**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

Итоговые JSON-файлы - ./storage/storage\_1.json и ./storage/storage\_2.json

Файлы ошибок - ./logger/27.05.2024 14-17-35.txt и ./logger/27.05.2024 14-24-07.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. Сначала подключим необходимые модули, а также установим константы:

const FS = require('fs');

const PATH = require('path');

const URL = 'https://www.anekdot.ru/random/anekdot/';

const STORAGE\_DIR = 'C:/Storage';

const STORAGE\_FILE\_NAME = 'storage.json';

const LOGGER\_DIR = 'C:/Logger';

const storyDataModel = {

  id: null,

  text: null,

  date: null,

  rating: null,

  tags: [],

  author: null

};

let dataStorage = [];

Здесь:

1. FS – модуль, необходимый для работы с файловой системой диска
2. PATH – модуль, необходимый для работы с путями файловой системы диска
3. URL – константа, представляющая url ресурса, с которым мы работаем
4. 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 на диске C
5. 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 на диске C
6. STORAGE\_FILE\_NAME – константа, представляющая из себя имя самого JSON-файла по директории STORAGE\_DIR
7. LOGGER\_DIR - константа, которая содержит путь до директории, в которую будут записываться все файлы .txt, содержащие ошибки. В данном случае – это директория Logger на диске C
8. storyDataModel – объект, представляющий модель каждого анекдота, который будет записан в JSON-файл. Его можно воспринимать как своеобразный интерфейс, или класс-модель
9. dataStorage – массив, который будет содержать объекты анекдотов, который в итоге будет записан в JSON-файл

Далее запустим основную функцию main, которая является точкой входа. Сначала нужно сделать GET-запрос на заданный url и преобразовать ответ в формат text, чтобы получить html-код страницы по указанному url. Это выполняется в асинхронной функции getHTML:

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());

}

Перед тем, как переходить к разбору полученного html-кода, напишем три функции, которые позволят применять к строкам регулярные выражения:

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

function applyRegexByCleanedExec(regex, html) {

  applyRegexCheck(regex, html);

  let matches = [];

  let match;

  while ((match = regex.exec(html)) !== null) {

    const cleanedMatch = match[0].replace(/\s+/g, ' ').trim();

    matches.push(cleanedMatch);

  }

  return matches;

}

1. Функция applyRegexByFirstExec:

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

function applyRegexByFirstExec(regex, html) {

  applyRegexCheck(regex, html);

  const result = regex.exec(html);

  if (result && result[1]) {

    return result[1];

  }

}

1. Функция applyRegex:

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

function applyRegex(regex, html) {

  applyRegexCheck(regex, html);

  return html.match(regex);

}

1. Функция applyRegexCheck:

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

function applyRegexCheck(regex, html) {

  if (!(regex instanceof RegExp) || typeof html !== 'string') {

    throw new Error('regex must be a regular expression and html must be a string');

  }

}

После получения html-кода страницы – нужно получить каждый анекдот и записать его в dataStorage в виде объекта storyDataModel. 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 dataChunk = applyRegex(

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

    HTML

  );

Здесь с помощью регулярного выражения, в dataChunk 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 dataChunk, нужно пройти по каждому анекдоту, получить из него данные и записать их в специальный объект, а в конце добавить этот объект в dataStorage и после перейти к следующему анекдоту:

for (let i = 0; i < dataChunk.length; i++) {

    const data = { ...storyDataModel };

    data.id = applyRegexByFirstExec(/data-id="([^"]+)"/, dataChunk[i]);

    if (!data.id) {

      continue;

    }

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

    const dataDate = applyRegexByFirstExec(

      /<[a-z]+ class="title">\s\*<a href=".+?">(.\*?)<\/a>\s\*<\/[a-z]+>/, dataChunk[i]

    );

    const [dataDateDay, dataDateMonth, dataDateYear] = dataDate.split('.').map(Number);

    data.date = new Date(`${dataDateMonth}.${dataDateDay}.${dataDateYear}`);

    data.rating = applyRegexByFirstExec(/data-r="([^;]+)/, dataChunk[i]);

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

    if (dataTagsContainer) {

      const dataTagsItem = applyRegexByCleanedExec(/<a href="(.\*?)">(.\*?)<\/a>/g, dataTagsContainer);

      const currentTags = [];

      for (let j = 0; j < dataTagsItem.length; j++) {

        currentTags.push(applyRegexByFirstExec(/>(.\*?)</s, dataTagsItem[j]));

      }

      data.tags = currentTags;

    }

    const dataAuthor = applyRegexByFirstExec(

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

    );

    data.author = dataAuthor ? dataAuthor : `${data.author}`;

    dataStorage.push(data);

  }

Здесь:

1. data – объект, представляющий собой определенный анекдот
2. data.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. data.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, которая заменяет все теги br на \n, а все оставшиеся теги на пробел:

function removeHTML(str) {

  return str.replace(/<br\s\*\/?>/gi, ' \n')

    .replace(/<[^>]\*>?/gm, ' ');

}

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

dataDate - 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

data.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

dataTagsContainer - ищется контейнер (div с классом tags) анекдота с помощью регулярного выражения в определенном чанке. Если его получилось найти (у анекдота указаны теги), то в dataTagsItem записываются все ссылки из dataTagsContainer (с помощью регулярного выражения), а после происходит цикл по каждому элементу dataTagsItem, 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

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)

dataAuthor - ищется автор анекдота с помощью регулярного выражения /<[a-z]+ class=”auth” href=”[^”]\*”>(.\*?)<\/[a-z]+>/s в определенном чанке. Если его нет, то в data.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. dataStorage.push(data) – в конце итерации в dataStorage добавляется созданный объект анекдота

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 на диске:

ensureDirectoryExists(STORAGE\_DIR);

const storageFilePath = PATH.join(STORAGE\_DIR, STORAGE\_FILE\_NAME);

clearFile(storageFilePath);

writeToFile(storageFilePath, dataStorage);

Здесь:

1. 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:

function ensureDirectoryExists(directoryPath) {

  if (!FS.existsSync(directoryPath)) {

    FS.mkdirSync(directoryPath, { recursive: true });

  }

}

1. storageFilePath – определение пути до JSON-файла
2. clearFile – функция, которая проверяет – содержит ли переданный файл какие-то записи. Если содержит, то функция очищает этот файл:

function clearFile(filePath) {

  if (FS.existsSync(filePath)) {

    FS.writeFileSync(filePath, '');

  }

}

1. writeToFile – функция, которая записывает в файл переданную запись:

function writeToFile(filePath, record) {

  FS.writeFileSync(filePath, record);

}

Таким образом, на диске C 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.

Также стоит отметить лог ошибок. В случае, если где-то будет брошена ошибка, выполнится следующее:

main().catch((err) => {

  ensureDirectoryExists(LOGGER\_DIR);

  const errorLogFilePath = PATH.join(

    LOGGER\_DIR,

    `${getCurDate()}.${getCurMonth()}.${getCurYear()} ${getCurHours()}-${getCurMinutes()}-${getCurSeconds()}.txt`

  );

  logErrorToFile(err, errorLogFilePath);

})

Здесь:

1. errorLogFilePath – создается путь до файла ошибок, где файл ошибок имеет заданный формат наименования (текущее время) – dd.mm.yy. hh:mm:ss.txt
2. 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:

function ensureDirectoryExists(directoryPath) {

  if (!FS.existsSync(directoryPath)) {

    FS.mkdirSync(directoryPath, { recursive: true });

  }

}

1. logErrorToFile – функция, которая записывает в переданный файл сообщение об ошибки в заданном формате:

<Дата> <Время> - <Тип ошибки>:

<CustomError>: <Тут наше описание ошибки>

<SystemError>: <Тут системный текст ошибки>

function logErrorToFile(error, filePath) {

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

    CustomError: ${error.message}

    SystemError: ${location}

  `;

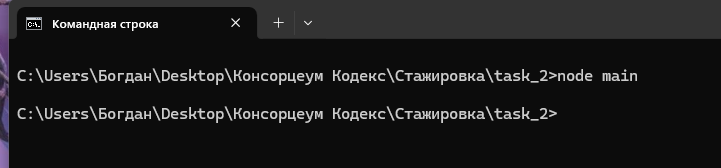
  FS.appendFileSync(filePath, errMessage + '\n', 'utf8');

}

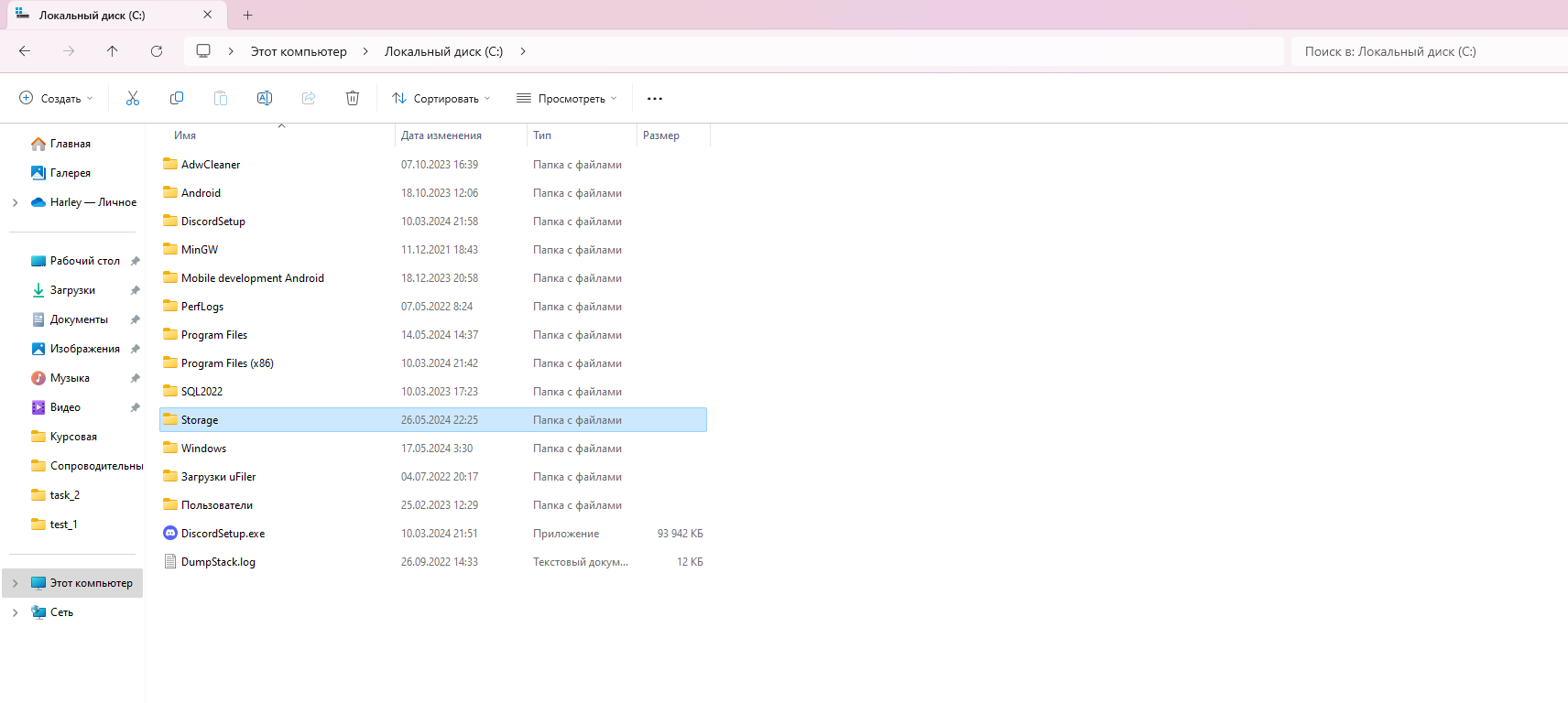
**Тесты:**

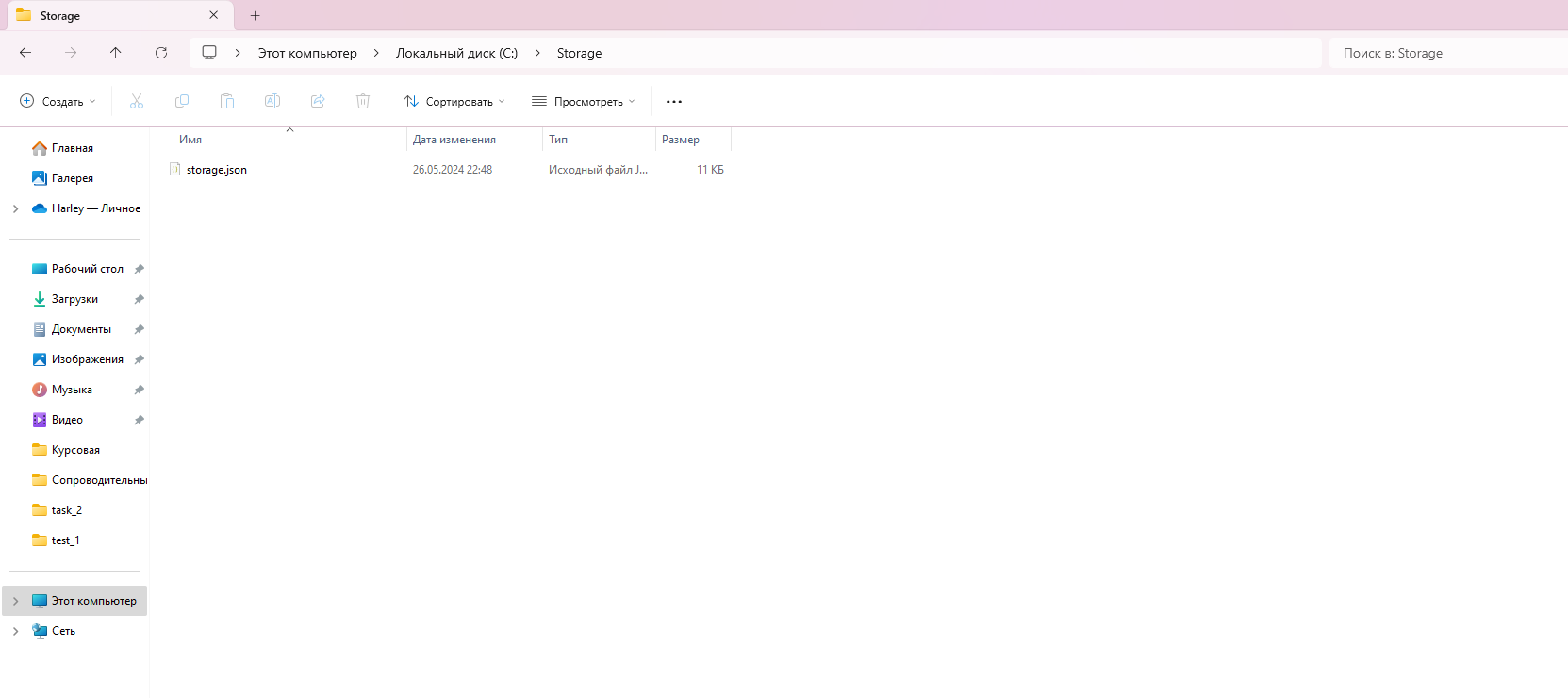
1. Без ошибок:

-Запустим программу:

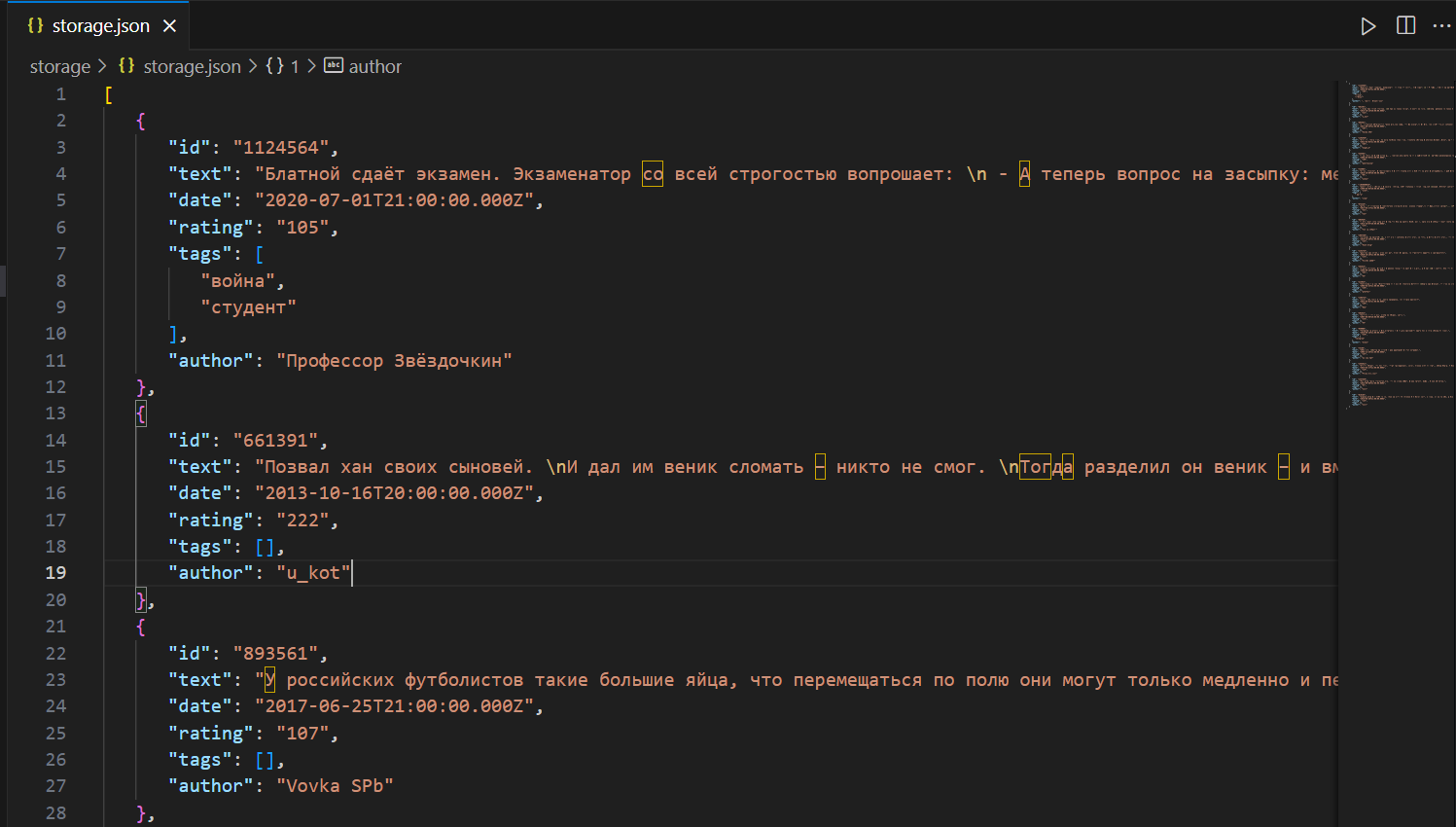


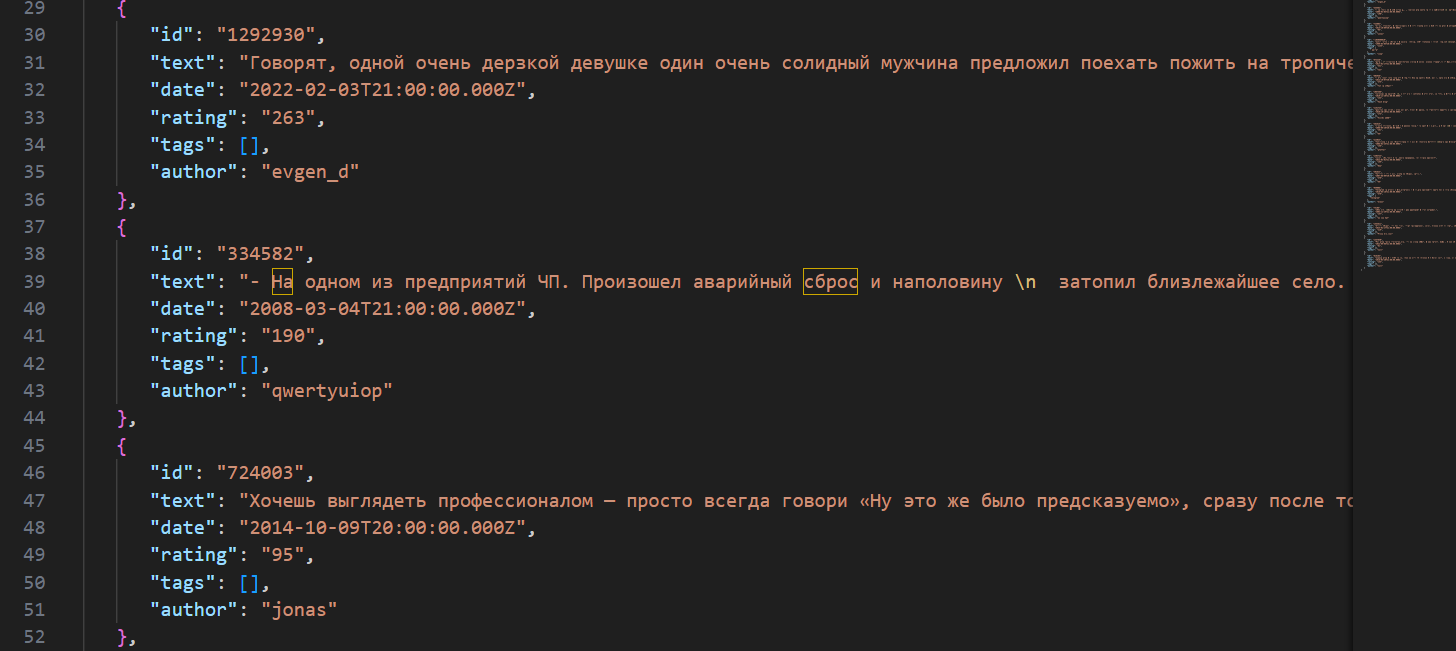
Как можно заметить – программа выполнилась. Это значит, что на диске C должна появиться директория Storage с файлом storage.json:

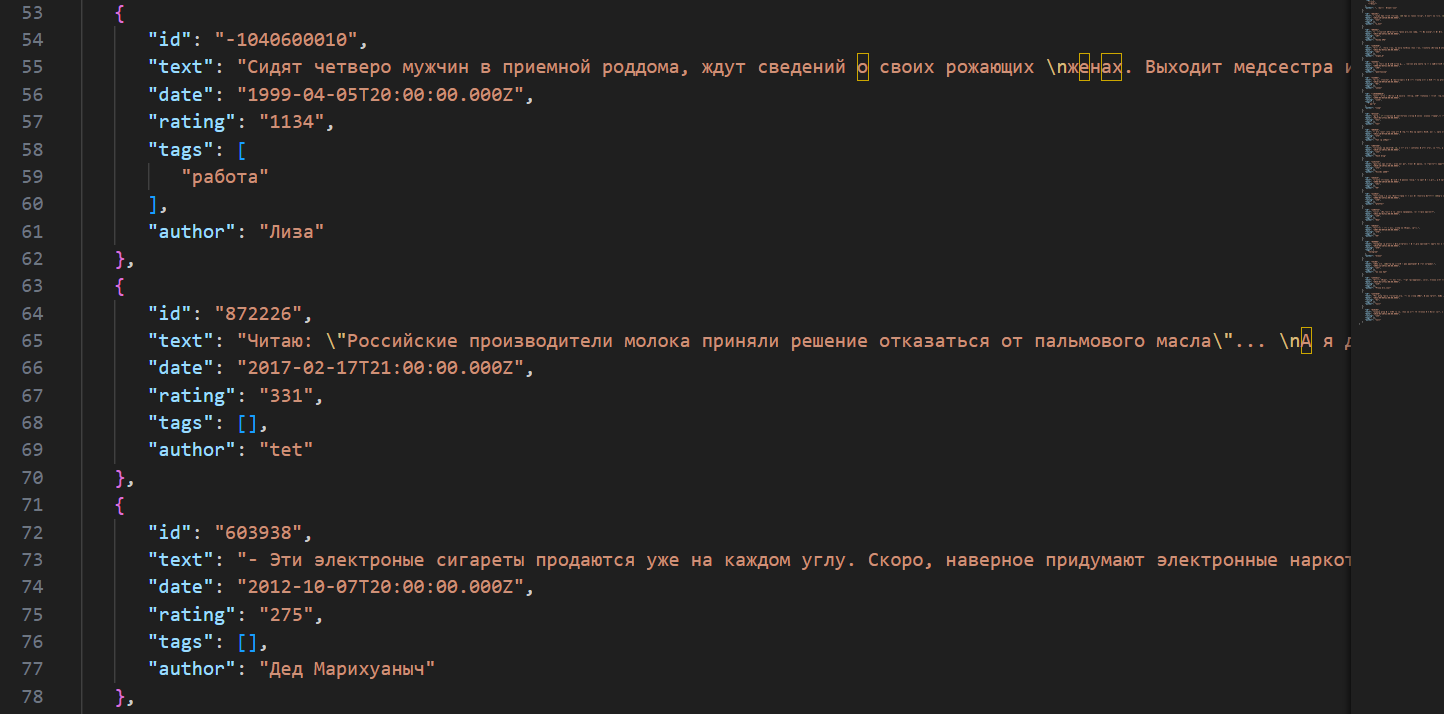


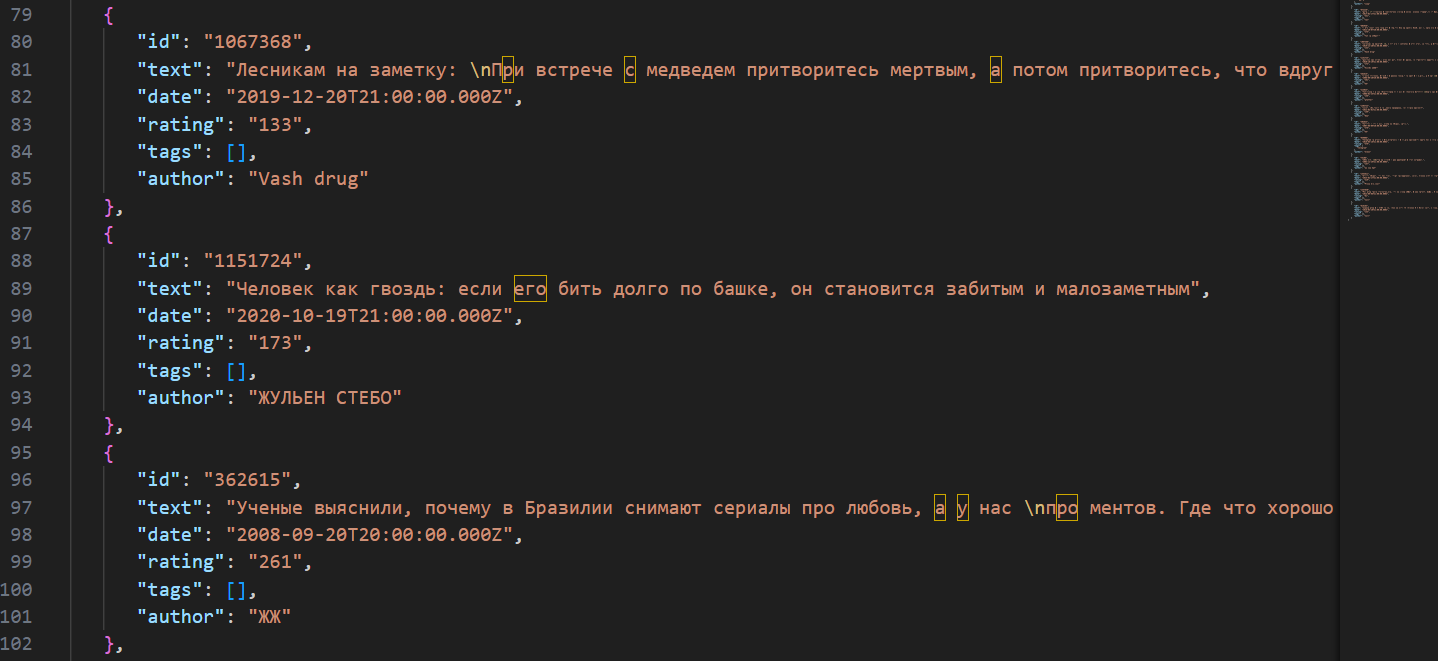


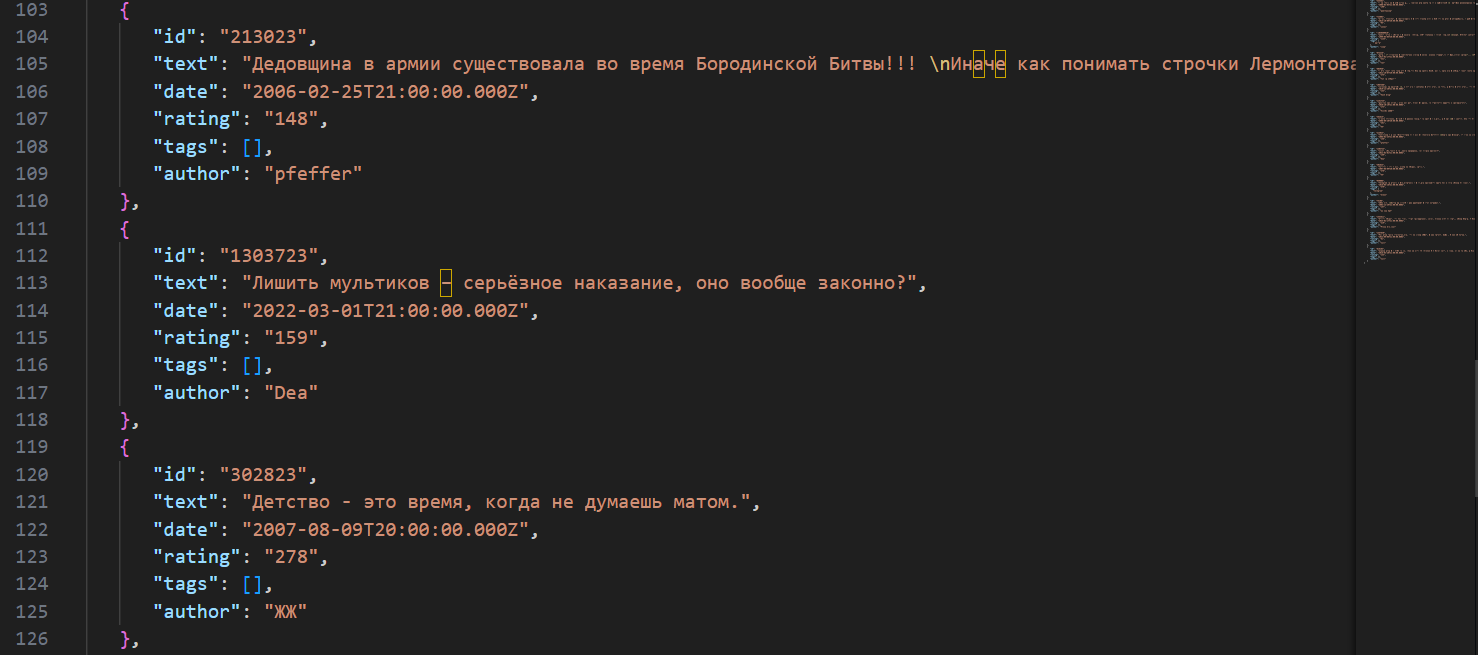
Как можно заметить – все есть. Теперь осталось проверить содержимое файла storage.json:

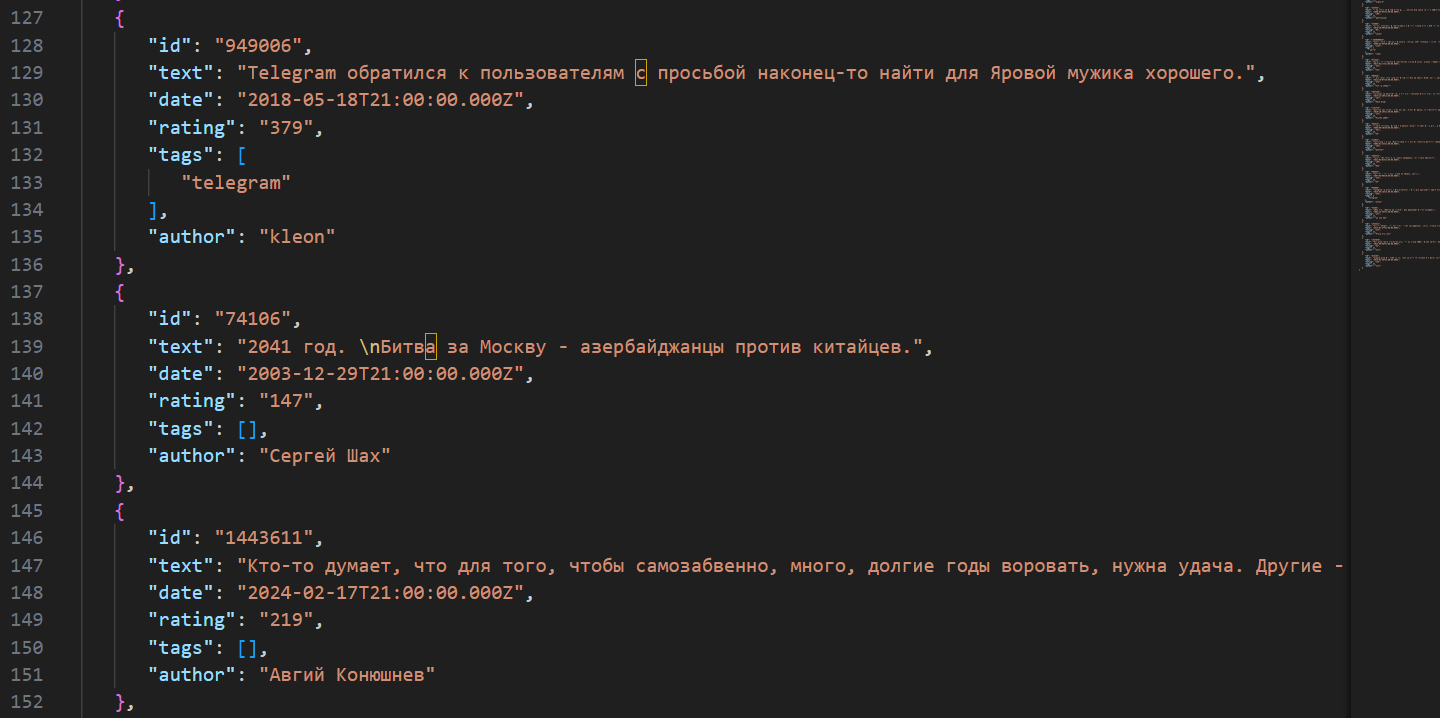


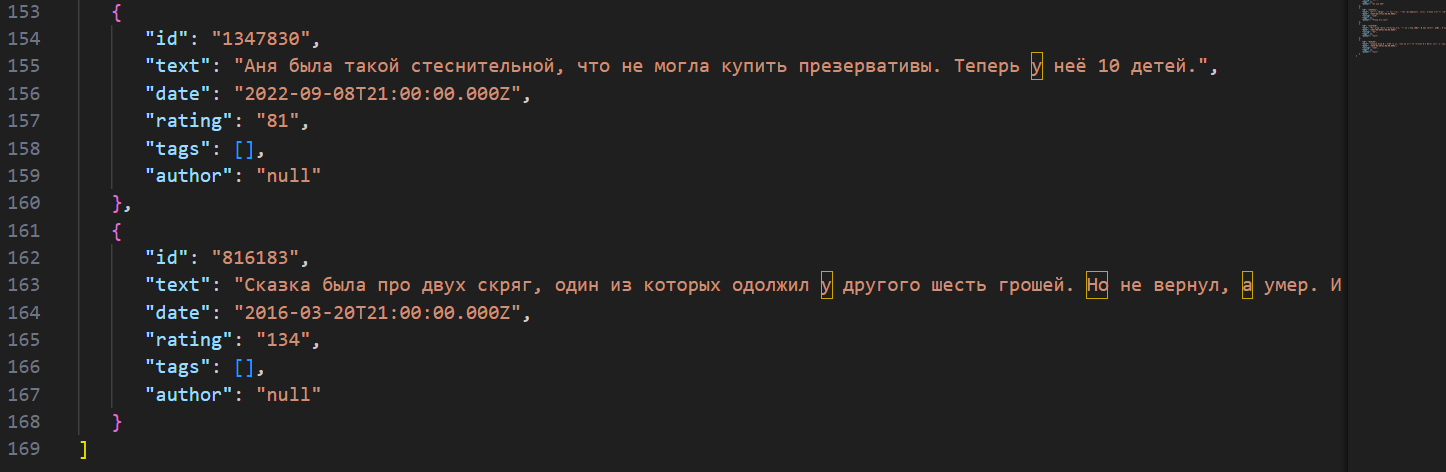






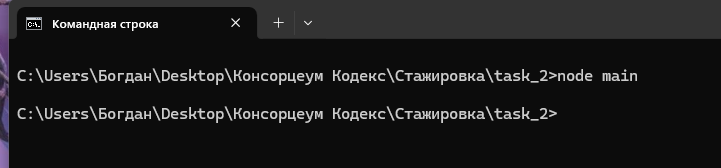


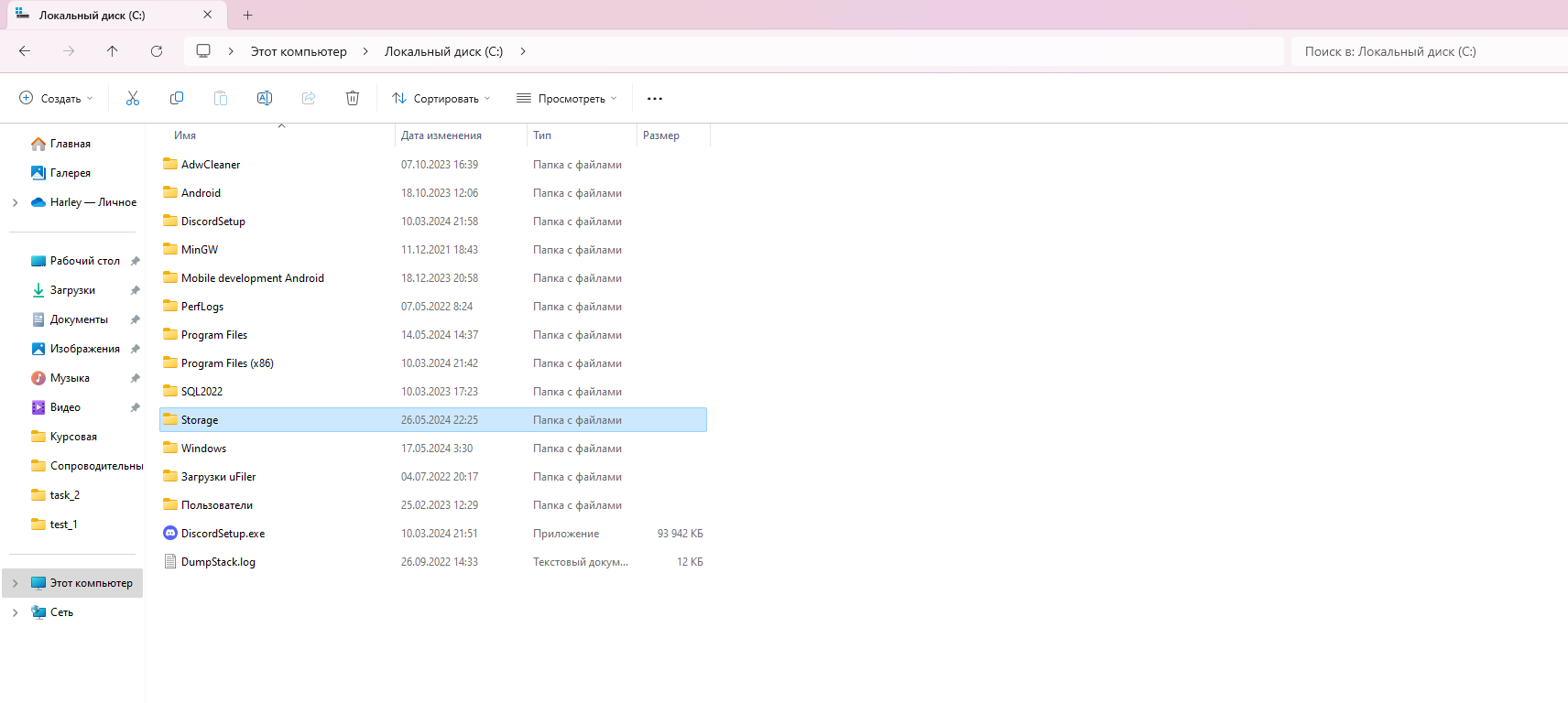


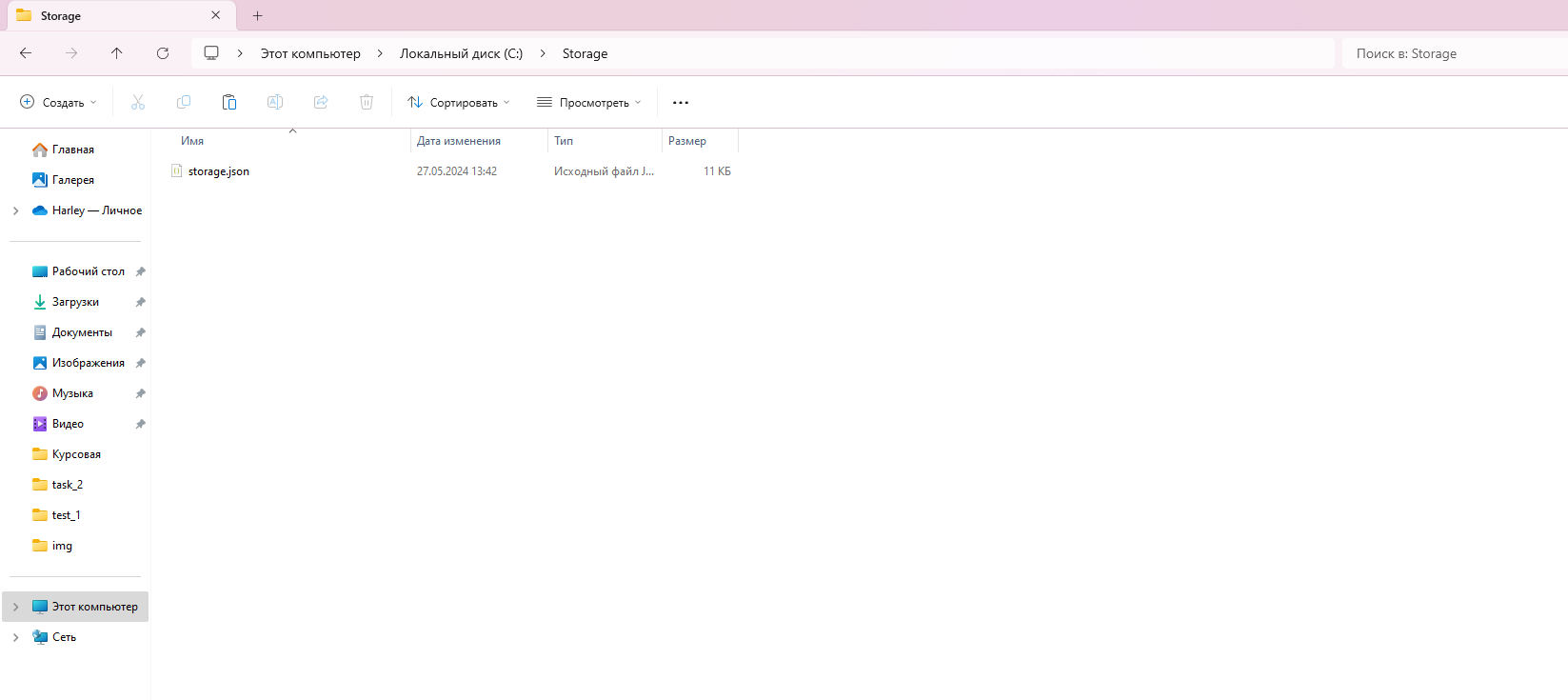


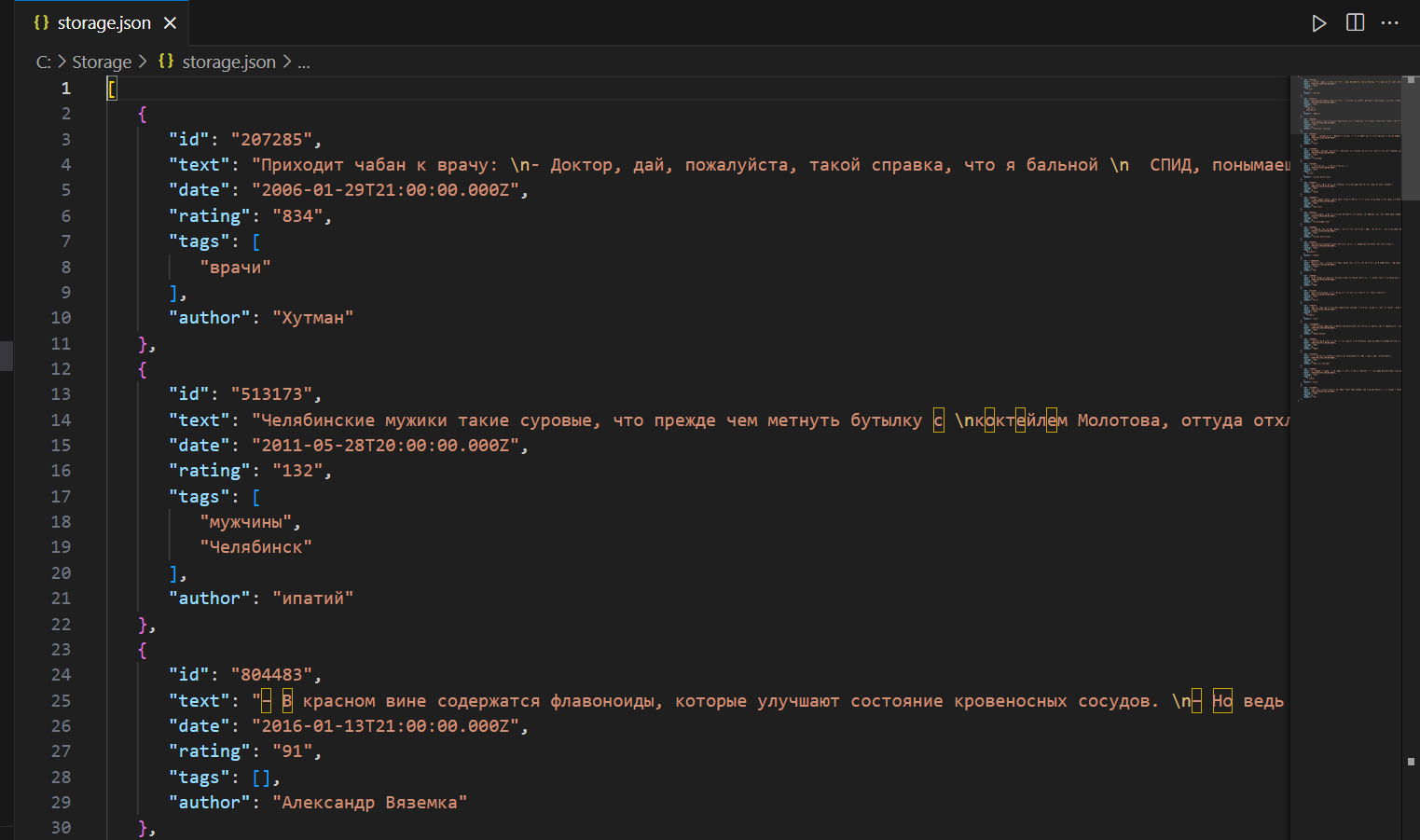
Как можно заметить – весь формат верный и число анекдотов полное – 20.

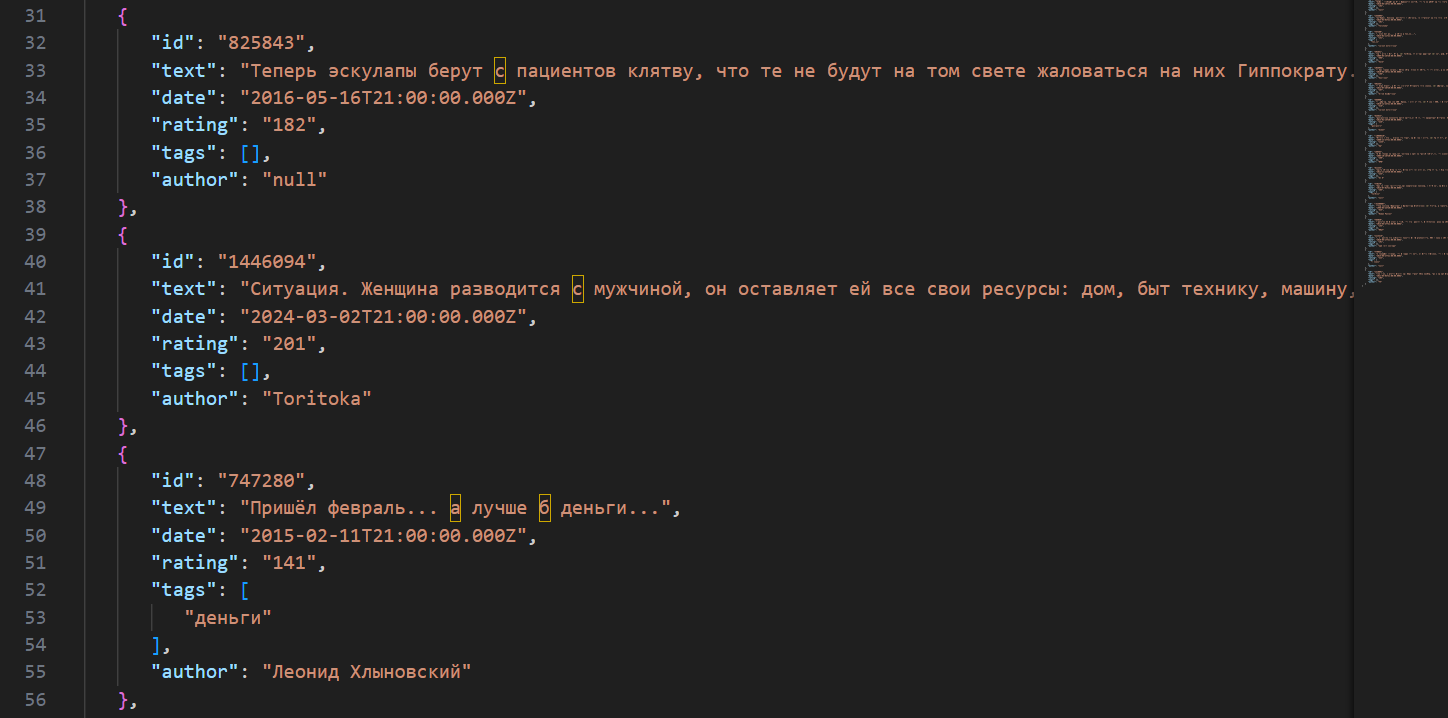
-Теперь запустим программу еще раз, чтобы убедиться, что не будет создано новой директории Storage или нового файла storage.json. Все должно отразиться в той же директории и в том же файле, также storage.json – должен обновить свое содержимое:

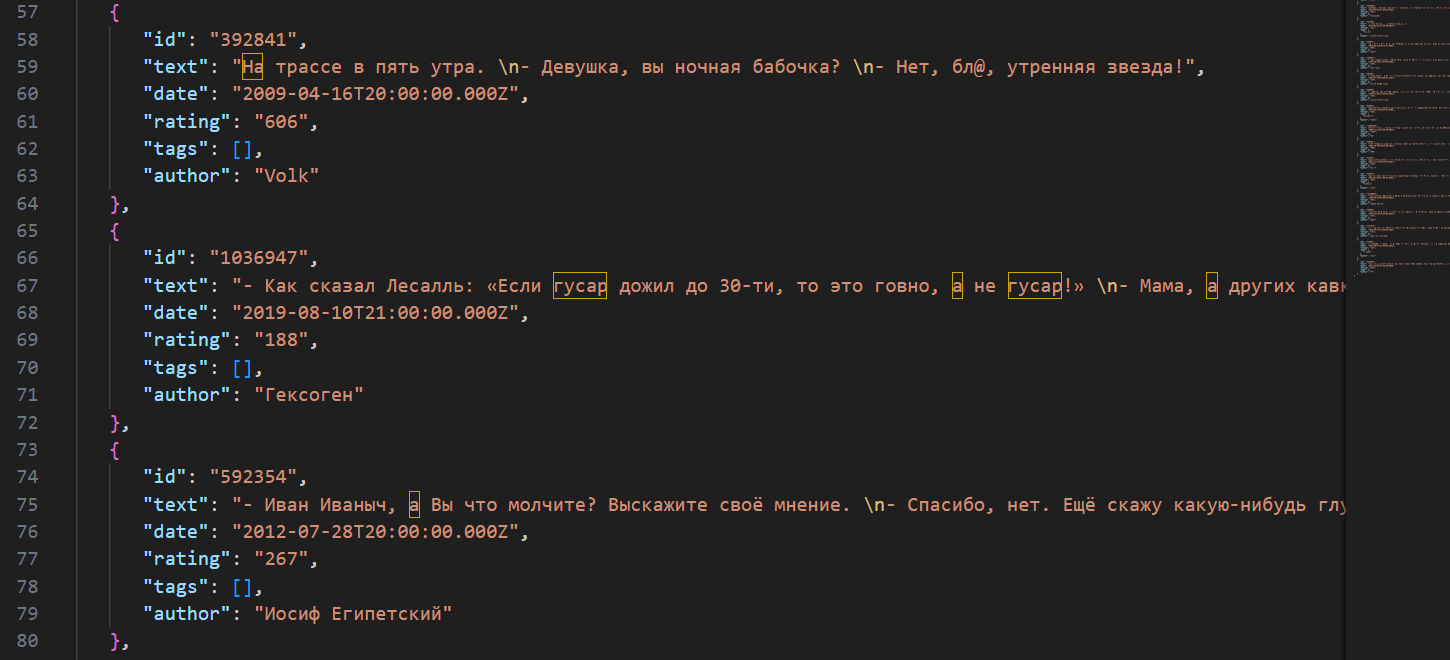


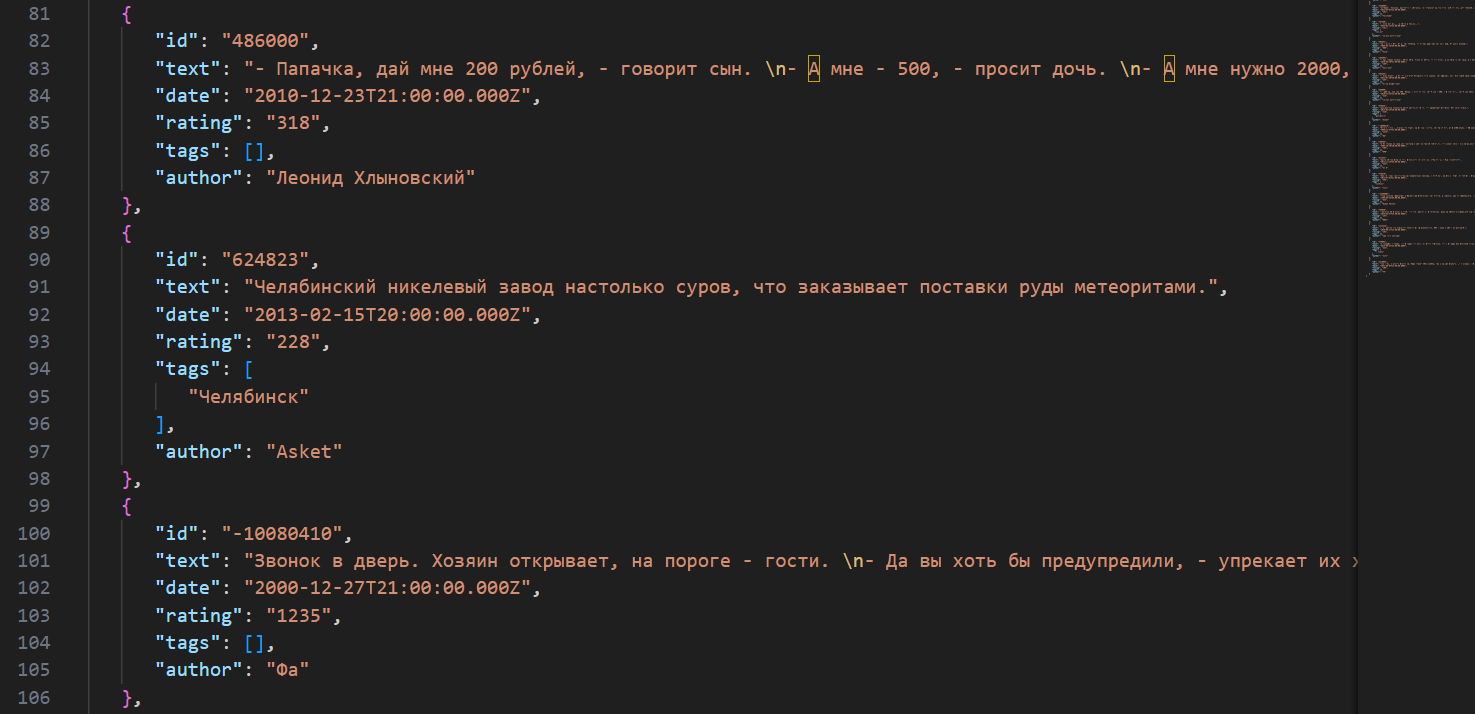


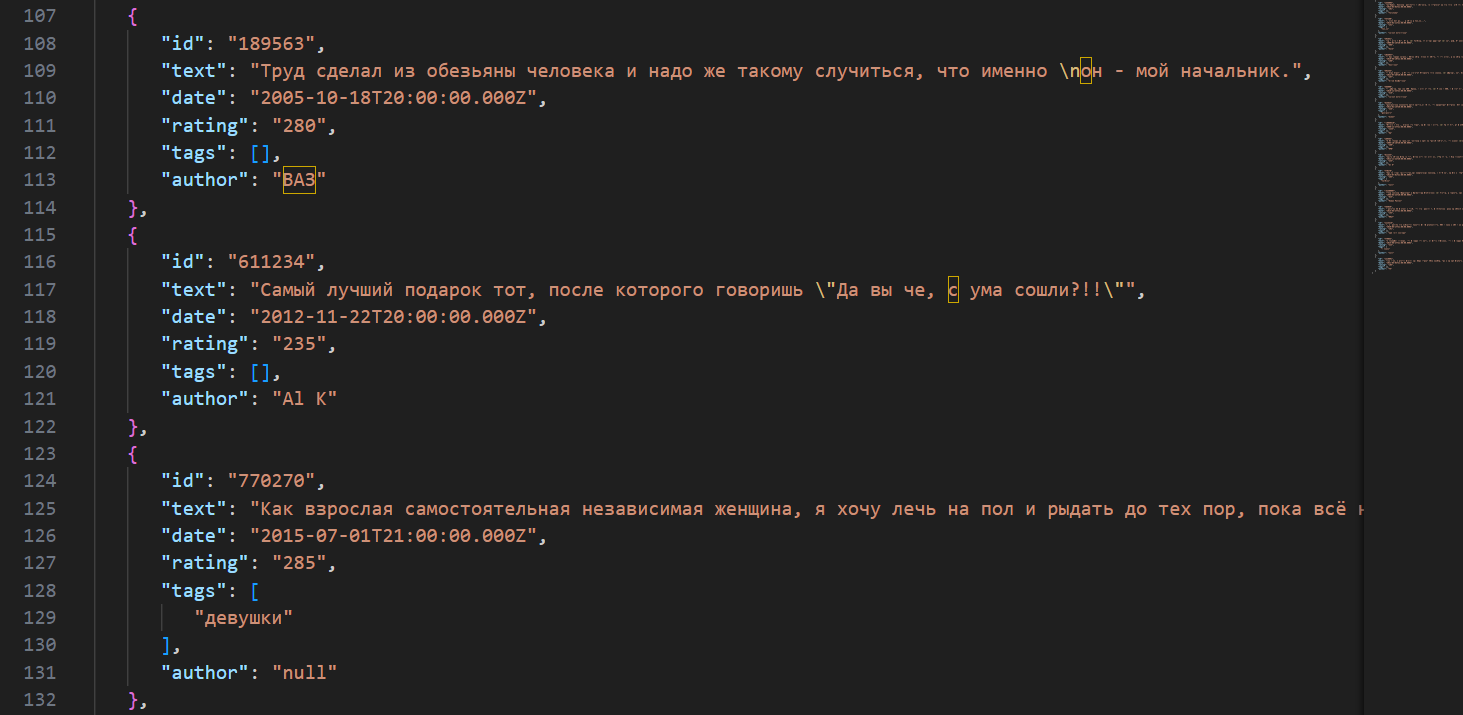


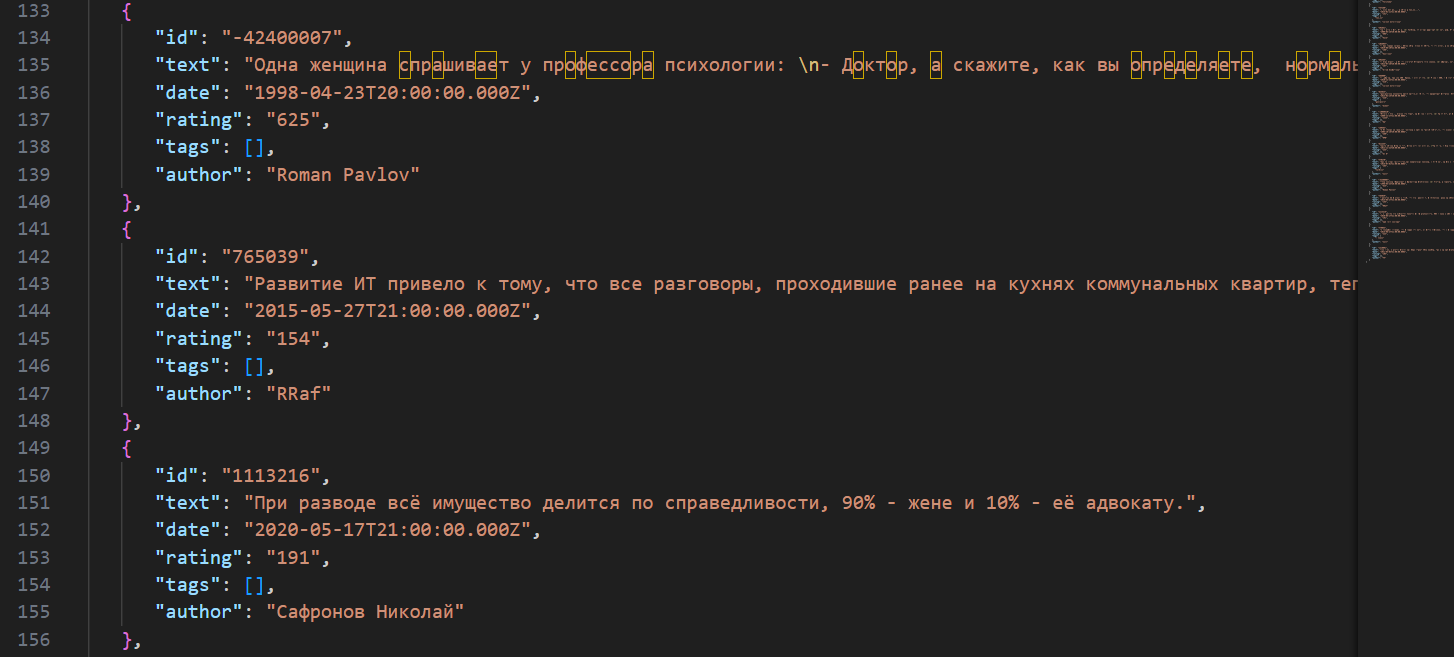


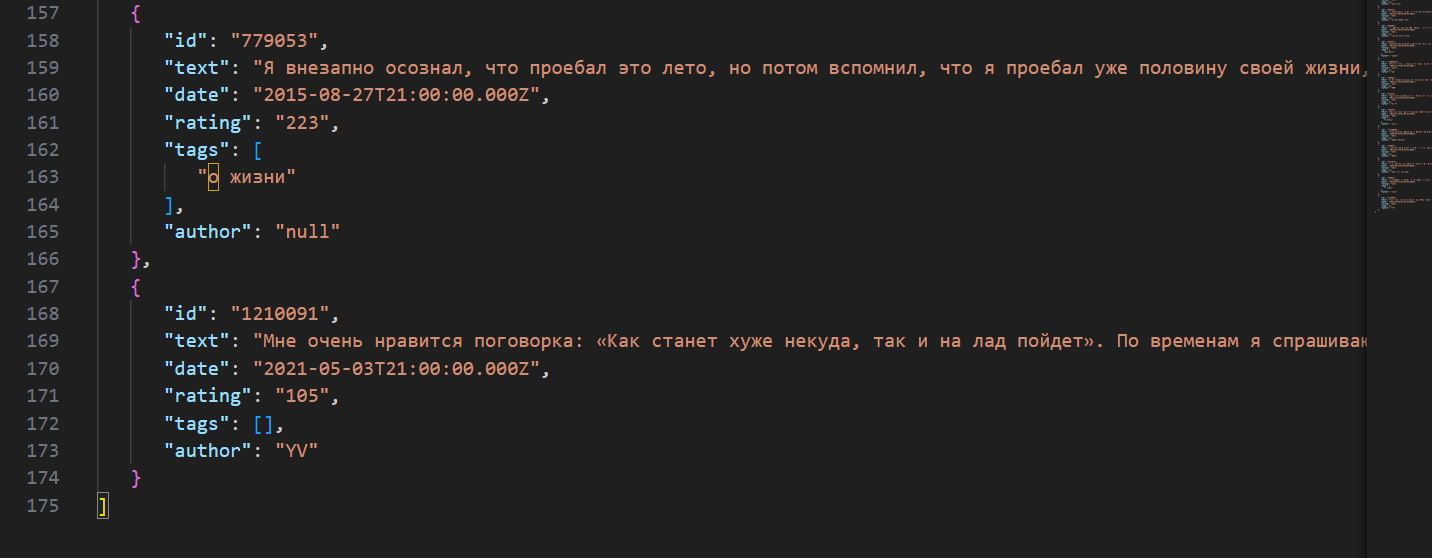








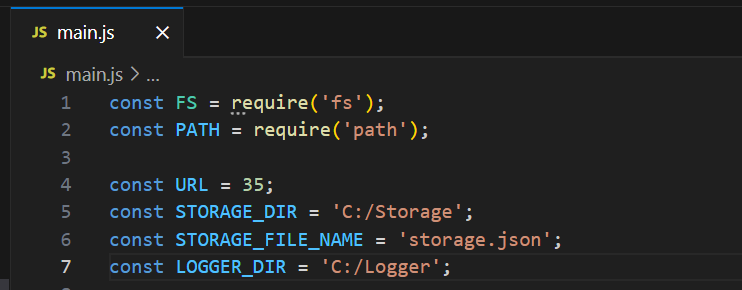


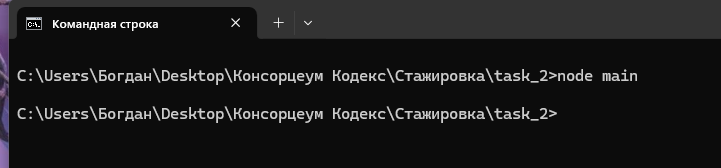


Как можно заметить – все работает правильно

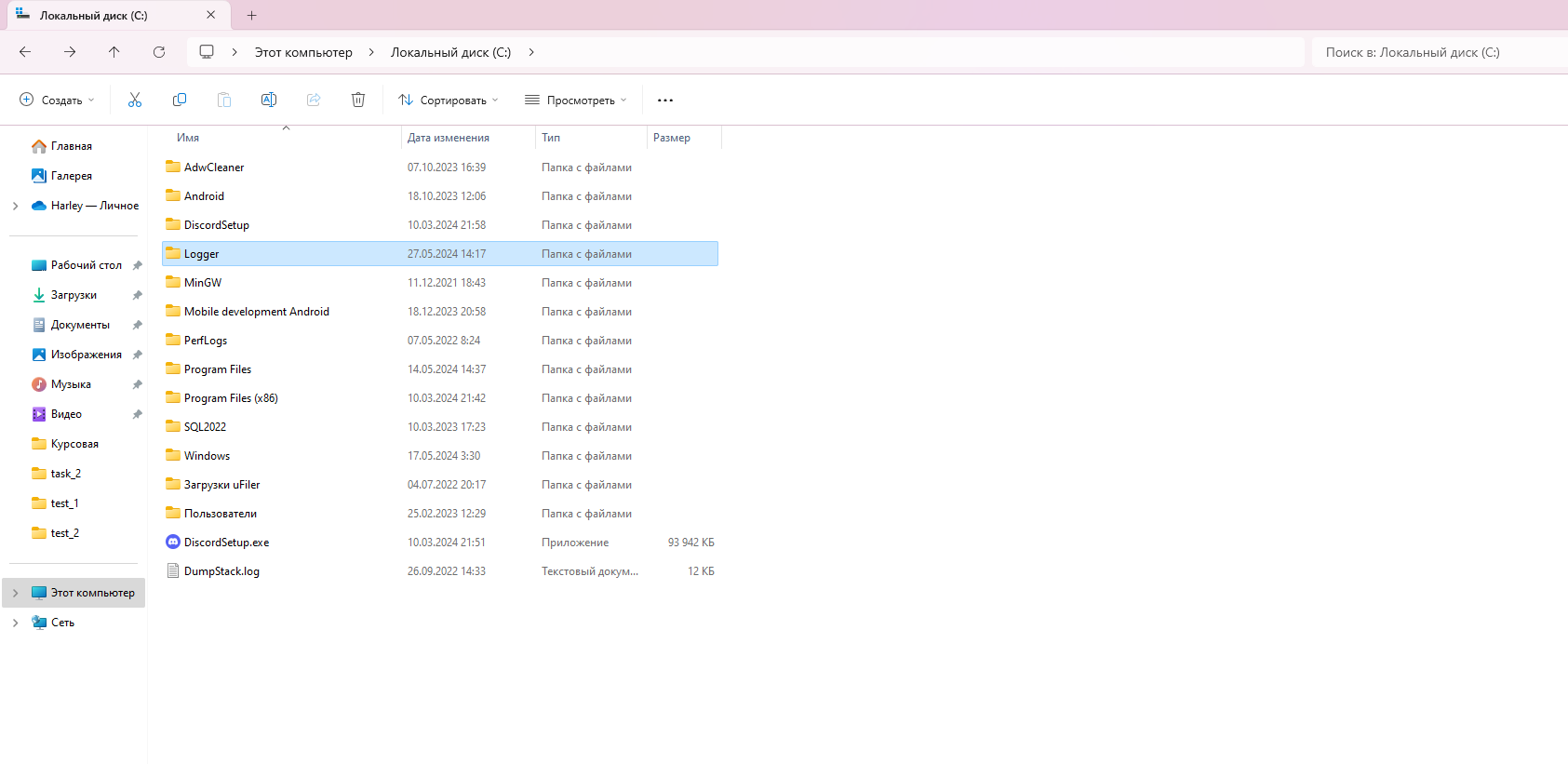
1. С ошибками (перед этим удалим создавшуюся директорию Storage на диске C):

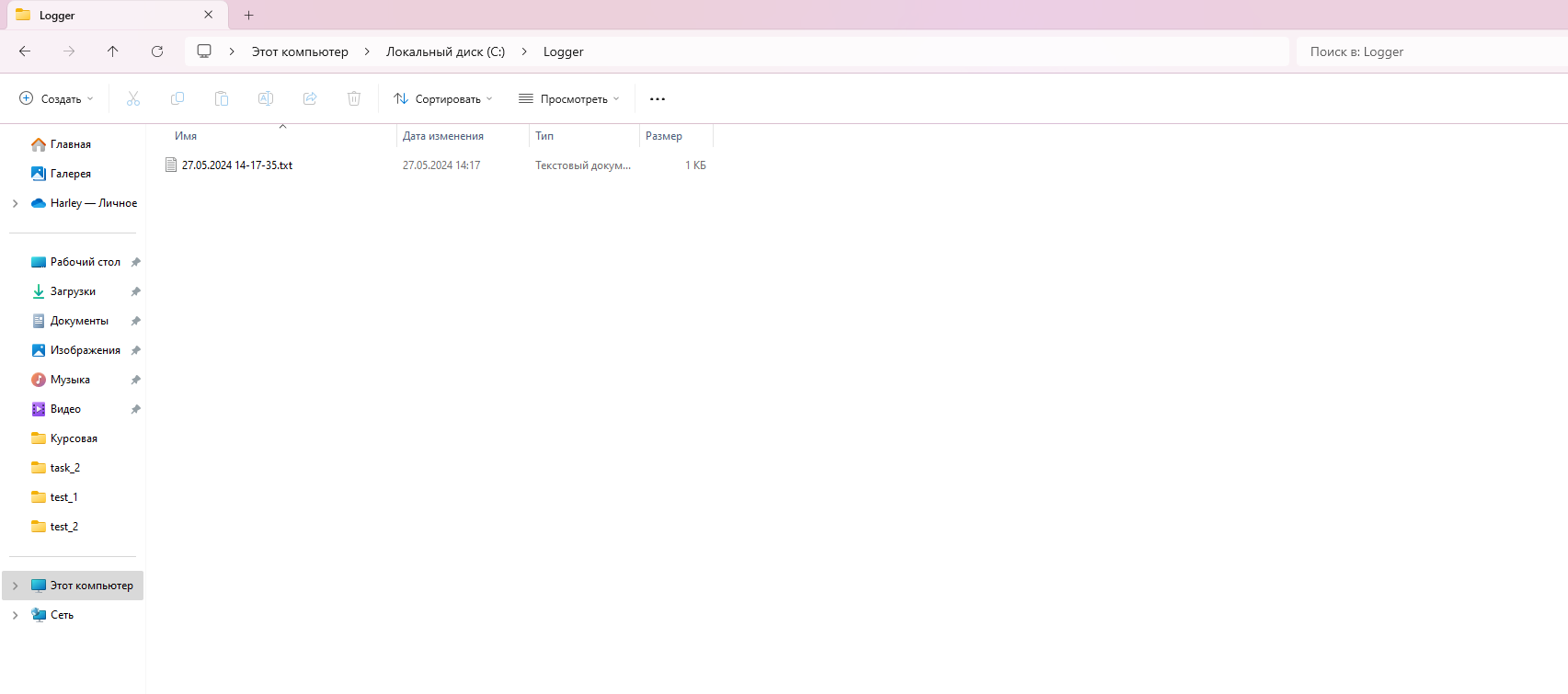
-Добавим ошибку в url – вместо строки, отправим число:



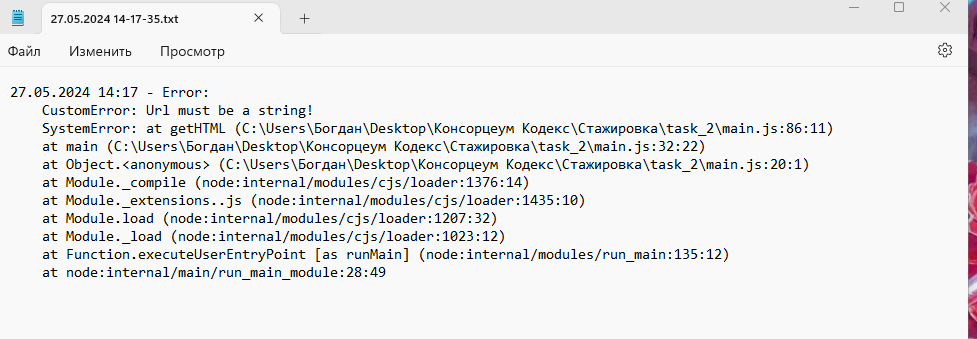


Программа выполнилась. Но она должна была выполниться с ошибкой, это значит, что на диске C должна появиться директория Logger с файлом dd.mm.yy. hh:mm:ss.txt и при этом, на диске C не должно быть директории Storage:



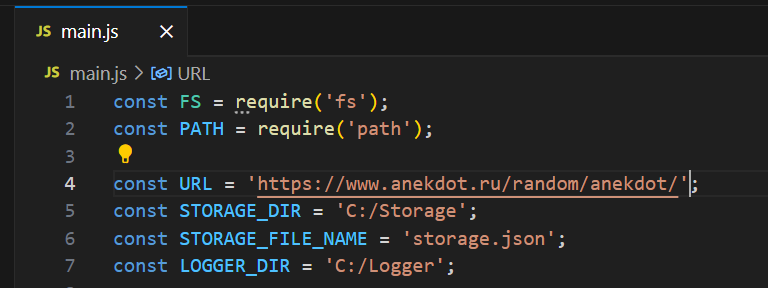


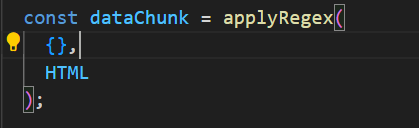
В Logger появился соответствующий файл, и на диске C отсутствует Storage. Значит, все верно. Осталось проверить содержимое файла dd.mm.yy. hh:mm:ss.txt:

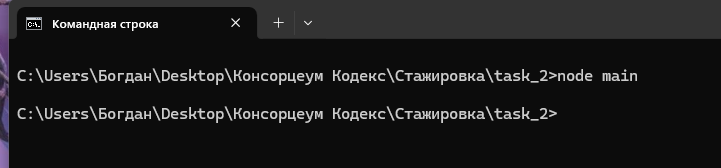


Данный файл содержит корректное содержимое.

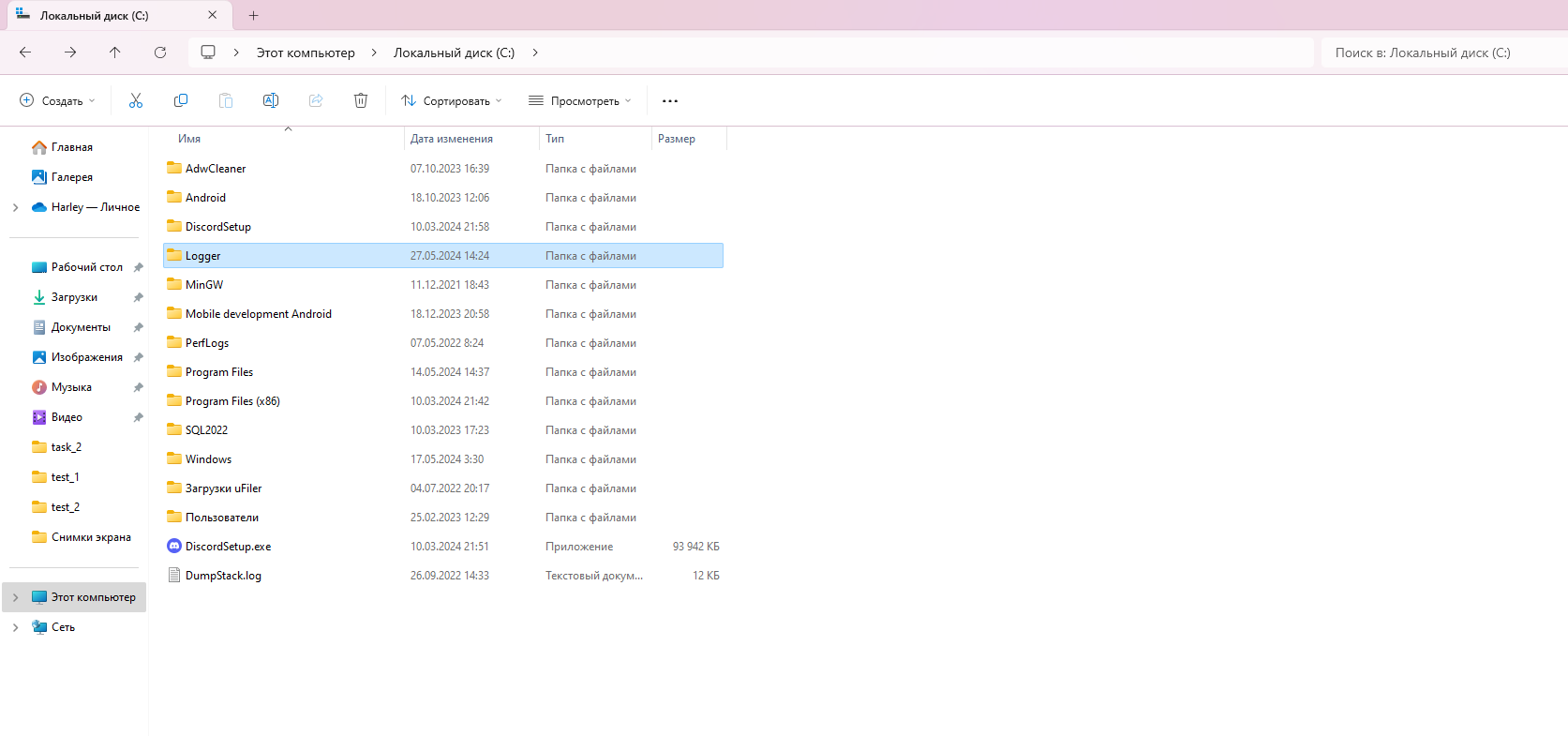
-Теперь вернем назад url и сделаем ошибку, передав вместо регулярного выражения – обычный объект, чтобы убедиться, что при следующей ошибке – в директории Logger создастся новый файл с ошибкой:

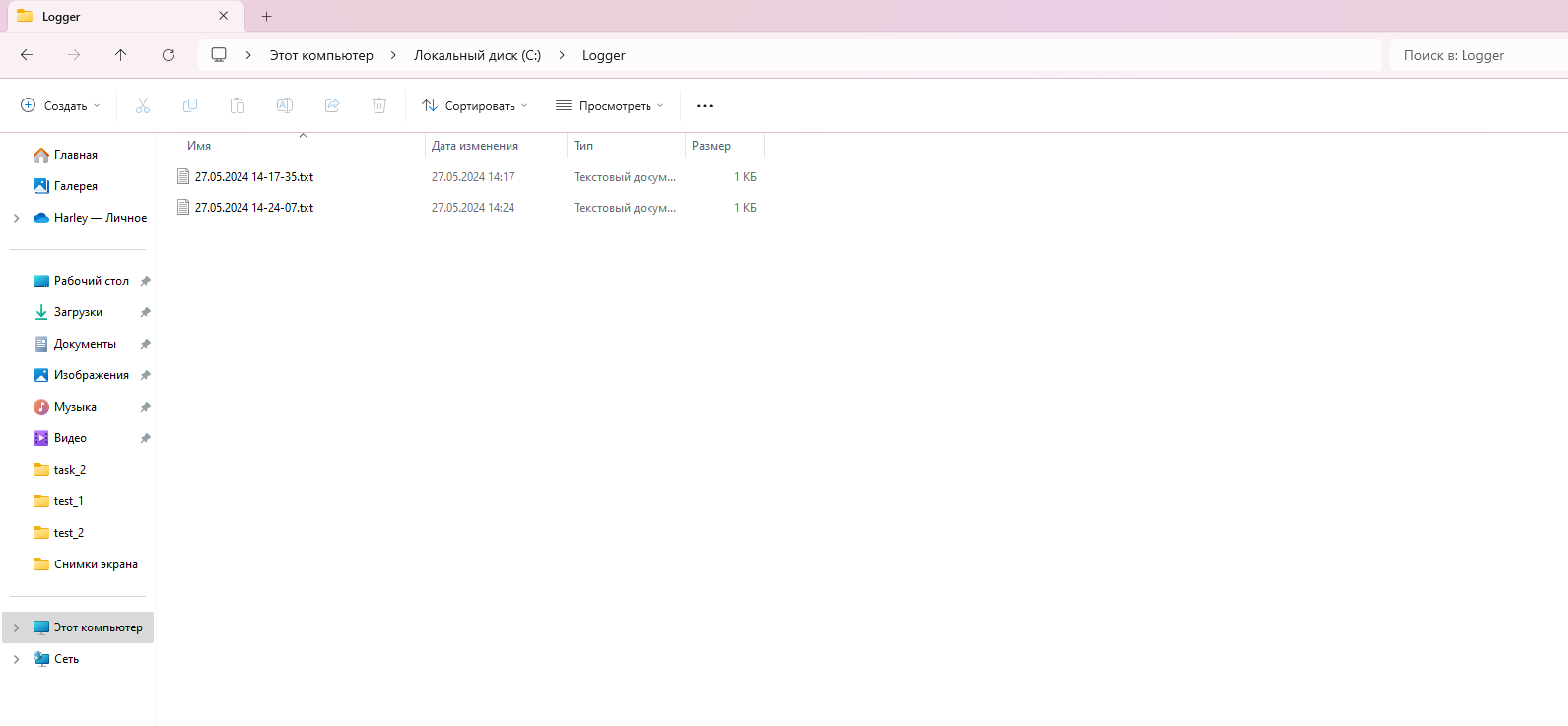




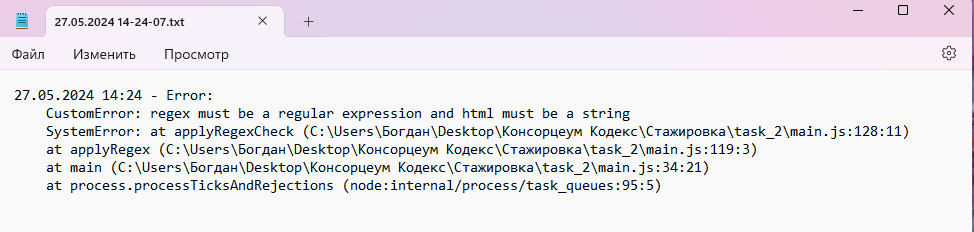


Программа выполнилась, снова проверяем диск C и Logger:

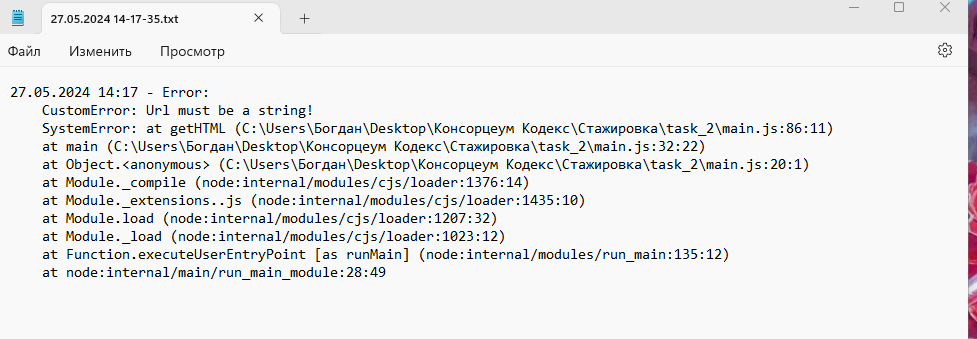




Директория Storage отсутствует, а в Logger создался новый файл. Значит, все верно. Осталось проверить содержимое нового файла:



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.