**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/](https://www.anekdot.ru/random/anekdot/%20) 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 вычленить 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/](https://www.anekdot.ru/random/anekdot/%20) 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:

1. Элемент p с классом title, который содержит ссылку, содержащую искомую дату публикации
2. Элемент div с классом text, который содержит искомый текст анекдота
3. Элемент div с классом tags, который содержит ссылки на теги, относящиеся к данному анекдоту
4. Элемент div с классом votingbox, который содержит элемент div с классом rates, который имеет атрибут data-r, где первое значение до двоеточия – искомый рейтинг
5. 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:

1. 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)
2. 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

1. /<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 <br/> в XHTML или XML. Однако, в HTML5 тег <br> может быть написан как <br> без слеша. Символ ? делает предыдущий символ (в данном случае /) необязательным, что позволяет регулярное выражение соответствовать как <br>, так и <br/>
* 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

1. /<[^>]\*>?/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:

1. 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

1. Функция applyRegexByFirstExec:

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

1. Функция applyRegex:

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

1. 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. 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. 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 {

  applyRegexByCleanedExec,

  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"[^>]\*>([\s\S]\*?(?=<[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">\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 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:

1. /<[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
2. ([\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
3. /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 вместо for

      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:

1. storyData – объект, представляющий собой определенный анекдот
2. 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

1. 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
2. storyData.date - ищется дата публикации анекдота с помощью регулярного выражения /<[a-z]+ class=”title”>\s\*<a href=”.+?”>(.\*?)<\/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\*<a href=”.+?”> – 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

1. 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

1. 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:
2. /<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

1. /<a href=”(.\*?)”>(.\*?)<\/a>/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
* <\/a> - ищет закрывающий тег a
* /g - флаг, который указывает на глобальный поиск, позволяя найти все совпадения в тексте, а не только первое

1. />(.\*?)</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)

1. 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

1. 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');

}

1. 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:

1. fs – модуль, необходимый для работы с файловой системой диска
2. appendFile – функция, которая предназначена для добавления текста в конец файла по указанному пути
3. writeToFile – функция, которая записывает в файл переданную запись
4. 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:
5. clearFile – функция, которая проверяет – содержит ли переданный файл какие-то записи. Если содержит, то функция очищает этот файл
6. 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('\n');

    location = locationLines.trim();

    const locationsArray = location.split('\n').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('\n');

  }

  const errMessage = `${getCurDate()}.${getCurMonth()}.${getCurYear()} ${getCurHours()}:${getCurMinutes()} - ${typeError}:

    CustomError: ${error.message}

    SystemError: ${location}

  `;

  appendFile(errorLogFilePath, errMessage);

}

export default logErrorToFile;

Здесь:

1. path – модуль, необходимый для работы с путями файловой системы диска
2. 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
3. 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>

1. 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 {

  writeToFile,

  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);

  writeToFile(storageFilePath, dataStorage);

}

Здесь:

1. path – модуль, необходимый для работы с путями файловой системы диска
2. URL – константа, представляющая url ресурса, с которым мы работаем
3. 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
4. STORAGE\_FILE\_NAME – константа, представляющая из себя имя самого JSON-файла по директории STORAGE\_DIR
5. 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);

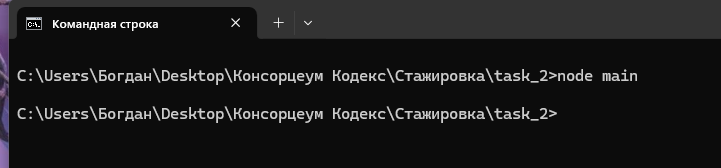
writeToFile(storageFilePath, dataStorage);

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

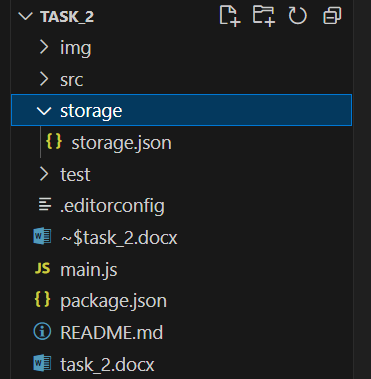
**Тесты:**

1. Без ошибок:

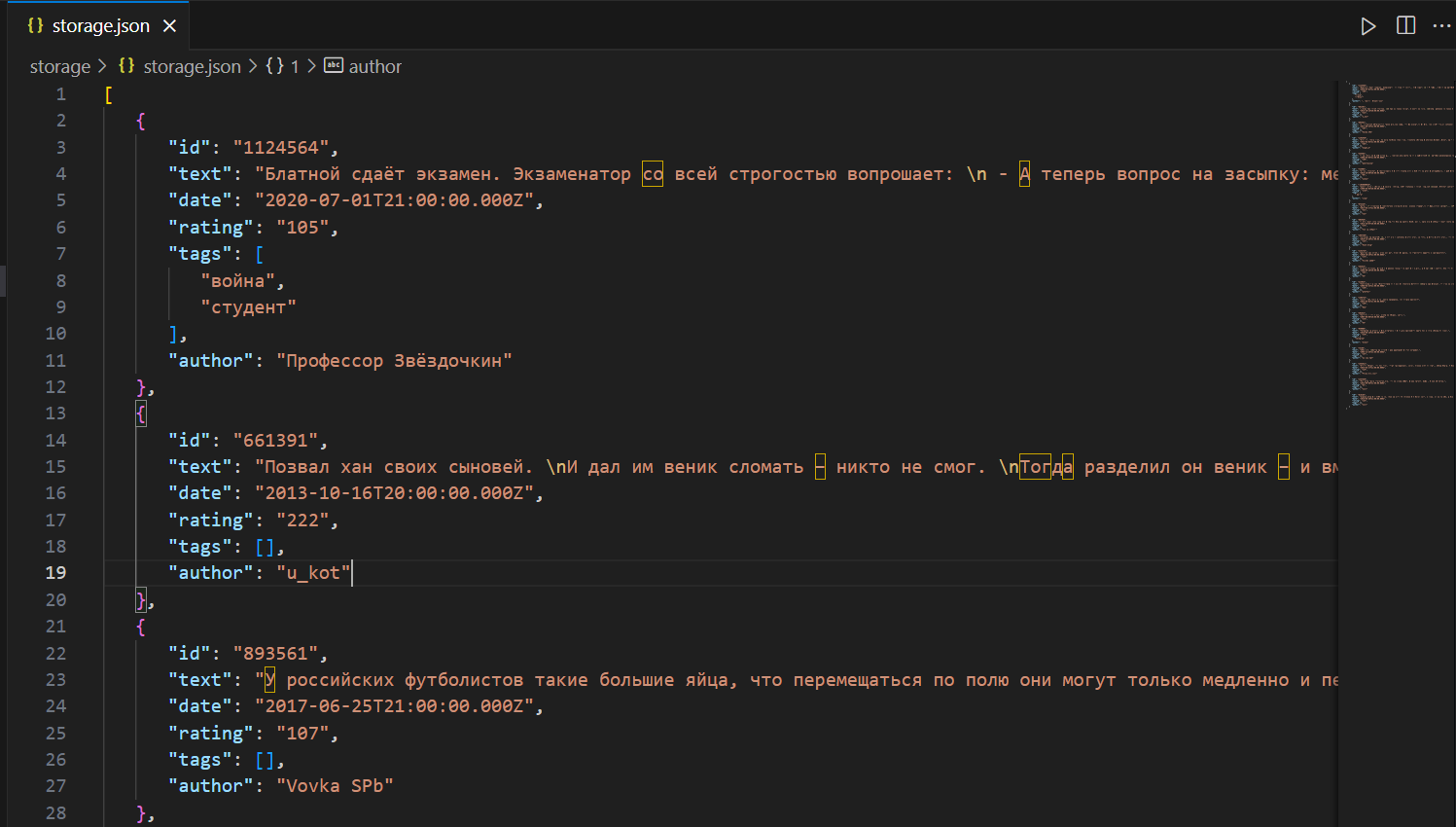
-Запустим программу:

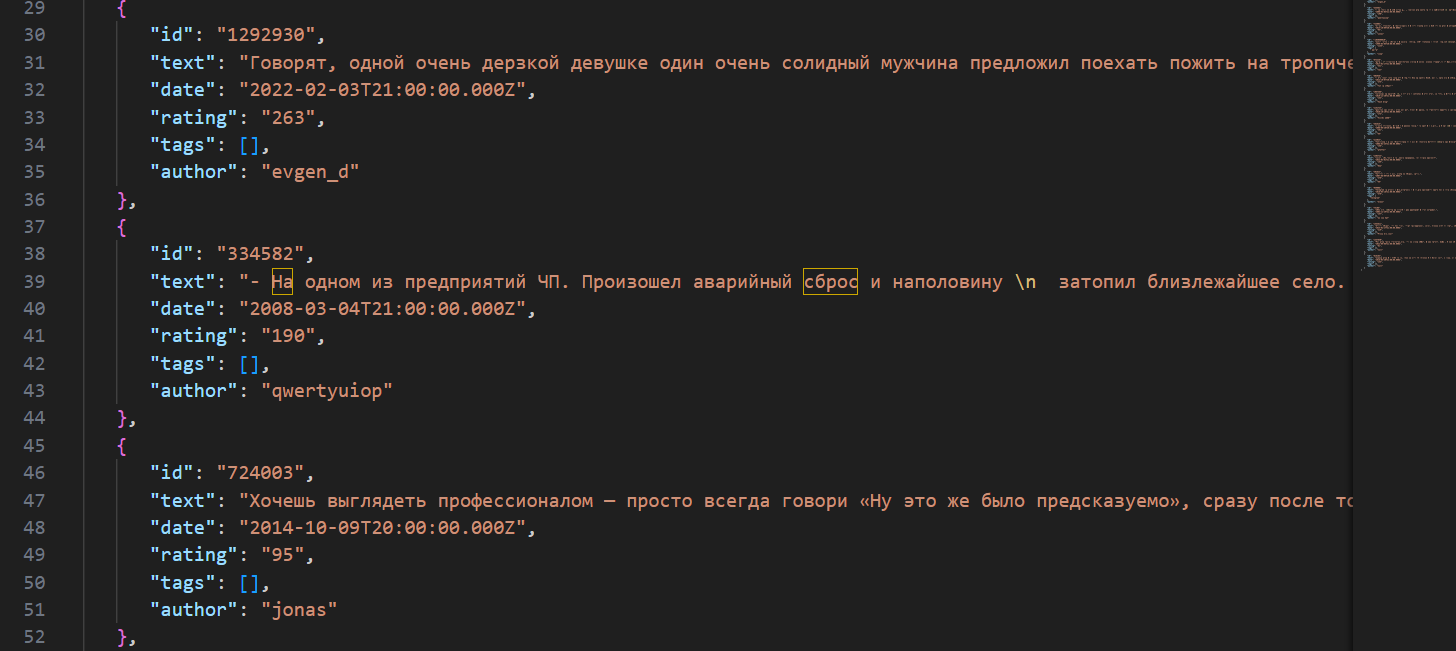


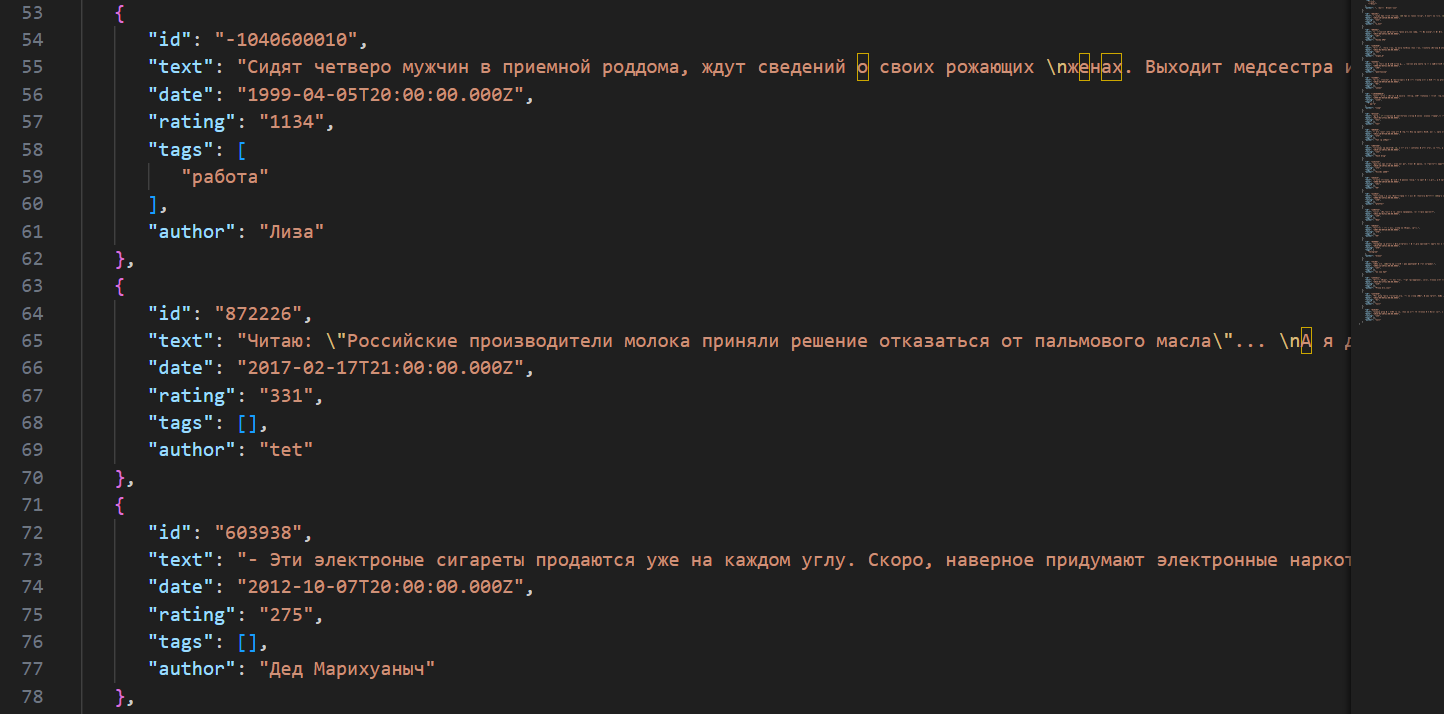
Как можно заметить – программа выполнилась. Это значит, что в корне проекта должна появиться директория storage с файлом storage.json:

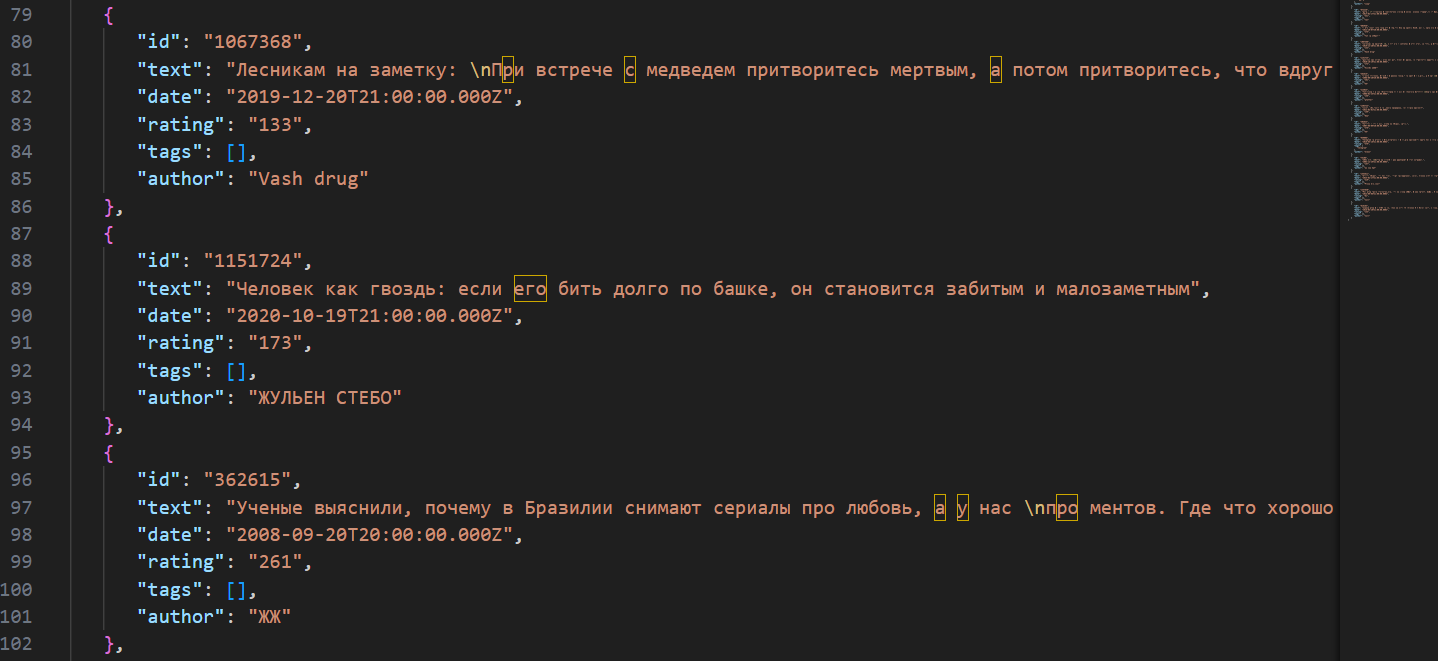


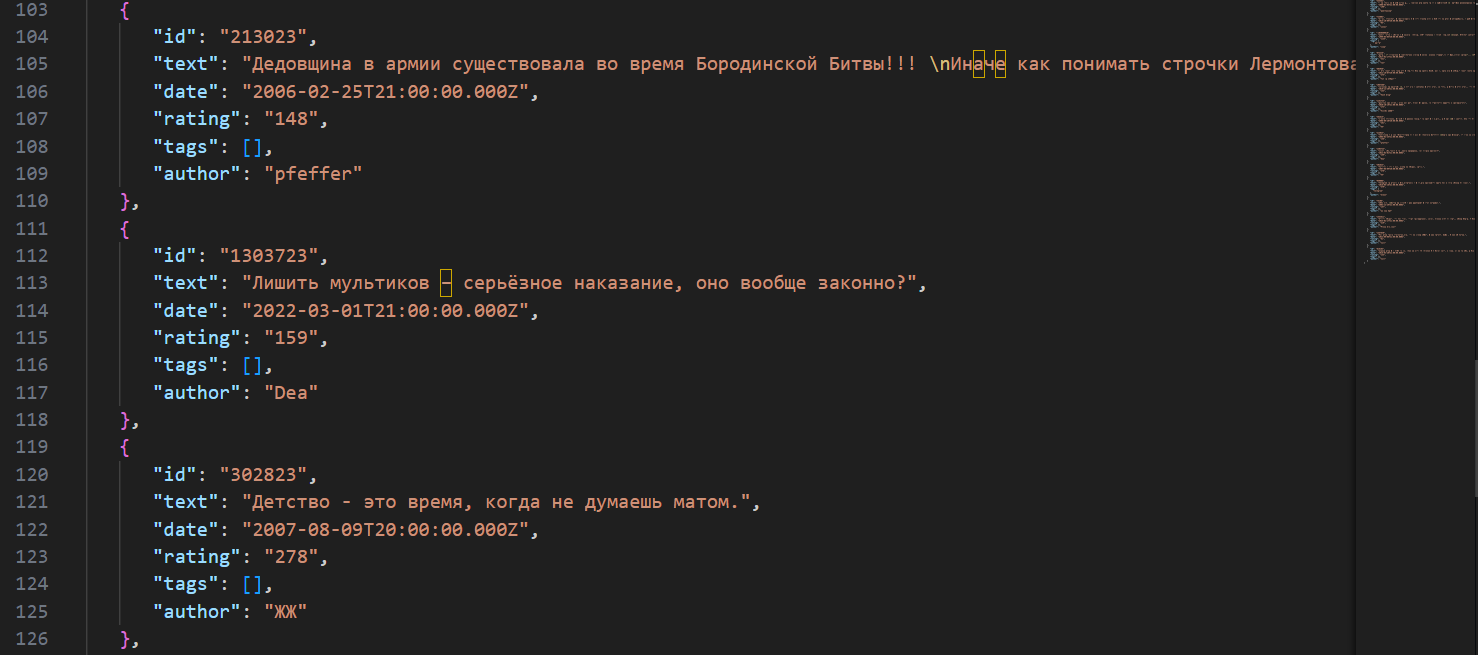
Как можно заметить – все есть. Теперь осталось проверить содержимое файла storage.json:

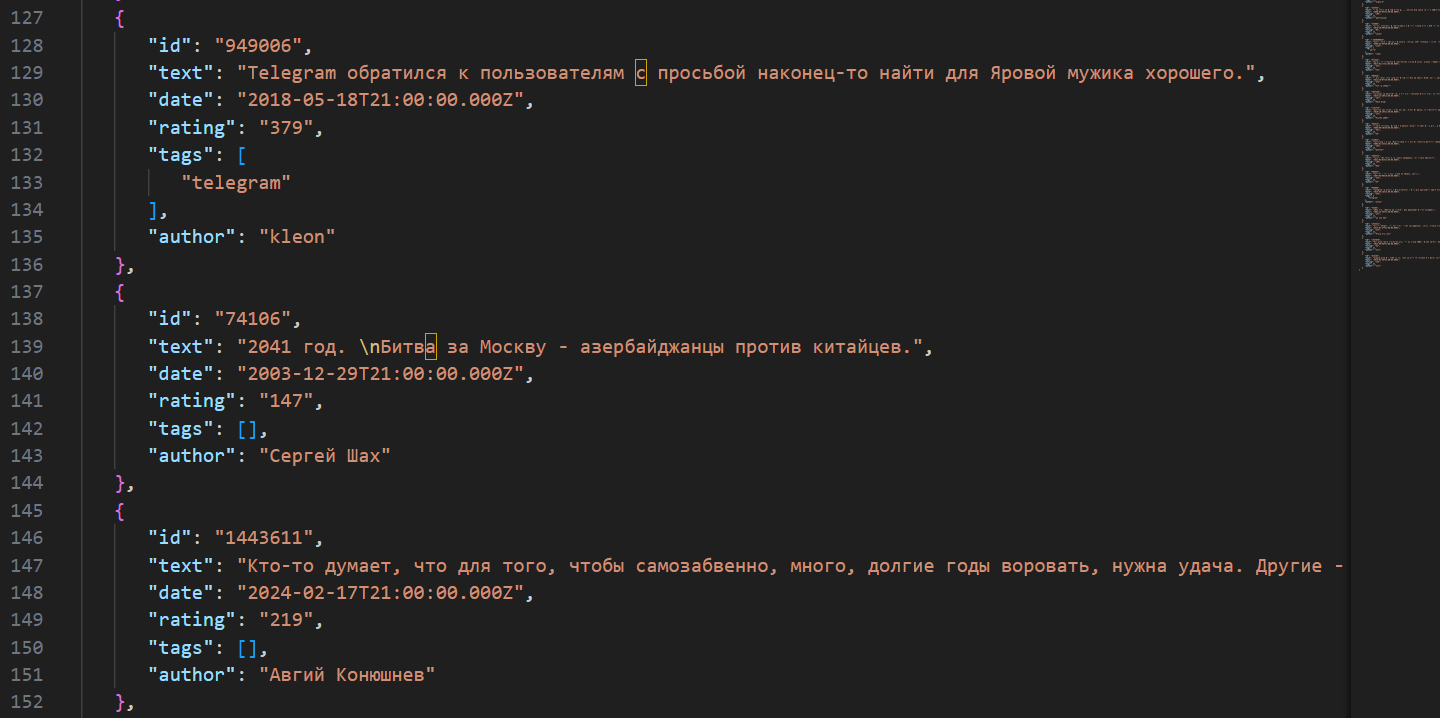


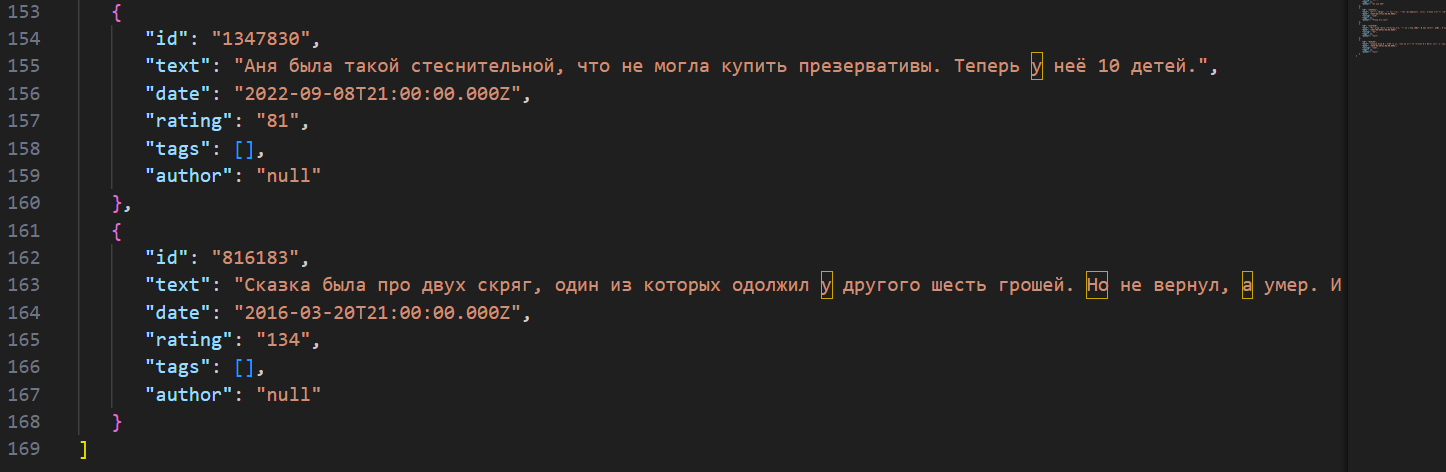






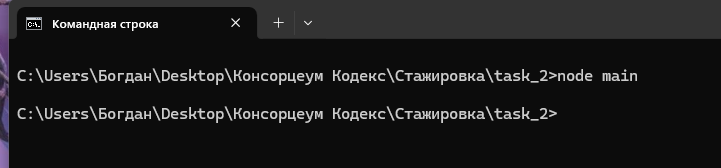


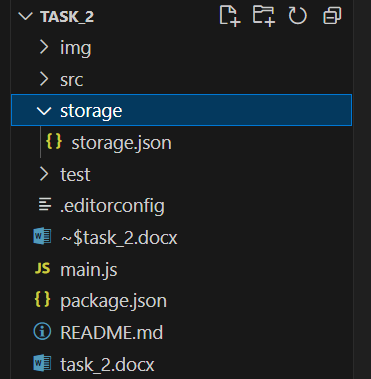


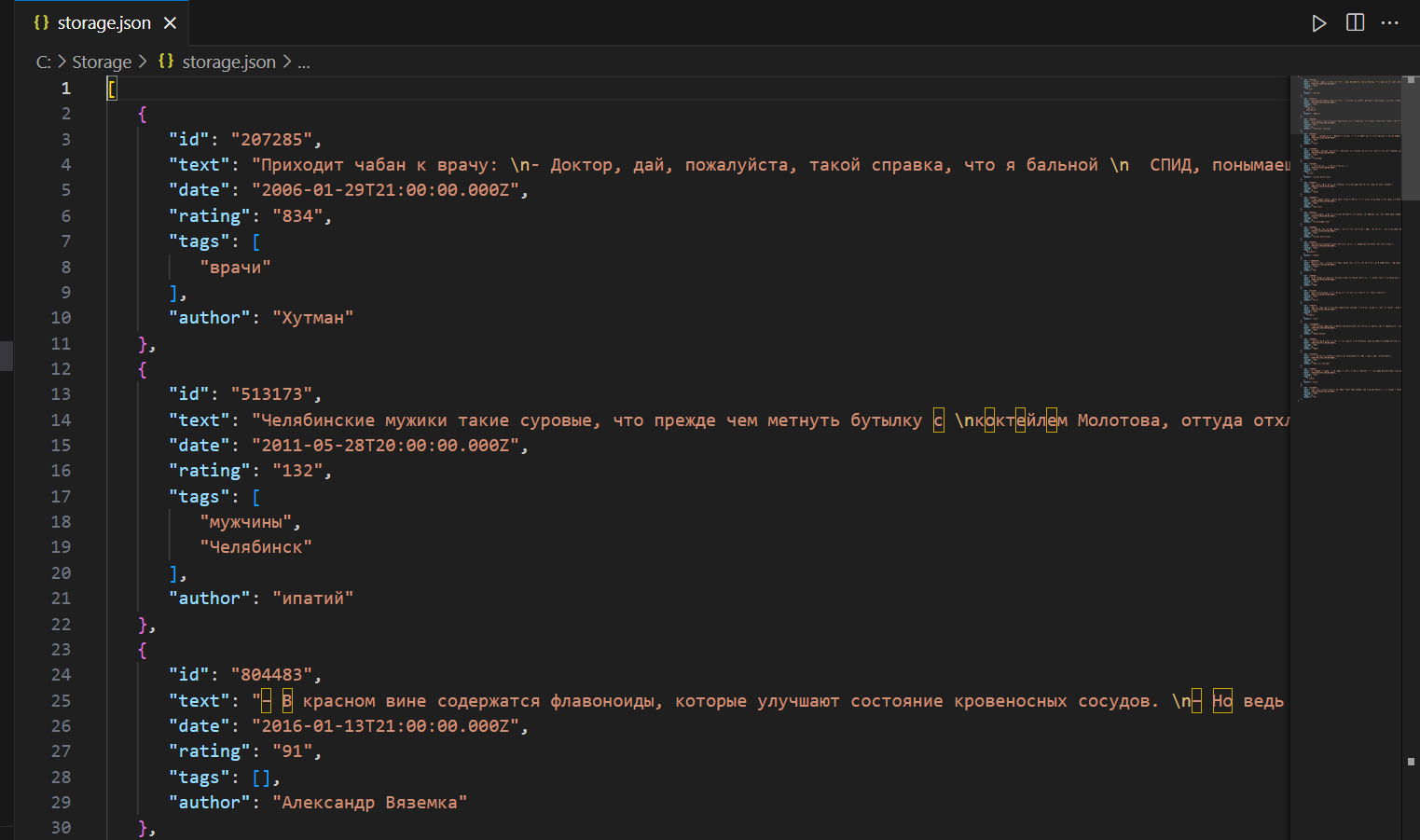


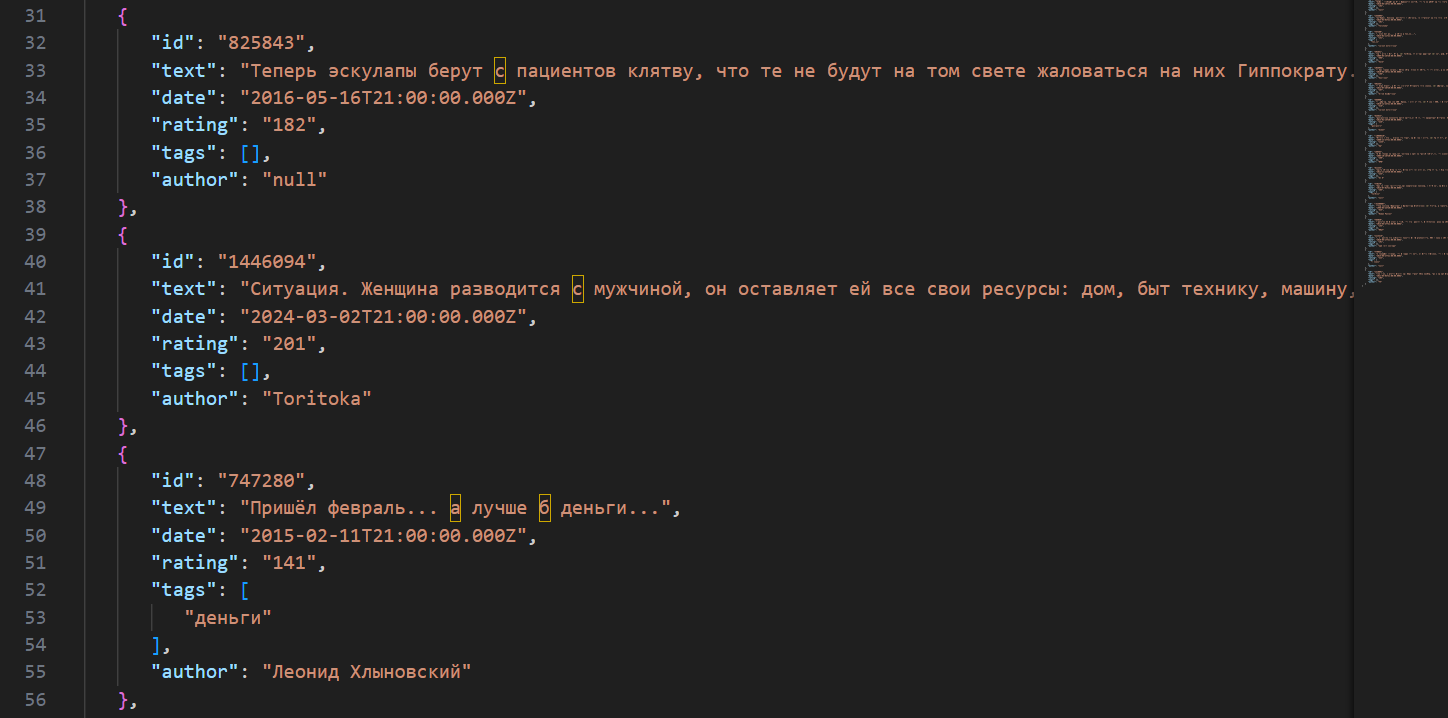
Как можно заметить – весь формат верный и число анекдотов полное – 20.

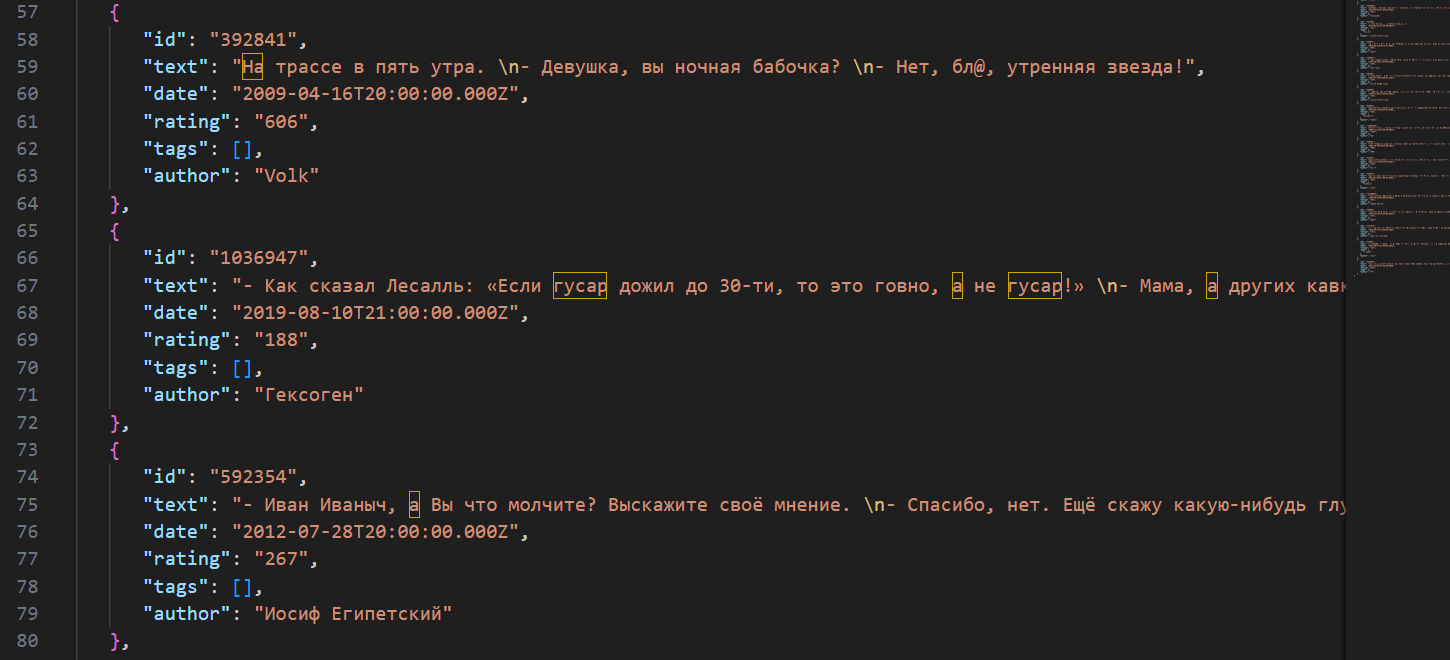
-Теперь запустим программу еще раз, чтобы убедиться, что не будет создано новой директории storage или нового файла storage.json. Все должно отразиться в той же директории и в том же файле, также storage.json – должен обновить свое содержимое:

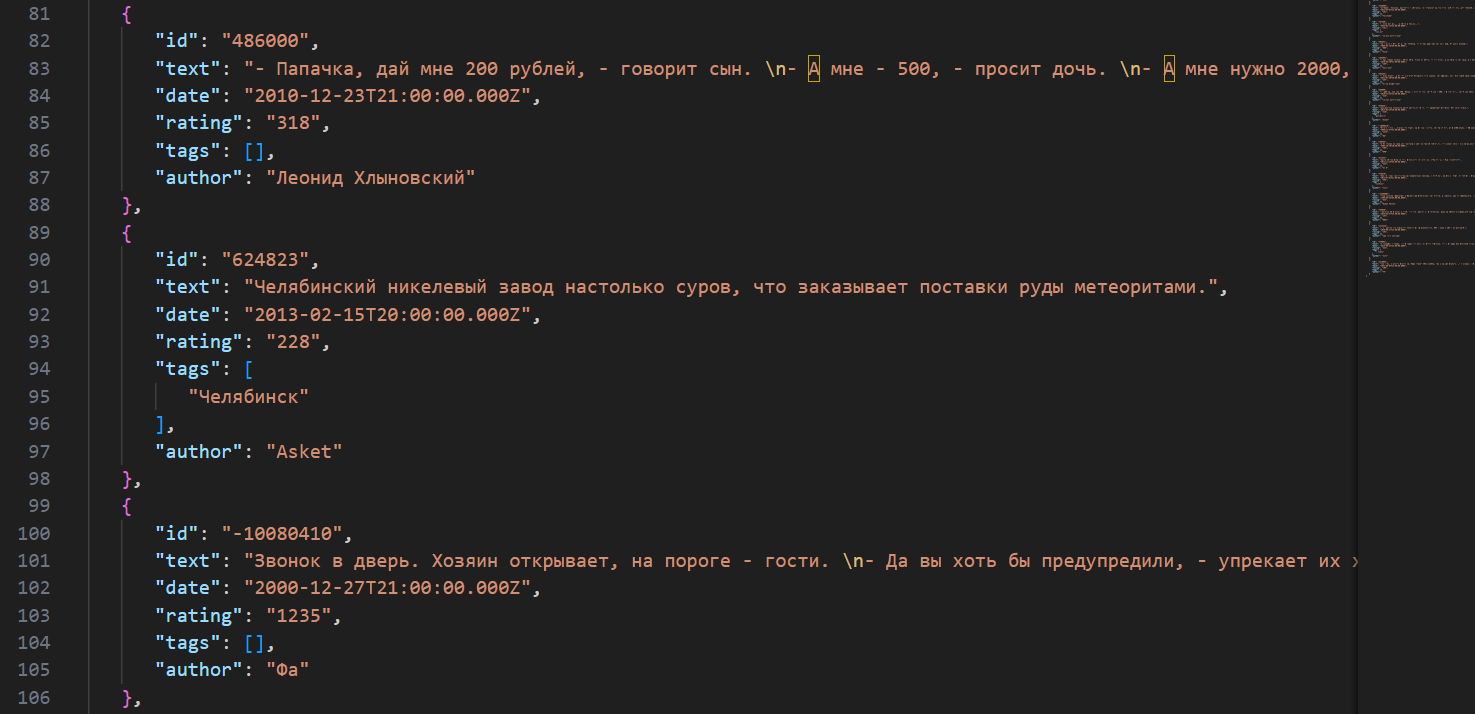


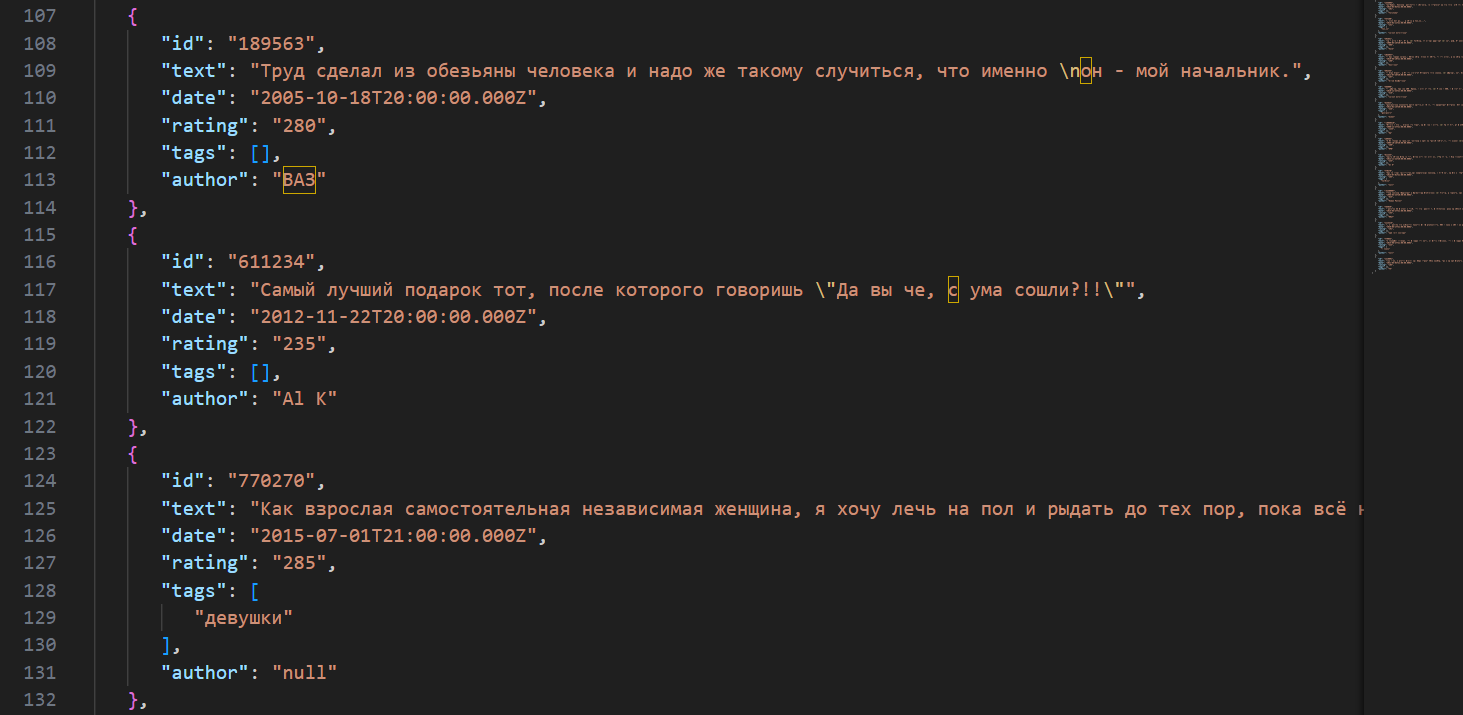


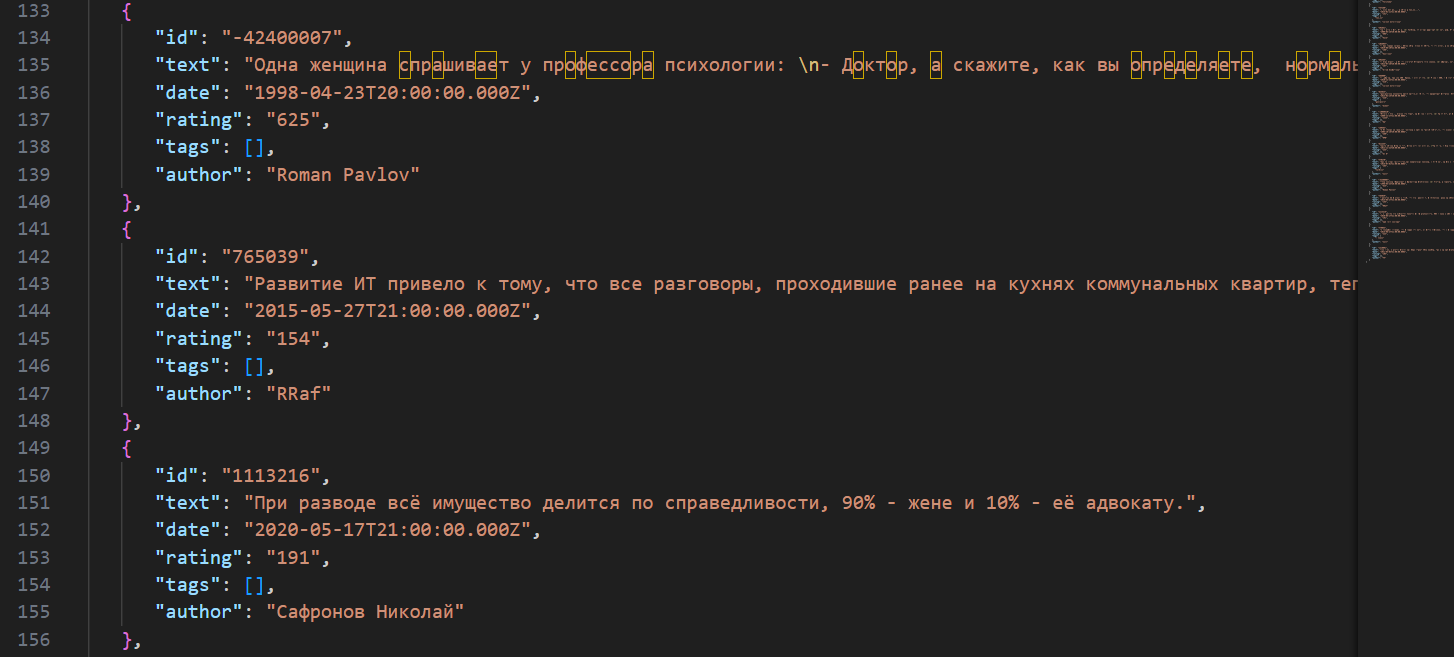


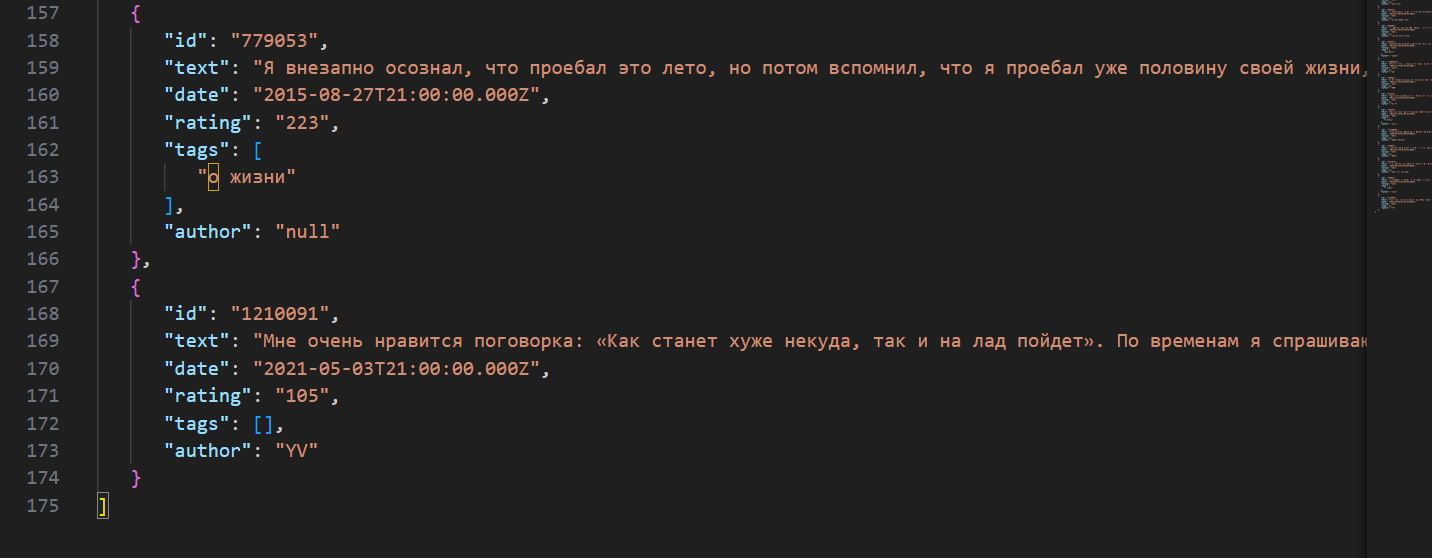








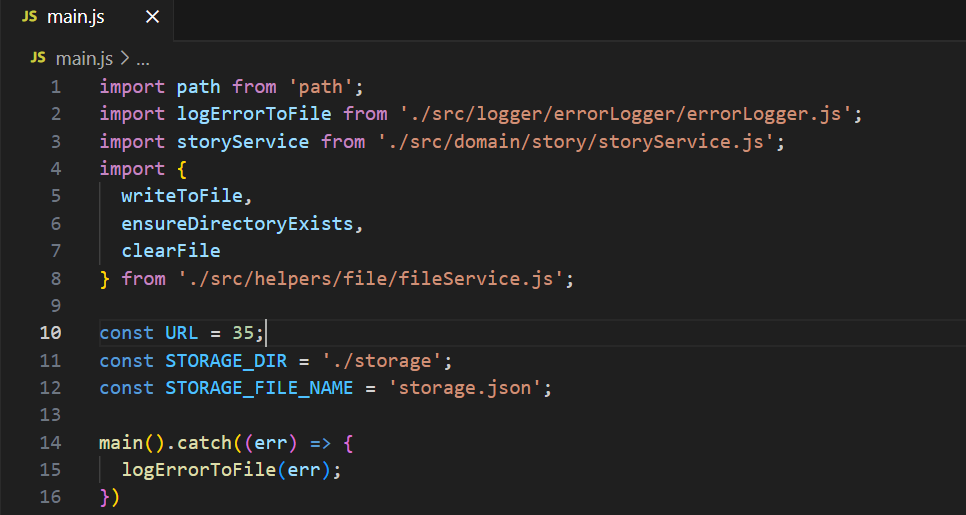


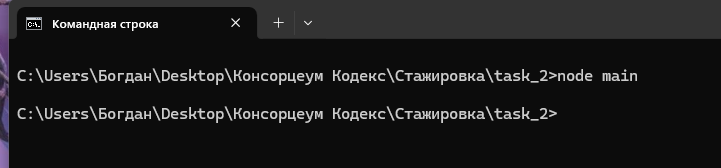


Как можно заметить – все работает правильно

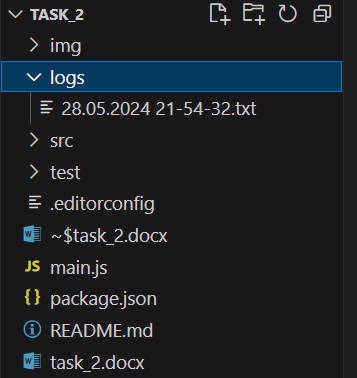
1. С ошибками (перед этим удалим создавшуюся директорию storage в корне проекта):

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

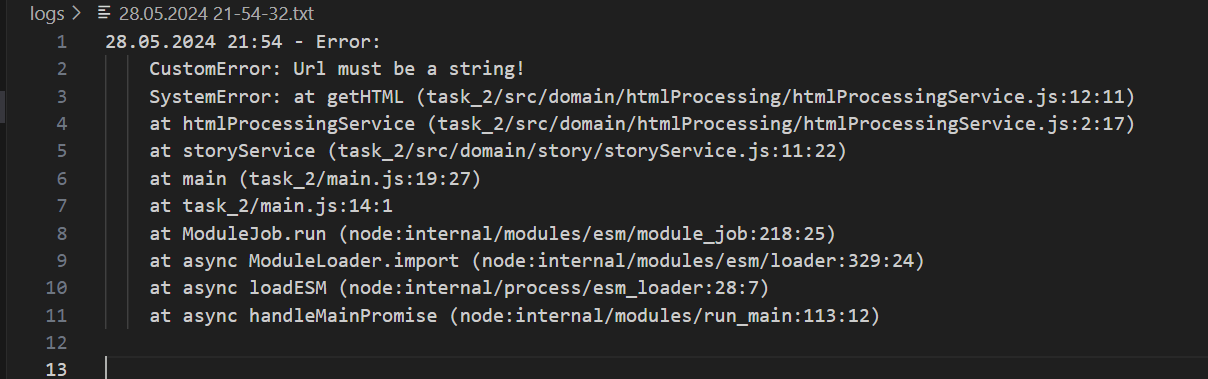




Программа выполнилась. Но она должна была выполниться с ошибкой, это значит, что в корне проекта должна появиться директория logs с файлом dd.mm.yy. hh:mm:ss.txt и при этом, в корне проекта не должно быть директории storage:

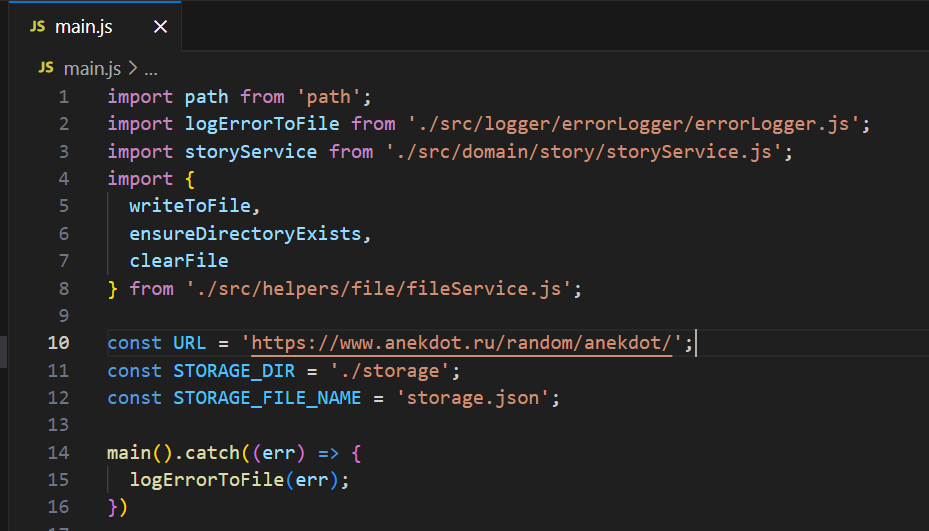


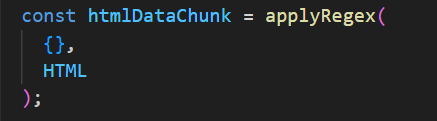
В logs появился соответствующий файл, и в корне проекта отсутствует storage. Значит, все верно. Осталось проверить содержимое файла dd.mm.yy. hh:mm:ss.txt:

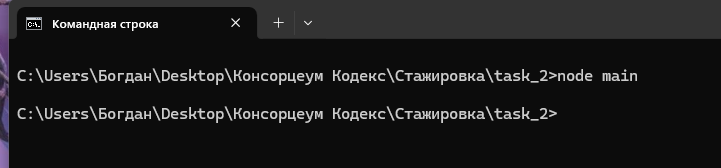


Данный файл содержит корректное содержимое.

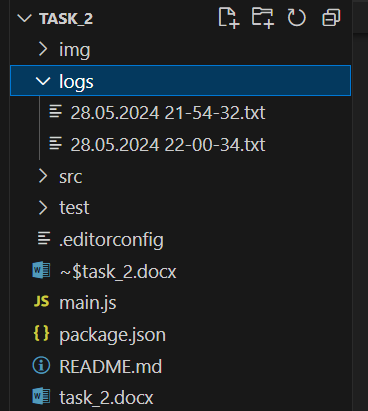
-Теперь вернем назад url и сделаем ошибку, передав вместо регулярного выражения – обычный объект, чтобы убедиться, что при следующей ошибке – в директории logs создастся новый файл с ошибкой:



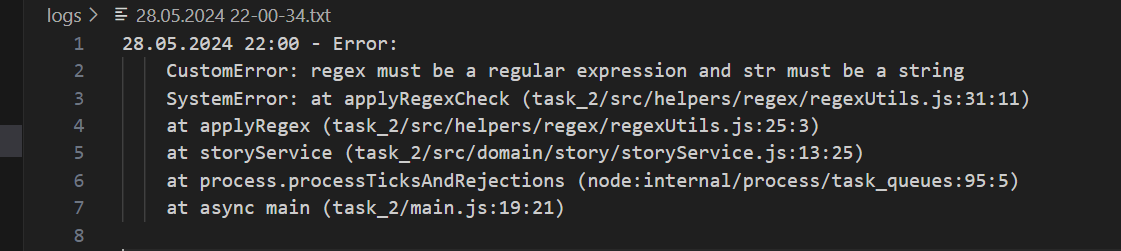




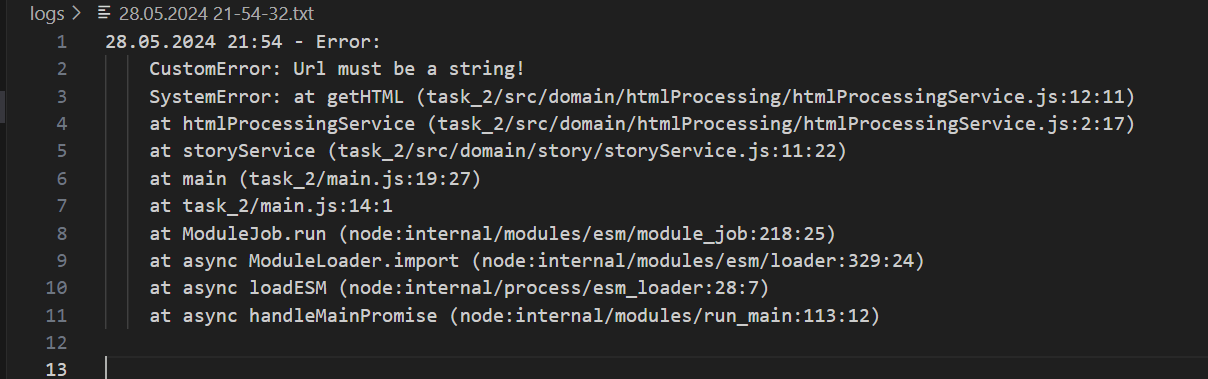
Программа выполнилась, снова проверяем корень проекта и logs:



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:



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