

Так же необходимо предусмотреть обработку ошибок на разных этапах:

- Сетевые ошибки. Ошибки соединения, получения.
- Ошибки разбора полученного HTML, с выводом тэга и наименования операции
- Ошибки сохранения и дозаписи сформированной структуры JSON на диск
- Прочие ошибки при выполнении программы

Все ошибки должны быть записаны на диск в log файл. Наименование файла – текущие дата/время (dd.mm.yyyy hh:mm:ss).

Формат ошибки:

<Дата> <Время> - <Тип ошибки>:

<CustomError>: <Тут наше описание ошибки>

<SystemError>: <Тут системный текст ошибки>

Например:

14.12.2023 11:15 – Ошибка разбора HTML:

CustomError: Ошибка разбора HTML в строке <div class="text">Диалог в бухгалтерии...

SystemError: Error in line 34 at position 25...

Анализ:

Веб-страница по данному адресу <https://www.anekdot.ru/random/anekdot/> состоит из 20-ти случайных анекдотов. У каждого анекдота есть id, текст, дата публикации, рейтинг. Также у анекдотов автор, может быть, либо указан, либо не указан. То же касается и тегов – они могут быть либо указаны, либо не указаны. Тегов может быть больше одного.

Все анекдоты располагаются в элементе div с классом topicbox. У этого элемента div есть атрибут data-id, который и содержит искомое id. Содержимое данного div:

- a. Элемент p с классом title, который содержит ссылку, содержащую искомую дату публикации
- b. Элемент div с классом text, который содержит искомый текст анекдота
- c. Элемент div с классом tags, который содержит ссылки на теги, относящиеся к данному анекдоту
- d. Элемент div с классом votingbox, который содержит элемент div с классом rates, который имеет атрибут data-r, где первое значение до двоеточия – искомый рейтинг
- e. Элемент div с классом btn2, который содержит ссылку с классом auth, которая предоставляет автора анекдота

Решение:

Для решения данной задачи – воспользуемся сетевым запросом GET, а также обычными регулярными выражениями.

Для этого создадим модуль htmlProcessing для получения html-кода страницы по переданному url и его обработки:

```
export async function htmlProcessingService(url) {
  return (await getHTML(url));
}

export function removeHTML(str) {
  return str.replace(/<br¥s*¥/?>/gi, ' ¥n')
    .replace(/<[>]*?>/gm, ' ');
}

async function getHTML(url) {
  if (typeof url !== 'string') {
    throw new Error('Url must be a string!');
  }

  const response = await fetch(url);
  if (!response.ok) {
    throw new Error(`${response.status} ${response.text}`);
  }

  return (await response.text());
}
```

Здесь:

- a. htmlProcessingService – асинхронная функция, которая вызывает функцию getHTML (в будущем сюда возможно добавить какую-то логику предобработки. Именно поэтому я создал здесь эту функцию, а не использовал только getHTML)
- b. getHTML – асинхронная функция, которая по заданному url получает html-код страницы и возвращает его в виде строки

removeHTML – функция, которая удаляет в строке все html-теги, при этом теги br заменяются на \n, а все остальные теги – на пробел. Описание регулярного выражения:

1) /<[a-z]+ class="text">(.*?)<\/[a-z]+>/s:

- <[a-z]+ class="text"> – ищем открывающий тег html, который начинается с <, затем идет атрибут class="text" и заканчивается >
- (.*?) – группа захвата которая соответствует любому количеству любых символов (кроме новой строки, благодаря флагу s внизу), минимально необходимому для соответствия

- `<[/[a-z]+>` - ищет закрывающий тег HTML, который начинается с символа `</`, за которым следует один или несколько символов в нижнем регистре.
- `/s` – флаг, который позволяет регулярному выражению учитывать новые строки в качестве одного из возможных символов, что важно при работе с HTML, где теги могут быть разделены новыми строками

2) `/<br\s*\/?>/gi`:

- `<br` – соответствие тегу `br`
- `\s*` – соответствует нулю или нескольким пробельным символам (включая пробелы, табуляции, переводы строк и т.д.). Это позволяет учесть пробелы, которые могут следовать непосредственно за тегом `br`
- `/?` – символ `/` соответствует символу `/`, который является обязательным в конце тега `
` в XHTML или XML. Однако, в HTML5 тег `
` может быть написан как `
` без слеша. Символ `/` делает предыдущий символ (в данном случае `/`) необязательным, что позволяет регулярное выражение соответствовать как `
`, так и `
`
- `g` – флаг, который указывает на глобальный поиск. Он позволяет найти все совпадения в тексте, а не только первое
- `i` – флаг, который указывает на игнорирование регистра. Он позволяет регулярное выражение соответствовать символам верхнего и нижнего регистра без различия

3) `/<[^>]*>?/gm`:

- `<` – соответствие символу `<` (открывающий символ тега)
- `\s*` – соответствует нулю или нескольким пробельным символам (включая пробелы, табуляции, переводы строк и т.д.). Это позволяет учесть пробелы, которые могут следовать непосредственно за тегом `br`
- `[^>]*` – группа захвата, которая соответствует любому количеству символов, которые не являются символом `>`
- `>?` – Символ `>` соответствует символу закрытия тега в HTML или XML. Символ `?` делает предыдущий символ (в данном случае `>`) необязательным, что позволяет регулярному выражению соответствовать тегам, которые могут быть закрытыми или пустыми

- `g` – флаг, который указывает на глобальный поиск. Он позволяет найти все совпадения в тексте, а не только первое
- `m` – флаг, который указывает на многострочный режим. Он позволяет регулярному выражению соответствовать символам в начале и конце каждой строки, а не только в начале и конце всего текста

А также модуль `regex` для применения регулярных выражений к строкам:

```
export function applyRegexByCleanedExec(regex, str) {
  applyRegexCheck(regex, str);

  let matches = [];
  let match;

  while ((match = regex.exec(str)) !== null) {
    const cleanedMatch = match[0].replace(/¥s+/g, ' ').trim();
    matches.push(cleanedMatch);
  }

  return matches;
}

export function applyRegexByFirstExec(regex, str) {
  applyRegexCheck(regex, str);

  const result = regex.exec(str);
  if (result && result[1]) {
    return result[1];
  }
}

export function applyRegex(regex, str) {
  applyRegexCheck(regex, str);
  return str.match(regex);
}

function applyRegexCheck(regex, str) {
  if (!(regex instanceof RegExp) || typeof str !== 'string') {
    throw new Error('regex must be a regular expression and str must be a string');
  }
}
```

Здесь:

a. Функция `applyRegexByCleanedExec`:

-Данная функция применяет регулярное выражение к html-коду. Она извлекает все совпадения, после чего очищает эти совпадения от лишних пробелов и добавляет их в массив

b. Функция `applyRegexByFirstExec`:

-Данная функция применяет регулярное выражение к html-коду. Она ищет только первое совпадение и возвращает первую группу захвата этого совпадения

c. Функция `applyRegex`:

-Данная функция применяет регулярное выражение к html-коду. Она ищет все совпадения

d. Функция `applyRegexCheck`:

-Данная функция используется для проверки типов параметров `regex` и `html`, которые передаются во все функции применения регулярного выражения:
`applyRegexByCleanedExec`, `applyRegexByFirstExec`, `applyRegex`

Также для работы, потребуется модули для работы с файловой системой, с датой/временем, с анекдотами, а также с логами ошибок. Все модули можно поделить на слои – `domain` (отвечает за работу с данными, а также за определение структуры данных), `helpers` (отвечает за взаимодействие с функционалом различных сущностей, который можно назвать “вспомогательным”) и `logger` (отвечает за логирование). Рассмотрим каждый из них:

1) Модуль `story` слоя `domain`:

1.1) `storyModel.js` – представляет из себя модель, по которой формируются все анекдоты (некий интерфейс анекдота):

```
const storyModel = {
  id: null,
  text: null,
  date: null,
  rating: null,
  tags: [],
  author: null
};

export default storyModel;
```


- 1.2) storyService.js – представляет из себя сервис, который из полученный строки с html-кодом веб-страницы, получает все анекдоты, а затем преобразует их в нужный формат:

```
import storyModel from './storyModel.js';
import { htmlProcessingService, removeHTML } from
'../htmlProcessing/htmlProcessingService.js';
import {
  applyRegexByClearedExec,
  applyRegexByFirstExec,
  applyRegex
} from '../../helpers/regex/regexUtils.js';

async function storyService(url) {
  const storyDataStorage = [];
  const HTML = await htmlProcessingService(url);

  const htmlDataChunk = applyRegex(
    /<[a-z]+[>]*class="topicbox"[^>]*>([sS]*?(?=<[a-
z]+[>]*class="topicbox"|<\/body>))/gs,
    HTML
  );

  for (let i = 0; i < htmlDataChunk.length; i++) {
    const storyData = { ...storyModel };

    storyData.id = applyRegexByFirstExec(/data-id="([~"]+)"/, htmlDataChunk[i]);
    if (!storyData.id) {
      continue;
    }

    storyData.text = removeHTML(applyRegexByFirstExec(/<[a-z]+ class="text">(.*?)<\/[a-
z]+>/s, htmlDataChunk[i]));

    const storyDataDate = applyRegexByFirstExec(
      /<[a-z]+ class="title">[sS]*<a href=".[?"]>(.*?)<\/a>[sS]*<\/[a-z]+>/, htmlDataChunk[i]
    );
    const [storyDataDateDay, storyDataDateMonth, storyDataDateYear] =
storyDataDate.split('.').map(Number);

    storyData.date = new
Date(`${storyDataDateMonth}.${storyDataDateDay}.${storyDataDateYear}`);
    storyData.rating = applyRegexByFirstExec(/data-r="([~;]+)/, htmlDataChunk[i]);

    const storyDataTagsContainer = applyRegexByFirstExec(/<div
class="tags">(.*?)<\/div>/s, htmlDataChunk[i]);

    if (storyDataTagsContainer) {
```

```

    const storyDataTagsItem = applyRegexByCleanedExec(/<a href="(.*?)">(.*?)</a>/g,
storyDataTagsContainer);
    const currentStoryTags = storyDataTagsItem.map(item =>
applyRegexByFirstExec(/>(.*?)</s, item)); // map вместо for
    storyData.tags = currentStoryTags;
  }

  const dataAuthor = applyRegexByFirstExec(
    /<[a-z]+ class="auth" href="["']*"(.*?)</[a-z]+>/s, htmlDataChunk[i]
  );

  storyData.author = dataAuthor ? dataAuthor : `${storyData.author}`;
  storyDataStorage.push(storyData);
}

return storyDataStorage;
}

export default storyService;

```

Разбор:

Нам нужно получить каждый анекдот и записать его в массив storyDataStorage в виде объекта storyModel. Сначала получим html-код веб-страницы с помощью модуля htmlProcessing:

```
const HTML = await htmlProcessingService(url);
```

Во избежание ошибок и неправильной записи данных анекдотов в JSON-файл, разобьем всю веб-страницу на чанки, где каждый чанк будет содержать один анекдот, и уже будем осуществлять поиск и преобразование анекдота по очереди в каждом чанке:

```

const htmlDataChunk = applyRegex(
  /<[a-z]+[<^>]*class="topicbox"[<^>]*(<[^\s\S]*?(?=<[a-z]+[<^>]*class="topicbox"|</body>))/gs,
  HTML
);

```

Здесь с помощью регулярного выражения и модуля регулярных выражений, в htmlDataChunk записывается массив, где каждый элемент – это элемент веб-страницы div с классом topicbox. Разбор регулярного выражения:

- а. /<[a-z]+[<^>]* - эта часть ищет открывающий тег <, за которым следует имя тега, состоящее из строчных букв [a-z]+. Следующий [<^>] означает любое количество символов, которые не являются > (это позволяет игнорировать атрибуты в теге). Затем идет class="topicbox", что указывает на конкретный класс, который мы хотим найти

- b. `([\\s\\S]*?(?=<[a-z]+[>]*class="topicbox" | </body>))` - Здесь происходит захват содержимого между нашими тегами. `[\\s\\S]` означает любой символ, включая перенос строки, что позволяет захватывать весь текст внутри тега. `*?` делает поиск нежадным, то есть он пытается захватить как можно меньше текста до следующего совпадения. `(?=<[a-z]+[>]*class="topicbox" | </body>)` является утверждением, которое говорит о том, что после захваченного текста должен следовать либо другой тег с классом `topicbox`, либо закрывающий тег `</body>`. Это предотвращает бесконечные циклы при поиске
- c. `/gs` – флаги регулярного выражения, где `g` - означает глобальный поиск, что позволяет найти все совпадения в тексте, а не только первое, а `s` - означает режим "dotall", где точка `.` соответствует любому символу, включая перенос строки, что расширяет возможности захвата текста

После получения всех анекдотов в массив `htmlDataChunk`, нужно пройти по каждому анекдоту, получить из него данные и записать их в специальный объект, а в конце добавить этот объект в `storyDataStorage` и после перейти к следующему анекдоту:

```
for (let i = 0; i < htmlDataChunk.length; i++) {
  const storyData = { ...storyModel };

  storyData.id = applyRegexByFirstExec(/data-id="([^\"]+)"/, htmlDataChunk[i]);
  if (!storyData.id) {
    continue;
  }

  storyData.text = removeHTML(applyRegexByFirstExec(<[/a-z]+ class="text">(.*?)<[/a-z]+>/s, htmlDataChunk[i]));

  const storyDataDate = applyRegexByFirstExec(
    <[/a-z]+ class="title">%s*<a href=".+?">(.*?)<[/a]>%s*<[/a-z]+>/, htmlDataChunk[i]
  );
  const [storyDataDateDay, storyDataDateMonth, storyDataDateYear] =
    storyDataDate.split('.').map(Number);

  storyData.date = new
  Date(`${storyDataDateMonth}.${storyDataDateDay}.${storyDataDateYear}`);
  storyData.rating = applyRegexByFirstExec(/data-r="([^\"]+)"/, htmlDataChunk[i]);

  const storyDataTagsContainer = applyRegexByFirstExec(<[/div
class="tags">(.*?)<[/div>/s, htmlDataChunk[i]);

  if (storyDataTagsContainer) {
    const storyDataTagsItem = applyRegexByCleanedExec(<[/a href="(.*?)">(.*?)<[/a>/g,
    storyDataTagsContainer);
    const currentStoryTags = storyDataTagsItem.map(item =>
    applyRegexByFirstExec(<[/>(.*?)<[/s, item)); // map в м е с т о f o r
    storyData.tags = currentStoryTags;
  }

  const storyDataAuthor = applyRegexByFirstExec(
```

```

    /<[a-z]+ class="auth" href="["']*">(.*?)<\/[a-z]+>/s, htmlDataChunk[i]
  );

  storyData.author = storyDataAuthor ? storyDataAuthor : `${storyData.author}`;
  storyDataStorage.push(storyData);
}

```

Здесь:

- a. storyData – объект, представляющий собой определенный анекдот
- b. storyData.id – ищется id анекдота с помощью регулярного выражения /data-id="([^^"]+)"/ в определенном чанке. Если найти не получилось, то пропускается итерации цикла, так как в таком случае считается, что данный чанк не будет иметь анекдота. Описание регулярного выражения:
 - data-id=" – ищем начало атрибута data-id
 - ([^^"]+) – группа захвата для одного или несколько символов, которые не являются двойными кавычками
- c. storyData.text – ищется текст анекдота с помощью регулярного выражения /<[a-z]+ class="text">(.*?)<\/[a-z]+>/s в определенном чанке. Если получилось найти, то вызывается функция removeHTML модуля htmlProcessing, которая заменяет все теги br на \n, а все оставшиеся теги на пробел
- d. storyData.date - ищется дата публикации анекдота с помощью регулярного выражения /<[a-z]+ class="title">\s*(.*?)<\/a>\s*<\/[a-z]+>/ в определенном чанке. Но полученный формат нас не устраивает, так как он будет следующим: dd.mm.yy. А нам нужно будет из это создать объект Date, который принимает данные в формате mm.dd.yy. Поэтому воспользуемся дальше деструктуризацией и запишем в data.date дату публикации анекдота в нужном формате. Описание регулярного выражения:
 - <[a-z]+ class="title">\s* – ищем открывающий тег html, который начинается с <, затем один или несколько символов в нижнем регистре, а затем имеет атрибут class="title". \s* соответствует нулю или нескольким пробельным символам, включая пробелы, табуляции и переводы строк
 - \s* – ищет ноль или несколько пробельных символов, за которыми следует открывающий тег a с атрибутом href, содержащим хотя бы один символ. /s* снова соответствует нулю или нескольким пробельным символам
 - (.*?) - Это группа захвата, которая соответствует любому количеству любых символов (кроме новой строки, благодаря отсутствию флага s), минимально необходимому для соответствия

- `<\a>\s*<\[a-z]+>` – ищет закрывающий тег `a` и нуль или несколько пробельных символов, за которыми следует закрывающий тег, соответствующий типу открывающего тега, найденного ранее
- e. `storyData.rating` - ищется рейтинг анекдота с помощью регулярного выражения `/data-r="([^\;]+)/` в определенном чанке. Описание регулярного выражения:
- `data-r="` – ищем начало атрибута `data-r`
 - `([^\;]+)` – группа захвата, которая соответствует одному или нескольким символам, которые НЕ являются символом точки с запятой
- f. `storyDataTagsContainer` - ищется контейнер (`div` с классом `tags`) анекдота с помощью регулярного выражения в определенном чанке. Если его получилось найти (у анекдота указаны теги), то в `storyDataTagsItem` записываются все ссылки из `storyDataTagsContainer` (с помощью регулярного выражения), а после происходит цикл по каждому элементу `storyDataTagsItem`, где для каждого элемента берется значение ссылки (с помощью регулярного выражения), то есть сам тег, и записывается в массив `data.tags`. Описание регулярного выражения:

1) `/<div class="tags">(.*?)</div>/s:`

- `<div class="tags">` – ищет открывающий тег `div`, за которым следует атрибут `class="tags"`. Это указывает на начало блока контента, который мы хотим извлечь
- `(.*?)` – группа захвата, которая соответствует любому количеству любых символов (кроме новой строки, благодаря флагу `s` внизу), минимально необходимому для соответствия
- `</div>` - ищет закрывающий тег `div`, который соответствует концу блока контента, который мы хотим извлечь
- `/s` - флаг, который делает точку `.` соответствующей также символу новой строки, что расширяет ее возможности поиска

2) `/(.*?)/g:`

- `<a href="` – ищет открывающий тег `a`, за которым следует атрибут `href`. Это указывает на начало атрибута `href`, который содержит URL-адрес
- `(.*?)` – Первая группа захвата соответствует любому количеству любых символов (кроме новой строки), минимально необходимому для соответствия. Это позволяет извлечь значение атрибута `href`

- `>(.*)` – символ `>` соответствует закрывающему слэшу тега. Вторая группа захвата `(.*)` соответствует любому количеству любых символов (кроме новой строки), минимально необходимому для соответствия. Это позволяет извлечь содержимое тега `a`
- `` - ищет закрывающий тег `a`
- `/g` - флаг, который указывает на глобальный поиск, позволяя найти все совпадения в тексте, а не только первое

3) `/>(.*)</s`:

- `>` – ищет символ `>` (символ закрытия html-тега)
- `(.*)` – группа захвата, которая соответствует любому количеству любых символов (кроме новой строки), минимально необходимому для соответствия
- `</` - ищет символ `</` (символ начала закрывающего html-тега)

g. `storyDataAuthor` - ищется автор анекдота с помощью регулярного выражения `<[a-z]+ class="auth" href="^[^"]*">(.*)</[a-z]+>/s` в определенном чанке. Если его нет, то в `storyData.author` будет записан `null`, в противном случае – сам автор. Описание регулярного выражения:

- `<[a-z]+ class="auth" href="^[^"]*">` – ищет открывающий тег HTML, который начинается с символа `<`, за которым следует один или несколько символов в нижнем регистре, а затем имеет атрибуты `class="auth"` и `href`. Атрибут `href` должен содержать хотя бы одну цифру или специальный символ, но не должен содержать двойные кавычки
- `(.*)` - Это группа захвата, которая соответствует любому количеству любых символов, минимально необходимому для соответствия
- `</[a-z]+>` – ищет закрывающий тег HTML, который начинается с символа `</`, за которым следует один или несколько символов в нижнем регистре

h. `storyDataStorage.push(data)` – в конце итерации в `storyDataStorage` добавляется созданный объект анекдота

2) Модуль `date` слоя `helpers` – предоставляет функции, которые позволяют получать определенную часть даты в отформатированном виде:

```

export function getCurDate() {
  return new Date().getDate().toString().padStart(2, '0');
}

export function getCurMonth() {
  return (new Date().getMonth() + 1).toString().padStart(2, '0');
}

export function getCurYear() {
  return new Date().getFullYear();
}

export function getCurHours() {
  return new Date().getHours().toString().padStart(2, '0');
}

export function getCurMinutes() {
  return new Date().getMinutes().toString().padStart(2, '0');
}

export function getCurSeconds() {
  return new Date().getSeconds().toString().padStart(2, '0');
}

```

- 3) Модуль file слоя helpers – предоставляет функции, для работы с файловой системой:

```

import fs from 'fs';

export function appendFile(filePath, message) {
  fs.appendFileSync(filePath, message + '¥n', 'utf8');
}

export function writeToFile(filePath, record) {
  fs.writeFileSync(filePath, record);
}

export function ensureDirectoryExists(directoryPath) {
  if (!fs.existsSync(directoryPath)) {
    fs.mkdirSync(directoryPath, { recursive: true });
  }
}

export function clearFile(filePath) {
  if (fs.existsSync(filePath)) {
    fs.writeFileSync(filePath, '');
  }
}

```

Здесь:

- a. fs – модуль, необходимый для работы с файловой системой диска
 - b. appendFile – функция, которая предназначена для добавления текста в конец файла по указанному пути
 - c. writeToFile – функция, которая записывает в файл переданную запись
 - d. ensureDirectoryExists – функция, которая проверяет, что директорию по указанному пути существует. Если директория существует, то функция ничего не делает. Если директории не существует, то функция ее создает:
 - e. clearFile – функция, которая проверяет – содержит ли переданный файл какие-то записи. Если содержит, то функция очищает этот файл
- 4) Модуль errorLogger слоя logger – позволяет производить логирование любых ошибок в файл по определенному пути:

```
import path from 'path';
import { appendFile, ensureDirectoryExists } from "../helpers/file/fileService.js";
import {
  getCurDate,
  getCurMonth,
  getCurYear,
  getCurHours,
  getCurMinutes,
  getCurSeconds
} from '../helpers/date/dateUtils.js';

const LOGGER_DIR = './logs';

function logErrorToFile(error) {
  ensureDirectoryExists(LOGGER_DIR);

  const errorLogFilePath = path.join(
    LOGGER_DIR,
    `${getCurDate()}.${getCurMonth()}.${getCurYear()} ${getCurHours()}-${getCurMinutes()}-${getCurSeconds()}.txt`
  );

  const lines = error.stack.split('\n');
  const typeError = lines[0].split(':')[0];
  const atIndex = lines.findIndex(line => line.trim().startsWith('at '));

  let location = 'Unknown Location';

  if (atIndex !== -1) {
```



```

const locationLines = lines.slice(atIndex).join('
');
location = locationLines.trim();

const locationsArray = location.split('
').map(line => {
  const startCut = line.indexOf("file:");
  const endCut = line.indexOf("task_2");

  if (startCut !== -1 && endCut !== -1) {
    const result = line.substring(0, startCut) + line.substring(endCut);
    return `${result}`;
  }
  return line;
});

location = locationsArray.join('
');
}

const errorMessage = `${getCurDate()}.${getCurMonth()}.${getCurYear() }
${getCurHours()}:${getCurMinutes() } - ${typeError}:
  CustomError: ${error.message}
  SystemError: ${location}
`;

appendFile(errorLogFilePath, errorMessage);
}

export default logErrorToFile;

```

Здесь:

- a. path – модуль, необходимый для работы с путями файловой системы диска
- b. `LOGGER_DIR` - константа, которая содержит путь до директории, в которую будут записываться все файлы .txt, содержащие ошибки. В данном случае – это директория logs в корне проекта
- c. logErrorToFile – функция, которая записывает в создаваемый ею файл сообщение об ошибки в заданном формате:

<Дата> <Время> - <Тип ошибки>:

<CustomError>: <Тут наше описание ошибки>

<SystemError>: <Тут системный текст ошибки>

- d. errorLogFilePath – создается путь до файла ошибок, где файл ошибок имеет заданный формат наименования (текущее время) – dd.mm.yy. hh:mm:ss.txt

Далее запустим основную функцию main, которая является точкой входа:

```
import path from 'path';
import logErrorToFile from './src/logger/errorLogger/errorLogger.js';
import storyService from './src/domain/story/storyService.js';
import {
  writeFile,
  ensureDirectoryExists,
  clearFile
} from './src/helpers/file/fileService.js';

const URL = 'https://www.anekdot.ru/random/anekdot/';
const STORAGE_DIR = './storage';
const STORAGE_FILE_NAME = 'storage.json';

main().catch((err) => {
  logErrorToFile(err);
})

async function main() {
  let dataStorage = await storyService(URL);
  dataStorage = JSON.stringify(dataStorage, null, 3);

  ensureDirectoryExists(STORAGE_DIR);
  const storageFilePath = path.join(STORAGE_DIR, STORAGE_FILE_NAME);

  clearFile(storageFilePath);
  writeFile(storageFilePath, dataStorage);
}
```

Здесь:

- a. path – модуль, необходимый для работы с путями файловой системы диска
- b. URL – константа, представляющая url ресурса, с которым мы работаем
- c. STORAGE_DIR – константа, которая содержит путь до директории, в которой будет храниться JSON-файл, в который в итоге будут записываться анекдоты. В данном случае – это директория storage в корне проекта
- d. STORAGE_FILE_NAME – константа, представляющая из себя имя самого JSON-файла по директории STORAGE_DIR
- e. dataStorage – массив, который будет содержать объекты анекдотов, который в итоге будет записан в JSON-файл

То есть сначала с помощью модуля story – мы получаем все анекдоты в отформатированном представлении:

```
let dataStorage = await storyService(URL);
```

После записи всех анекдотов в dataStorage – нужно перобразовать массив dataStorage в JSON-строку:

```
dataStorage = JSON.stringify(dataStorage, null, 3);
```

Далее просто записываем весь dataStorage в определенный JSON-файл:

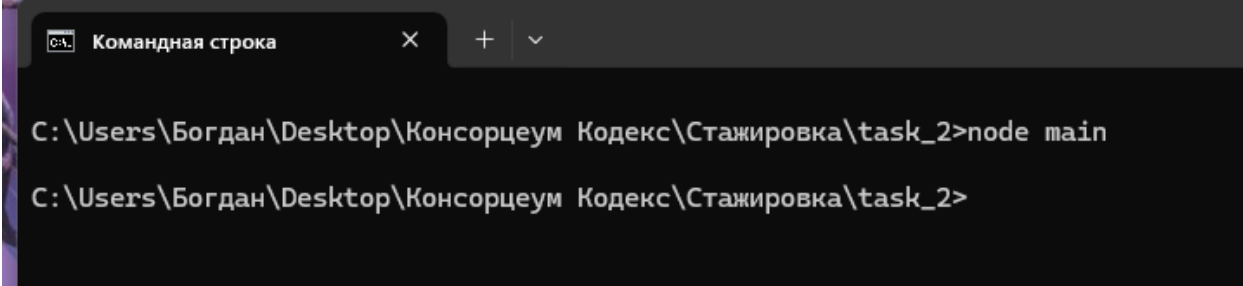
```
ensureDirectoryExists(STORAGE_DIR);  
const storageFilePath = path.join(STORAGE_DIR, STORAGE_FILE_NAME);  
  
clearFile(storageFilePath);  
writeToFile(storageFilePath, dataStorage);
```

Таким образом, в корне проекта появится директория Storage, в которой будет находиться файл storage.json, куда будут записываться все 20 анекдотов каждый раз, при запуске программы, причем при каждом запуске – старое содержимое файла storage.json будет заменяться на новое

Тесты:

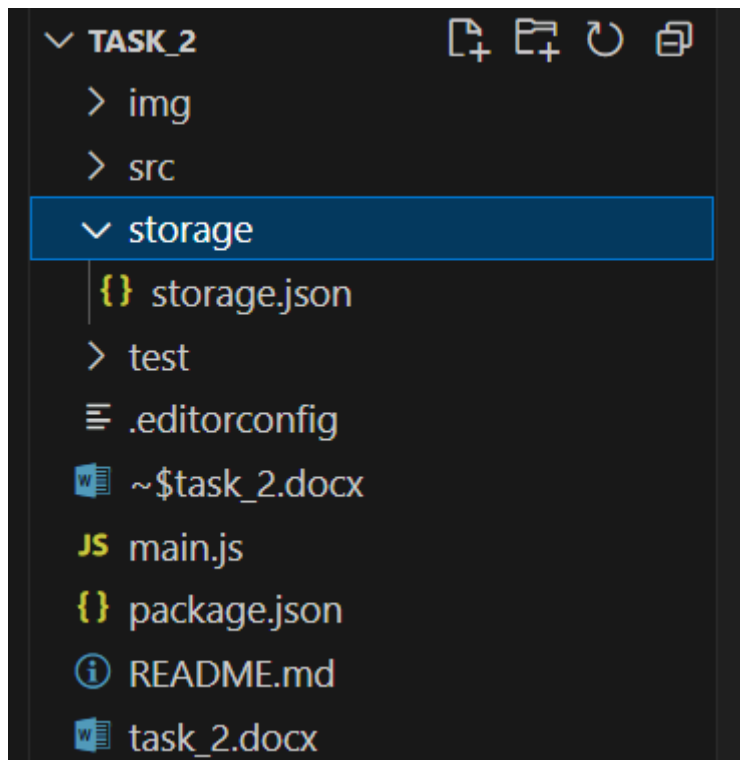
а. Без ошибок:

-Запустим программу:

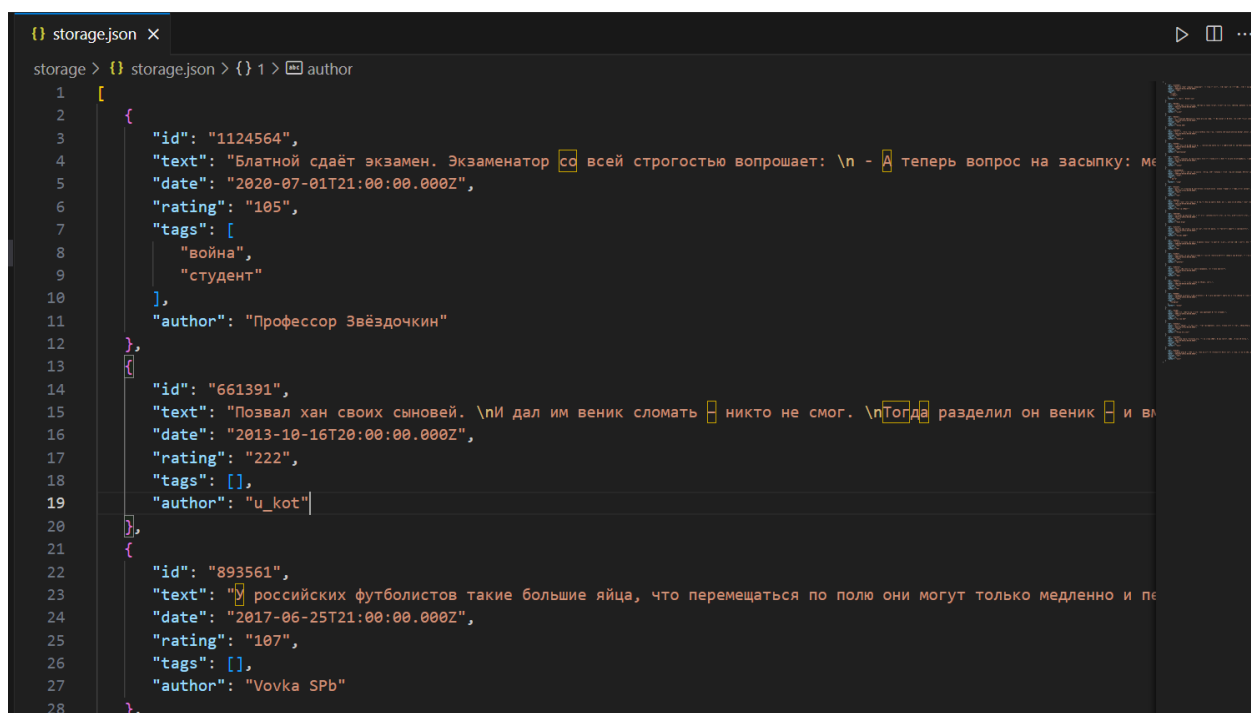


```
Командная строка  
C:\Users\Богдан\Desktop\Консорцеум Кодекс\Стажировка\task_2>node main  
C:\Users\Богдан\Desktop\Консорцеум Кодекс\Стажировка\task_2>
```

Как можно заметить – программа выполнена. Это значит, что в корне проекта должна появиться директория storage с файлом storage.json:



Как можно заметить – все есть. Теперь осталось проверить содержимое файла storage.json:



```

29 {
30   "id": "1292930",
31   "text": "Говорят, одной очень дерзкой девушке один очень солидный мужчина предложил поехать пожить на тропиче
32   "date": "2022-02-03T21:00:00.000Z",
33   "rating": "263",
34   "tags": [],
35   "author": "evgen_d"
36 },
37 {
38   "id": "334582",
39   "text": "- На одном из предприятий ЧП. Произошел аварийный сброс и наполовину \n затопил близлежащее село.
40   "date": "2008-03-04T21:00:00.000Z",
41   "rating": "190",
42   "tags": [],
43   "author": "qwertyuiop"
44 },
45 {
46   "id": "724003",
47   "text": "Хочешь выглядеть профессионалом – просто всегда говори «Ну это же было предсказуемо», сразу после то
48   "date": "2014-10-09T20:00:00.000Z",
49   "rating": "95",
50   "tags": [],
51   "author": "jonas"
52 },

```

```

53 {
54   "id": "-1040600010",
55   "text": "Сидят четверо мужчин в приемной роддома, ждут сведений о своих рожаящих \npженах. Выходит медсестра и
56   "date": "1999-04-05T20:00:00.000Z",
57   "rating": "1134",
58   "tags": [
59     "работа"
60   ],
61   "author": "Лиза"
62 },
63 {
64   "id": "872226",
65   "text": "Читаю: \"Российские производители молока приняли решение отказаться от пальмового масла\"... \nА я д
66   "date": "2017-02-17T21:00:00.000Z",
67   "rating": "331",
68   "tags": [],
69   "author": "tet"
70 },
71 {
72   "id": "603938",
73   "text": "- Эти электронные сигареты продаются уже на каждом углу. Скоро, наверное придумают электронные наркот
74   "date": "2012-10-07T20:00:00.000Z",
75   "rating": "275",
76   "tags": [],
77   "author": "Дед Марихуаныч"
78 },

```

```

79 {
80   "id": "1067368",
81   "text": "Лесникам на заметку: \nПри встрече с медведем притворитесь мертвым, а потом притворитесь, что вдруг
82   "date": "2019-12-20T21:00:00.000Z",
83   "rating": "133",
84   "tags": [],
85   "author": "Vash drug"
86 },
87 {
88   "id": "1151724",
89   "text": "Человек как гвоздь: если его бить долго по башке, он становится забитым и малозаметным",
90   "date": "2020-10-19T21:00:00.000Z",
91   "rating": "173",
92   "tags": [],
93   "author": "ЖУЛЬЕН СТЕБО"
94 },
95 {
96   "id": "362615",
97   "text": "Ученые выяснили, почему в Бразилии снимают сериалы про любовь, а у нас \nпро ментов. Где что хорошо
98   "date": "2008-09-20T20:00:00.000Z",
99   "rating": "261",
100   "tags": [],
101   "author": "ЖЖ"
102 },

```

```

103 {
104   "id": "213023",
105   "text": "Дедовщина в армии существовала во время Бородинской Битвы!!! \nИначе как понимать строчки Лермонтова",
106   "date": "2006-02-25T21:00:00.000Z",
107   "rating": "148",
108   "tags": [],
109   "author": "pfeffer"
110 },
111 {
112   "id": "1303723",
113   "text": "Лишить мультиков серьёзное наказание, оно вообще законно?",
114   "date": "2022-03-01T21:00:00.000Z",
115   "rating": "159",
116   "tags": [],
117   "author": "Dea"
118 },
119 {
120   "id": "302823",
121   "text": "Детство - это время, когда не думаешь матом.",
122   "date": "2007-08-09T20:00:00.000Z",
123   "rating": "278",
124   "tags": [],
125   "author": "ЖЖЖ"
126 },

```

```

127 {
128   "id": "949006",
129   "text": "Telegram обратился к пользователям с просьбой наконец-то найти для Яровой мужа хорошего.",
130   "date": "2018-05-18T21:00:00.000Z",
131   "rating": "379",
132   "tags": [
133     "telegram"
134   ],
135   "author": "kleon"
136 },
137 {
138   "id": "74106",
139   "text": "2041 год. \nБитва за Москву - азербайджанцы против китайцев.",
140   "date": "2003-12-29T21:00:00.000Z",
141   "rating": "147",
142   "tags": [],
143   "author": "Сергей Шах"
144 },
145 {
146   "id": "1443611",
147   "text": "Кто-то думает, что для того, чтобы самозабвенно, много, долгие годы воровать, нужна удача. Другие -
148   "date": "2024-02-17T21:00:00.000Z",
149   "rating": "219",
150   "tags": [],
151   "author": "Авгий Конюшнев"
152 },

```

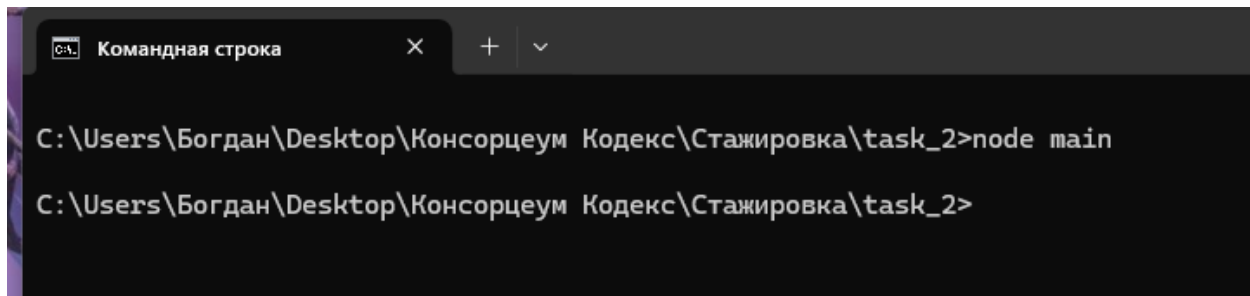
```

153 {
154   "id": "1347830",
155   "text": "Аня была такой стеснительной, что не могла купить презервативы. Теперь у неё 10 детей.",
156   "date": "2022-09-08T21:00:00.000Z",
157   "rating": "81",
158   "tags": [],
159   "author": "null"
160 },
161 {
162   "id": "816183",
163   "text": "Сказка была про двух скряг, один из которых одолжил у другого шесть грошей. Но не вернул, а умер. И
164   "date": "2016-03-20T21:00:00.000Z",
165   "rating": "134",
166   "tags": [],
167   "author": "null"
168 }
169 ]

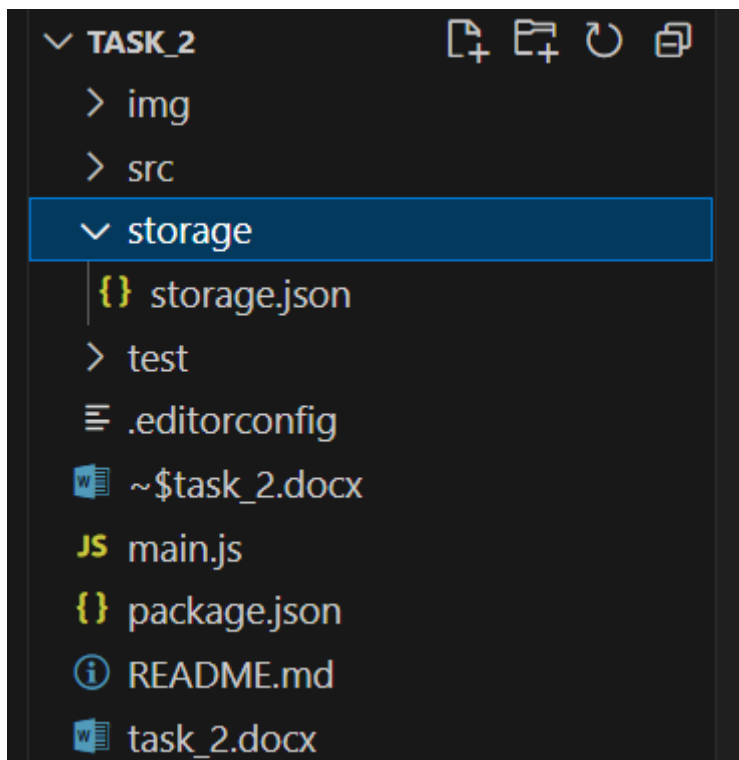
```

Как можно заметить – весь формат верный и число анекдотов полное – 20.

-Теперь запустим программу еще раз, чтобы убедиться, что не будет создано новой директории storage или нового файла storage.json. Все должно отразиться в той же директории и в том же файле, также storage.json – должен обновить свое содержимое:



```
Командная строка
C:\Users\Богдан\Desktop\Консорцеум Кодекс\Стажировка\task_2>node main
C:\Users\Богдан\Desktop\Консорцеум Кодекс\Стажировка\task_2>
```




```

57 {
58   "id": "392841",
59   "text": "На трассе в пять утра. \n- Девушка, вы ночная бабочка? \n- Нет, бл@, утренняя звезда!",
60   "date": "2009-04-16T20:00:00.000Z",
61   "rating": "606",
62   "tags": [],
63   "author": "Volk"
64 },
65 {
66   "id": "1036947",
67   "text": "- Как сказал Лесалль: «Если гусар дожил до 30-ти, то это говно, а не гусар!» \n- Мама, а других кав
68   "date": "2019-08-10T21:00:00.000Z",
69   "rating": "188",
70   "tags": [],
71   "author": "Гексоген"
72 },
73 {
74   "id": "592354",
75   "text": "- Иван Иванович, а Вы что молчите? Выскажите своё мнение. \n- Спасибо, нет. Ещё скажу какую-нибудь гл
76   "date": "2012-07-28T20:00:00.000Z",
77   "rating": "267",
78   "tags": [],
79   "author": "Иосиф Египетский"
80 },

```

```

81 {
82   "id": "486000",
83   "text": "- Папачка, дай мне 200 рублей, - говорит сын. \n- А мне - 500, - просит дочь. \n- А мне нужно 2000,
84   "date": "2010-12-23T21:00:00.000Z",
85   "rating": "318",
86   "tags": [],
87   "author": "Леонид Хлыновский"
88 },
89 {
90   "id": "624823",
91   "text": "Челябинский никелевый завод настолько суров, что заказывает поставки руды метеоритами.",
92   "date": "2013-02-15T20:00:00.000Z",
93   "rating": "228",
94   "tags": [
95     "Челябинск"
96   ],
97   "author": "Asket"
98 },
99 {
100   "id": "-10080410",
101   "text": "Звонок в дверь. Хозяин открывает, на пороге - гости. \n- Да вы хоть бы предупредили, - упрекает их
102   "date": "2000-12-27T21:00:00.000Z",
103   "rating": "1235",
104   "tags": [],
105   "author": "0a"
106 },

```

```

107 {
108   "id": "189563",
109   "text": "Труд сделал из обезьяны человека и надо же такому случиться, что именно \n - мой начальник.",
110   "date": "2005-10-18T20:00:00.000Z",
111   "rating": "280",
112   "tags": [],
113   "author": "BA3"
114 },
115 {
116   "id": "611234",
117   "text": "Самый лучший подарок тот, после которого говоришь \"Да вы че, с ума сошли?!\"",
118   "date": "2012-11-22T20:00:00.000Z",
119   "rating": "235",
120   "tags": [],
121   "author": "Al K"
122 },
123 {
124   "id": "770270",
125   "text": "Как взрослая самостоятельная независимая женщина, я хочу лечь на пол и рыдать до тех пор, пока всё
126   "date": "2015-07-01T21:00:00.000Z",
127   "rating": "285",
128   "tags": [
129     "девушки"
130   ],
131   "author": "null"
132 },

```

```

133 {
134   "id": "-42400007",
135   "text": "Одна женщина спрашивает у профессора психологии: \n- доктор, а скажите, как вы определяете, нормаль
136   "date": "1998-04-23T20:00:00.000Z",
137   "rating": "625",
138   "tags": [],
139   "author": "Roman Pavlov"
140 },
141 {
142   "id": "765039",
143   "text": "Развитие ИТ привело к тому, что все разговоры, проходившие ранее на кухнях коммунальных квартир, те
144   "date": "2015-05-27T21:00:00.000Z",
145   "rating": "154",
146   "tags": [],
147   "author": "RRaf"
148 },
149 {
150   "id": "1113216",
151   "text": "При разводе всё имущество делится по справедливости, 90% - жене и 10% - её адвокату.",
152   "date": "2020-05-17T21:00:00.000Z",
153   "rating": "191",
154   "tags": [],
155   "author": "Сафронов Николай"
156 },

```

```

157 {
158   "id": "779053",
159   "text": "Я внезапно осознал, что проебал это лето, но потом вспомнил, что я проебал уже половину своей жизни,
160   "date": "2015-08-27T21:00:00.000Z",
161   "rating": "223",
162   "tags": [
163     "жизни"
164   ],
165   "author": "null"
166 },
167 {
168   "id": "1210091",
169   "text": "Мне очень нравится поговорка: «Как станет хуже некуда, так и на лад пойдет». По временам я спрашиваю
170   "date": "2021-05-03T21:00:00.000Z",
171   "rating": "105",
172   "tags": [],
173   "author": "VY"
174 }
175 ]

```

Как можно заметить – все работает правильно

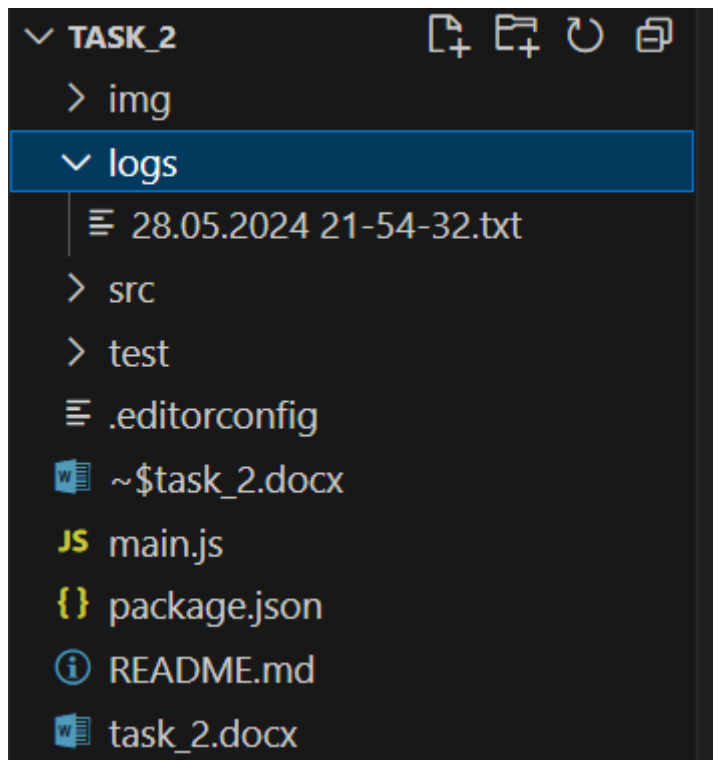
- б. С ошибками (перед этим удалим создавшуюся директорию storage в корне проекта):

-Добавим ошибку в url – вместо строки, отправим число:

```
JS main.js ×
JS main.js > ...
1  import path from 'path';
2  import logErrorToFile from './src/logger/errorLogger/errorLogger.js';
3  import storyService from './src/domain/story/storyService.js';
4  import {
5    writeFile,
6    ensureDirectoryExists,
7    clearFile
8  } from './src/helpers/file/fileService.js';
9
10 const URL = 35;
11 const STORAGE_DIR = './storage';
12 const STORAGE_FILE_NAME = 'storage.json';
13
14 main().catch((err) => {
15   logErrorToFile(err);
16 })
```

```
Командная строка × + ▾
C:\Users\Богдан\Desktop\Консорцеум Кодекс\Стажировка\task_2>node main
C:\Users\Богдан\Desktop\Консорцеум Кодекс\Стажировка\task_2>
```

Программа выполнилась. Но она должна была выполняться с ошибкой, это значит, что в корне проекта должна появиться директория logs с файлом dd.mm.yy.hh:mm:ss.txt и при этом, в корне проекта не должно быть директории storage:



В logs появился соответствующий файл, и в корне проекта отсутствует storage.
Значит, все верно. Осталось проверить содержимое файла dd.mm.yy. hh:mm:ss.txt:

```
logs > 28.05.2024 21-54-32.txt
1  28.05.2024 21:54 - Error:
2    CustomError: Url must be a string!
3    SystemError: at getHTML (task_2/src/domain/htmlProcessing/htmlProcessingService.js:12:11)
4    at htmlProcessingService (task_2/src/domain/htmlProcessing/htmlProcessingService.js:2:17)
5    at storyService (task_2/src/domain/story/storyService.js:11:22)
6    at main (task_2/main.js:19:27)
7    at task_2/main.js:14:1
8    at ModuleJob.run (node:internal/modules/esm/module_job:218:25)
9    at async ModuleLoader.import (node:internal/modules/esm/loader:329:24)
10   at async loadESM (node:internal/process/esm_loader:28:7)
11   at async handleMainPromise (node:internal/modules/run_main:113:12)
12
13
```

Данный файл содержит корректное содержимое.

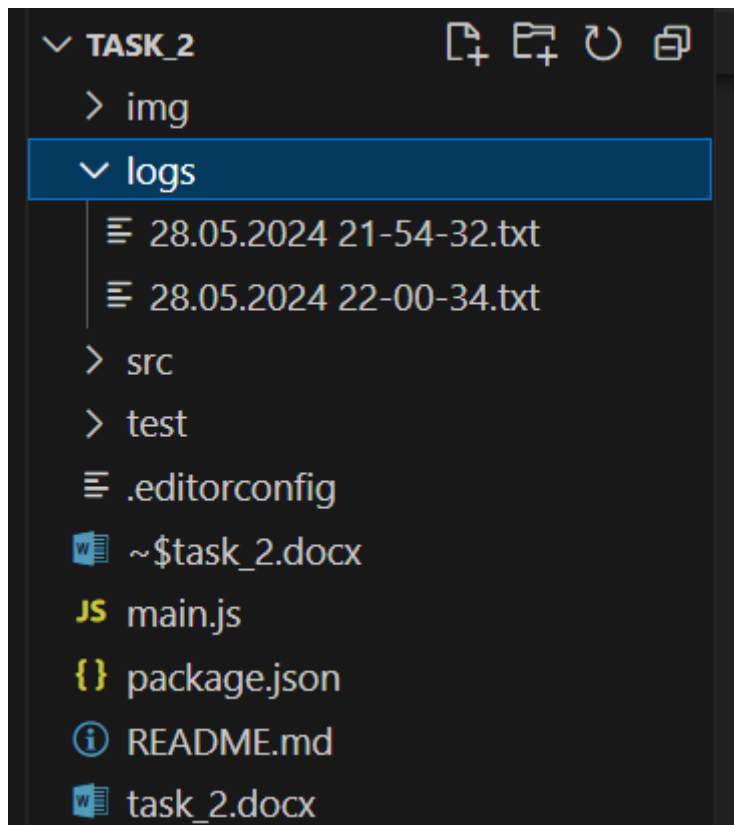
-Теперь вернем назад url и сделаем ошибку, передав вместо регулярного выражения – обычный объект, чтобы убедиться, что при следующей ошибке – в директории logs создастся новый файл с ошибкой:

```
JS main.js ×
JS main.js > ...
1  import path from 'path';
2  import logErrorToFile from './src/logger/errorLogger/errorLogger.js';
3  import storyService from './src/domain/story/storyService.js';
4  import {
5    writeFile,
6    ensureDirectoryExists,
7    clearFile
8  } from './src/helpers/file/fileService.js';
9
10 const URL = 'https://www.anekdot.ru/random/anekdot/';
11 const STORAGE_DIR = './storage';
12 const STORAGE_FILE_NAME = 'storage.json';
13
14 main().catch((err) => {
15   logErrorToFile(err);
16 })
```

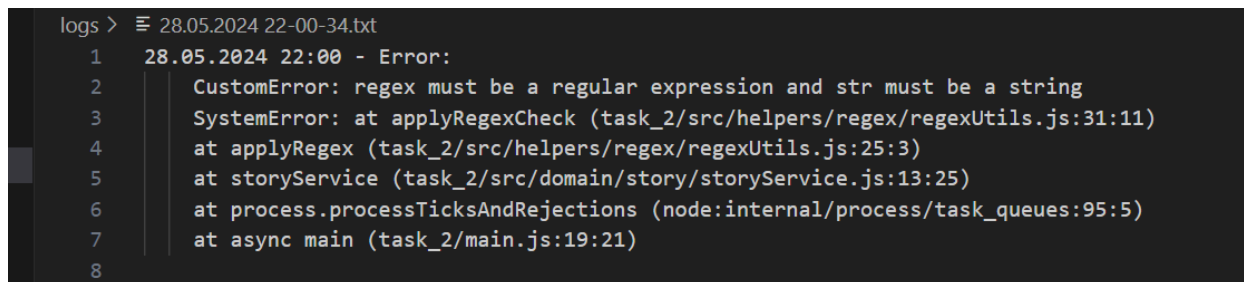
```
const htmlDataChunk = applyRegex(
  {},
  HTML
);
```

```
Командная строка
C:\Users\Богдан\Desktop\Консорцеум Кодекс\Стажировка\task_2>node main
C:\Users\Богдан\Desktop\Консорцеум Кодекс\Стажировка\task_2>
```

Программа выполнилась, снова проверяем корень проекта и logs:



Директория storage отсутствует, а в logs создался новый файл. Значит, все верно. Осталось проверить содержимое нового файла:



Новый файл содержит корректное содержимое. Также проверим, что в предыдущий файл не было ничего записано нового:

```
logs > 28.05.2024 21-54-32.txt
1 28.05.2024 21:54 - Error:
2   CustomError: Url must be a string!
3   SystemError: at getHTML (task_2/src/domain/htmlProcessing/htmlProcessingService.js:12:11)
4     at htmlProcessingService (task_2/src/domain/htmlProcessing/htmlProcessingService.js:2:17)
5     at storyService (task_2/src/domain/story/storyService.js:11:22)
6     at main (task_2/main.js:19:27)
7     at task_2/main.js:14:1
8     at ModuleJob.run (node:internal/modules/esm/module_job:218:25)
9     at async ModuleLoader.import (node:internal/modules/esm/loader:329:24)
10    at async loadESM (node:internal/process/esm_loader:28:7)
11    at async handleMainPromise (node:internal/modules/run_main:113:12)
12
13
```

В предыдущий файл ничего нового не записалось, значит все верно.