Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

БІБЛІОТЕКА AESON

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення» 121

Керівник курсової роботи
доц. Проценко В. С.
_____
(підпис)
“____” _____ 2020 р.

Виконав студент Шудра І. О.
“____” _____ 2020 р.

Київ 2020

Зміст

Анотація

У роботі представлено аналіз бібліотеки Aeson написаної на мові програмування Haskell. Також описано певні особливості JSON та використаних інструментів.

В якості практичної частини створено програму, що з використанням Aeson завантажує дані JSON з веб-ресурсу Reddit та обробляє їх.

У першому розділі наведено опис бібліотеки Aeson та інших використаних інструментів. У другому розділі описано процес побудови застосунку для отримання та обробки даних з сайту Reddit.

Вступ

Питання збереження та передачі даних є актуальним для розробників будь-яких застосунків – як для веб-сайтів, так і для будь-яких інших. Формат збереження даних тут відіграє дуже важливу роль, оскільки він повинен бути “легким” (займати якнайменше місця на диску), простим у використанні та обробці, а також, бажано, читабельним для людини. Одним із найпопулярніших форматів збереження даних є JSON. На сьогоднішній день він використовується майже всюди, від вищезгаданих веб-застосунків і до комп’ютерних ігор. Бібліотеки функцій для обробки JSON-даних написані та використовуються у багатьох мовах програмування, і Haskell – не виняток.

Метою даної курсової роботи є дослідження бібліотеки Aeson, її можливостей та методів взаємодії з даними, та створення програми для відображення роботи даної бібліотеки.

Методом дослідження є вивчення матеріалів про Haskell та Aeson та написання програми з використанням функціоналу даної бібліотеки, що матиме також практичну цінність.

Джерелами дослідження є різноманітні статті та книжки, що описують бібліотеку Aeson, Haskell та інший функціонал, використаний при проектуванні та створенні програми.

Робота складається з двох основних розділів.

У першому розділі розглядаються інструменти розробки – мова програмування Haskell, бібліотеки Aeson, Wreq, опис JSON та формат роботи із ним. У цьому розділі також міститься інформація про доступ до JSON-даних веб-ресурсу Reddit, що було використано у практичній частині роботи.

Другий розділ присвячено проектуванню та розробці програмного продукту, що реалізує завантаження даних з мережі інтернет та їх обробку.

Постановка задачі:

1. Розглянути та проаналізувати формат JSON, його структуру та загальні методи роботи з ним.
2. Проаналізувати доступні інструменти розробки — Haskell, Aeson. Проаналізувати структуру даних, отримуваних з веб-сайту Reddit.
3. Спроектувати та написати програму, що використовуватиме бібліотеку Aeson та реалізовуватиме обробку JSON-даних з її допомогою.

1. JSON. ОБРОБКА ФОРМАТУ JSON У МОВІ ПРОГРАМУВАННЯ HASKELL. ВИКОРИСТАННЯ БІБЛІОТЕКИ AESON

1.1 Формат збереження даних JSON та його структура

Перед тим, як використовувати бібліотеку для обробки JSON, потрібно зрозуміти що саме він собою являє. Згідно з інформацією на офіційному сайті [1], JSON розшифровується як JavaScript Object Notation, і є відносно легким (за обсягом місця, що він займає у пам’яті комп’ютера) форматом обміну даними. Він є простим для читання та написання людиною, а також його легко аналізувати та генерувати. Це – текстовий формат, що є повністю незалежним від мови програмування, і при цьому використовує правила, що є дотичними до С-подібних мов програмування – таких як C, C++, Java тощо. З огляду на всі ці властивості, JSON є дійсно дуже зручною формою обміну даними, чим і зумовлена його популярність серед розробників.

Загальна структура JSON побудована на двох основних типах: пари ключ/значення та списки. Перший тип існує та використовується у багатьох мовах програмування як об’єкт, структура, словник тощо, а другий зазвичай представлено у вигляді масиву, зв’язного списку або послідовності.

При використанні цих структур також використовуються менші типи – рядки, числа, булеві значення та значення null:

1) Рядки – структура, що майже повністю повторює відповідну з мови C або Java. Містить в собі певний текст, також може містити escape-послідовності. Записується у подвійних лапках.
2) Числа. Числа у JSON можуть бути як цілими, так і з плаваючою крапкою. Через це, при відсутності даних про те, чи тільки ціле число може знаходитись у даній комірці, варто обробляти такі комірки як числа з плаваючою крапкою. Інакше це може призвести до втрати точності числа, або ж до помилкової ситуації.

3) Булеві значення – значення true або false.

4) null – пусте значення.

Всередині пар ключ/значення та списків можуть знаходитись також інші об’єкти типу ключ-значення, або інші списки. З іншою інформацією стосовно даного формату можна ознайомитись на офіційному ресурсі [1].

У мові програмування Haskell типам JSON також можна визначити відповідники. Через це можна інтерпретувати об’єкти, записані у даній формі, через структури притаманні мові Haskell. Тобто, можна виконувати обробку JSON-файлів та отримувати типи, що будуть інтуїтивно дуже близько до заданих у файлі, що безперечно є плюсом до використання JSON у комбінації з даною мовою програмування.

1.2 Бібліотека Aeson

Як було зазначено вище, Haskell має досить сприятливі умови для обробки JSON-даних. Aeson — це одна з бібліотек для обробки JSON, що виконує швидку обробку та створення (серіалізацію та десеріалізацію) JSON-об’єктів, які після цього легко можна використовувати будь-якими іншими засобами мови.

Найголовніша складність у роботі з JSON із використанням Haskell в тому, що JSON майже не бере до уваги типи даних які зберігаються всередині, і більшість даних зберігається у простих рядках. Ця концепція погано вкладається у жорстку типізацію, з якою ми працюємо. Aeson у свою чергу дозволяє сумістити систему типів Haskell та гнучкі JSON-дані. При цьому ми отримуємо можливість обробляти дані не втрачаючи переваг, які надає жорстка типізація мови програмування Haskell.

Aeson покладається на дві основних функції, щоб транслювати дані між типами Haskell та JSON-даними: encode та decode.

Щоб використати ці дві функції, потрібно щоб тип даних, який використовується, був екземпляром класів ToJSON (для функцій encode) та FromJSON (для функцій decode відповідно).

```
decode :: FromJSON a ⇒ ByteString → Maybe a
```

Рис. 1.1 Сигнатура функції decode

Функція decode приймає JSON-дані та трансформує їх у потрібний тип. Сигнатура функції вказана на малюнку 1. Є дві речі, які варто зауважити. По-перше, ця функція повертає тип Maybe — один з типів, за допомогою якого зручно виявляти факт помилки в Haskell. Може бути багато причин, через які обробка даних піде не так, як хотілося б — наприклад, JSON може містити насправді не той тип, яким ми хочемо його представити. У такому випадку функція поверне Nothing. Інакше — дані типу Just a, де a — саме потрібний нам тип.

Haskell має інший тип, який використовується для більш детального опису помилок — Either. Aeson також пропонує функцію eitherDecode, яка повертає більш інформативні повідомлення про помилку через конструктор Left.

```
eitherDecode :: FromJSON a ⇒ ByteString → Either String a
```

Рис. 1.2 Сигнатура функції eitherDecode

Другий важливий нюанс полягає якраз в тому, що тип, який ми передаємо функціям decode або eitherDecode, повинен бути екземпляром класу FromJSON — як видно із сигнатури цих двох функцій.

Інша важлива функція, яку надає Aeson — це функція encode, що працює протилежно до decode.

```
encode :: ToJSON a ⇒ a → ByteString
```

Рис. 1.3 Сигнатура функції encode

Як ми бачимо із сигнатури на малюнку 3, encode приймає тип, що є екземпляром класу ToJSON, та перетворює його на JSON-об’єкт, представлений як ByteString. ToJSON це двійник FromJSON. Якщо тип є об’єктом обох цих класів, його запросто можна конвертувати в або з JSON.

Для багатьох класів, створення екземплярів ToJSON та FromJSON є дуже простим. Для прикладу створимо тип книги, який буде мати декілька простих полей — назва, автор та рік видання.

```
data Book = Book
          { title :: T.Text
          , author :: T.Text
          , year :: Int
          } deriving Show
```

Рис. 1.4 Структура типу Book

Існує найпростіший спосіб зробити тип Book екземпляром обох потрібних нам класів. Для цього достатньо використати розширення мови, що називається DeriveGeneric. Це розширення додає можливість писати стандартні визначення для екземплярів класів, що дозволяє створювати екземпляри без додаткових стрічок коду. Все, що потрібно зробити — це додати до нашого класу Book слово Generic:

```
data Book = Book
          { title :: T.Text
          , author :: T.Text
          , year :: Int
          } deriving (Show, Generic)
```

Рис. 1.5 Додаємо до класу Generic

Після цього потрібно додати всього дві стрічки коду, щоб забезпечити можливість використання цього типу у обробці JSON-даних:

```
instance FromJSON Book
instance ToJSON Book
```

Рис. 1.6 Найпростіший спосіб створення екземплярів ToJSON та FromJSON

В результаті отримуємо можливість легко використовувати цей клас із JSON-даними:

```
myBook :: Book
myBook = {title="LOTR",author="J.R. Tolkien", year=1948}

myBookJSON :: BC.ByteString
myBookJSON = encode myBook
-- out: "{\"title\":\"LOTR\",\"author\":\"J.R. Tolkien\",\"year\":1948}"

bookFromJSON :: Maybe Book
bookFromJSON = decode myBookJSON
-- out: Just (Book {title="LOTR",author="J.R. Tolkien", year=1948})
```

Рис. 1.7 Використання класу Book з encode та decode

Саме це вміє робити Aeson — зі стрічки, що містить JSON, ми змогли створити тип даних всередині Haskell. В той же час, у багатьох інших мовах програмування збереження JSON даних вимагає створення таблиці ключів-значень або словника. Завдяки Aeson, у Haskell можна отримувати щось набагато більш потужне з JSON-даних.

На прикладі “поганих” даних можна також продемонструвати роботу eitherDecode. Створимо стрічку з JSON-даними, що матиме замість ключа author ключ writer. У такому випадку, виклик та результат роботи цієї функції будуть виглядати так:

```
badJSON :: BC.ByteString
badJSON = "{\"writer\":\"LOTR\",\"author\":\"J.R. Tolkien\",\"year\":1948}"

bookFromBadJSON :: Maybe Book
bookFromBadJSON = eitherDecode badJSON
-- out: Left "Error in $: The key \"author\" was not found"
```

Рис. 1.8 Отримання тексту помилок через eitherDecode

Завдяки такому повідомленню можна дізнатись конкретну причину, з якої обробка JSON пройшла невдало.

Не дивлячись на те, що DeriveGeneric набагато спрощує використання Aeson, бувають випадки, коли потрібно вручну вказувати реалізацію потрібних функцій. Наприклад, маємо помилку, яку ми хочемо обробити. Тип помилки всередині JSON має параметри message та error.

```
sampleError :: BC.ByteString
sampleError = "{\"message\":\"Wrong!\",\"error\": 1}"
```

Рис. 1.9 JSON класу помилки

Як і раніше, для використання нашої бібліотеки, потрібно зробити відповідний цьому JSON тип даних, який ми будемо використовувати у наших функціях.

```
data ErrorMsg = ErrorMsg
                { message :: T.Text
                , error :: Int -- Can't do this
                } deriving Show
```

Рис. 1.10 Перша версія типу помилки

Проблема цього визначення полягає в тому, що слово error — зарезервоване слово, що вже має визначення в Haskell. Для того, щоб уникнути цієї проблеми, треба трохи змінити визначення нашого типу помилки:

```
data ErrorMsg = ErrorMsg
                { message :: T.Text
                , errorCode :: Int
                } deriving Show
```

Рис. 1.11 Коректний клас помилки

Тепер ми отримали правильний код, але ще одну додаткову проблему — при спробі зробити цей клас екземпляром FromJSON з використанням Generic, Aeson буде очікувати поле errorCode замість error. Щоб пояснити, що імена у полів різні, потрібно вручну написати визначення екземпляру класу FromJSON:

```
instance FromJSON ErrorMsg where
    parseJSON (Object v) =
        ErrorMsg <$> v .: "message"
        ErrorMsg <*> v .: "error"
```

Рис. 1.12 Екземпляр ErrorMsg класу FromJSON

Єдиний метод, який ми перевизначаємо — це метод parseJSON. Він приймає Object v що представляє собою об’єкт, з яким ми працюємо. Далі ми беремо всі поля типу ErrorMsg по порядку, та дістаємо з об

Те ж саме потрібно зробити і з визначенням екземпляру ToJSON, якщо ми хочемо використовувати цей клас для конвертації даних у JSON:

```
instance ToJSON ErrorMsg where
    toJSON (ErrorMsg message errorCode) =
        object [ "message" .= message
               , "error" .= errorCode
               ]
```

Рис. 1.13 Екземпляр ErrorMsg класу ToJSON

Тут ми перевизначаємо функцію toJSON, що отримує один параметр нашого типу ErrorMsg, та повертає об’єкт JSON. Для кожного поля ми

створюємо пару ключ-значення за допомогою оператора “.=” та будуємо з них JSON-об’єкт.

Ці три функції — encode, decode та eitherDecode — є базовими у Aeson, і знаючи такий спосіб їх використання, вже можна використовувати Aeson у своїх проекти. Але при цьому, Aeson пропонує значно більш глибокий функціонал, і використовується у багатьох модулях у якості допоміжної бібліотеки.

1.3 Інші засоби розробки

У цьому розділі стисло описано інші використані засоби розробки та деякі їх особливості.

1.3.1 Бібліотека Wreq

У розробці практичної частини даної курсової було використано бібліотеку Wreq [5]. Бібліотека Wreq [6] — це бібліотека для написання клієнтських HTTP-запитів, зосереджена на легкості використання.

Так виглядає простий GET-запит, написаний з використанням даної бібліотеки, а також отримання статусу його виконання:

```
response ← get "http://httpbin.org/get"
print response ^. responseStatus . statusCode
-- out: 200
```

Рис. 1.14 Простий GET-запит у Wreq

Ця бібліотека використовує специфічну структуру, лінзи (lens), щоб ефективно оперувати запитами. Одна із таких лінз — це responseStatus, приклад використання якої продемонстровано на малюнку 14.

При цьому немає потреби глибоко розуміти використання лінз, щоб ефективно оперувати даною бібліотекою. Основне їх призначення полягає в тому, щоб “зосередити увагу” на певній частині даних в Haskell. Таким чином,

за допомогою оператора “^.” з відповіді сервера виділяється статус (за допомогою лінзи responseStatus), із якого потім отримується код статусу.

До запитів також можна додавати параметри, що передаватимуться напряму в посиланні, за допомогою такого механізму:

```
let opts = defaults & param "limit" .~ ["25"]
                    & param "code" .~ ["done"]
response ← getWith opts "http://httpbin.org/get"
-- The url is "http://httpbin.org/get?limit=25&code=done"
```

Рис. 1.15 Передача параметрів через Wreq

Ми передаємо у посиланні два параметра — limit та done, з певними значеннями. При цьому, змінна defaults визначає список параметрів за замовчанням, тобто ті параметри, що є однаковими для всіх запитів. До них через лінзу param можна додавати ключі та значення — додаткові параметри. Нарешті, виклик функції get з використанням заданих параметрів, можна зробити через функцію getWith. Вона має таку сигнатуру:

```
getWith :: Network.Wreq.Options → String → IO (Response ByteString)
```

Рис. 1.16 Сигнатура функції getWith

Із сигнатури легко побачити, що ця функція приймає два параметри — один типу Network.Wreq.Options, що визначає параметри запиту, а інший String, що визначає посилання, на яке потрібно відправити запит.

З отриманого IO (Response ByteString), як було відображено вище, за допомогою дінз можна отримувати різноманітні дані та оперувати ними. Існує ще одна лінза, яка була використана при проектуванні програми — responseBody, що дозволяє отримати тіло запиту.

```
response ← get "http://httpbin.org/get"
print $ response ^. responseBody
-- out: ... response body ...
```

Рис. 1.17 Використання лінзи responseBody

Wreq має ще багато методів для роботи з запитами. З її допомогою, можна додавати до запитів заголовки (header), відправляти POST, UPDATE, DELETE та інші запити. Також є багато лінз, що дозволяють легко доступитись до різних даних із запитів та відповідей. Таким чином, бібліотека Wreq надає набір зручних інструментів для роботи з клієнтськими HTTP-запитами. У практичній частині використано саме getWith запит із цієї бібліотеки.

1.3.2 Веб-ресурс Reddit та доступ до його даних

Для демонстрації прикладу використання Aeson потрібно завантажити реально використовувані JSON-дані. Джерелом таких даних у цій роботі є дані з веб-сайту Reddit [8]. Reddit — це онлайн-сервіс, за допомогою якого користувачі можуть ділитися інформацією та обговорювати її. Вміст цього сервісу розділено на різні підсайти, так звані сабреддіти (subreddits). Кожен сабреддіт має своє власне унікальне ім’я, яке записується як (“r/” + назва_сабреддіту) — наприклад, r/haskell.

Reddit надає відкритий доступ до свого API — тобто хто завгодно, зареєструвавши свій застосунок, може його використовувати та отримувати потрібні дані, модифікувавши запит певним чином [7].

Також, до будь-якого посилання на цьому сайті можна додати розширення “.json” та побачити інформацію тієї ж сторінки, але у форматі JSON. Саме такі дані, автоматично завантажені з мережі інтернет, і було опрацьовано у практичній частині даної роботи.

2. ПРОЕКТУВАННЯ ТА НАПИСАННЯ ПРОГРАМИ ДЛЯ ДЕМОНСТРАЦІЇ РОБОТИ БІБЛІОТЕКИ AESON

2.1 Ідея та основа програми

Для створення практичної частини було обрано задачу, що має практичну цінність. Потрібно завантажити дані про топ-25 найактуальніших на даний момент публікацій з сабреддіту. Дані, що потрібно отримати – заголовок публікації та пряме посилання на неї.

Для демонстрації було обрано сабреддіт, присвячений мові програмування Haskell (r/haskell).

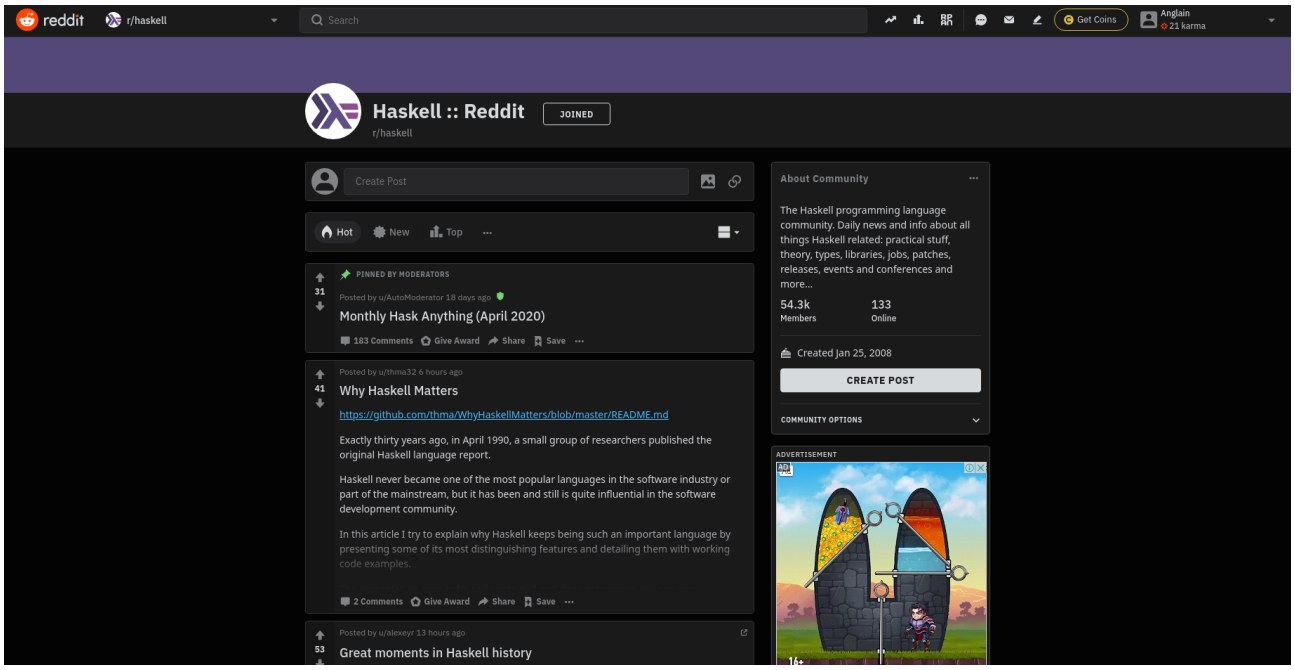

Рис. 2.1 r/haskell

Як вже було зазначено вище, якщо додати до посилання на цей сабреддіт “.json” – можна отримати дані цієї сторінки у форматі JSON, як нам і потрібно:

```
{
  "kind": "Listing",
  "data": {
    "modhash": "b1iqg75rokc49604697652905d5703edb43cd0842d131bde2d",
    "dist": 26,
    "children": [
      {
        "kind": "t3",
        "data": {
          "approved_at_utc": null,
          "subreddit": "haskell",
          "selftext": "This is your opportunity to ask any questions you feel do
          "author_fullname": "t2_6l4z3",
          "saved": false,
          "mod_reason_title": null,
          "gilded": 0,
          "clicked": false,
          "title": "Monthly Hask Anything (April 2020)",
          "link_flair_richtext": [],
          "subreddit_name_prefixed": "r/haskell",
          "hidden": false,
          "pwls": 6,
          "link_flair_css_class": null,
          "downs": 0,
          "hide_score": false,
          "name": "t3_fsgqd6",
          "quarantine": false,
          "link_flair_text_color": "dark",
          "author_flair_background_color": null,
          "subreddit_type": "public"
```


Рис. 2.2 Фрагмент даних з /r/haskell/hot.json

2.2 Завантаження потрібних даних

Для того, щоб виконати обробку необхідних та актуальних даних, необхідно спочатку завантажити JSON зі всіма даними по потрібному сабреддіту.

Є два варіанти вирішення даної проблеми – зареєструвати застосунок та використати функції, що надає Reddit API, або ж просто завантажити сторінку у форматі JSON як показано вище, і обробити та виділити потрібні дані самостійно.

У цій роботі використано другий метод. За допомогою бібліотеки Wreq ми робимо GET-запит до потрібного сайту (reddit.com/r/haskell/hot), до якого також додаємо параметр “limit=25”, щоб обмежити кількість публікацій до 25. Після

цього використовуємо лінзу responseBody щоб виділити чистий JSON з відповіді, яку ми отримуємо.

Таким чином ми отримали стрічку з JSON-даними, яку тепер можемо обробити так, як нам потрібно

```
-- Defining get request options and getting the response
let opts = defaults & param "limit" .~ ["25"]
response ← getWith opts "https://www.reddit.com/r/haskell/hot.json"
BC.putStrLn "Request returned status code:"
print $ response ^. responseStatus . statusCode
```

Рис. 2.3 Завантаження JSON з r/haskell

2.3 Побудова типів даних та робота з ними

Тепер, коли чистий JSON завантажено, можна виконувати його обробку. Тут вступає в дію Aeson. Для того, щоб використати функцію decode (або eitherDecode), нам потрібен тип, що буде екземпляром класу FromJSON. Цей тип повинен відображати JSON-дані.

Гарна новина полягає в тому, що нам немає потреби описувати всі дані. Ми можемо зробити відповідники лише для тих полів, які необхідно обробити щоб дістати заголовки та посилання на публікації. Згідно з цією інформацією, потрбно дослідити структуру отриманого JSON та, нарешті, створити відповідні типи.

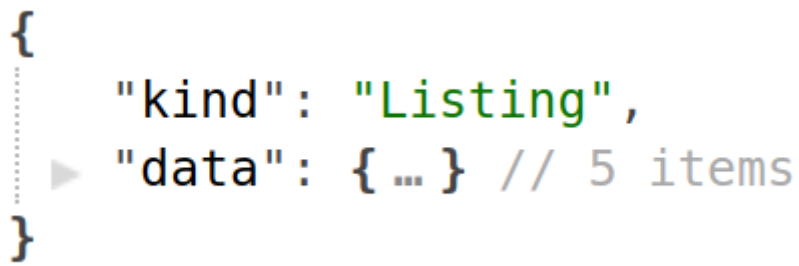

Рис. 2.4 Основа для RedditResponse

```
data RedditResponse = RedditResponse
                      { rrData :: RedditData
                      } deriving (Show, Generic)
instance FromJSON RedditResponse where
    parseJSON (Object v) =
        RedditResponse <$> v .: "data"
```

Рис. 2.5 Тип даних RedditResponse

На найвищому рівні нашого JSON знаходиться об’єкт, що має два поля - “kind” типу рядок, та “data”, у який вкладено інший об’єкт. Як і було зазначено, нам немає жодної необхідності виділяти всі дані, тому використовуємо в нашому типі даних лише поле “data”. Але маємо конфлікт, розглянутий у першій частині даної роботи — слово data є зарезервоваинм, тому потрібно вручну перевизначити метод parseJSON для даного типу. Поле rrData має тип RedditData, що буде створено нижче.


```
"data": {
    "modhash": "b1iqg75rokc49604697
    "dist": 26,
    "children": [ … ], // 26 items
    "after": "t3_g2v12v",
    "before": null
}
```


Рис. 2.6 Основа для RedditData

```
data RedditData = RedditData
                  { children :: [RedditDataChild]
                  } deriving (Show, Generic)
instance FromJSON RedditData
```

Рис. 2.7 Тип даних RedditData

Цього разу конфлікти імен відсутні, тому можна легко визначити екземпляр потрібного класу для даного типу. І знову, використовуємо лише поле “children”, оскільки всі інші поля не містять потрібних нам даних. Обробляємо наш об’єкт далі:

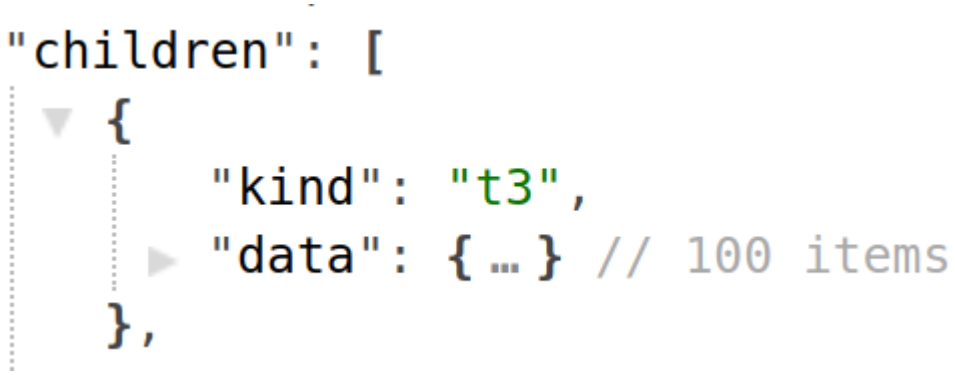

Рис. 2.8 Основа для RedditDataChild

Знову конфлікт, знову прописуємо вручну parseJSON та знову ігноруємо зайві поля. Підбираємось до останнього потрібного нам рівня вкладеності:

```
data RedditDataChild = RedditDataChild
                         { rdcData :: MainData
                         } deriving Show
instance FromJSON RedditDataChild where
    parseJSON (Object v) =
        RedditDataChild <$> v .: "data"
```

Рис. 2.9 Тип даних RedditDataChild

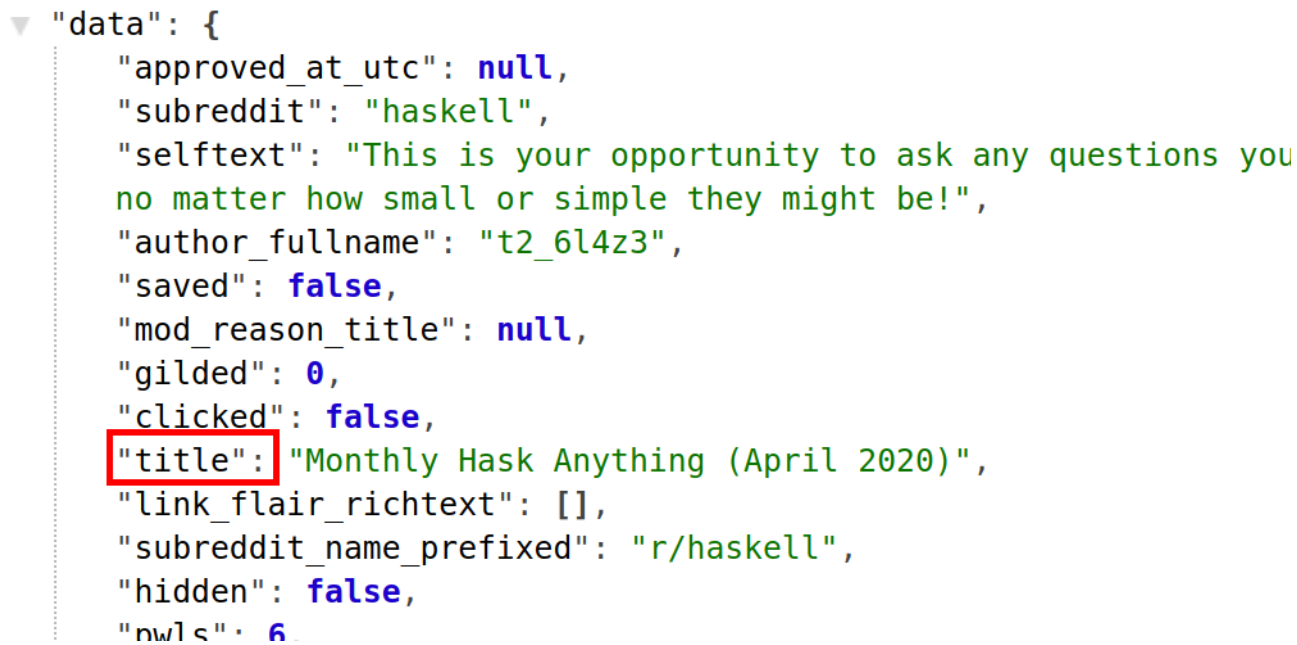

Рис. 2.10 Останній рівень вкладеності, основа для MainData

Можемо побачити, що саме на цьому рівні кожна публікація містить потрібні нам дані — title та permalink. Створюємо останній тип даних, який цього разу міститиме поля лише знайомих нам стандартних типів:

```
data MainData = MainData
                { title :: T.Text
                , permalink :: T.Text
                } deriving (Show, Generic)
instance FromJSON MainData
```

Рис. 2.11 Тип даних MainData

Самі title та permalink виглядають так:

```
"data": {
    "..." : "...",
    "title": "Monthly Hask Anything (April 2020)",
    "..." : "...",
    "permalink": "/r/haskell/comments/fsgqd6/monthly_hask_anything_april_2020/",
    "..." : "..."
},
```

Рис. 2.12 Приклад title та permalink

Ми створили чотири типи даних, за допомогою яких ми будемо обробляти наш JSON. Тепер, оскільки публікацій багато, нам потрібно створити декілька допоміжних функцій.

Перше, що потрібно зробити — це визначитись із форматом, у якому зручно було б зберігати потрібні нам дані. У цьому випадку, буде доречно зберігати наші заголовки та посилання у списку “двійок” (структура, що існує у мові Haskell) — списку пар з двох значень. Створимо функцію, яка прийматиме список дітей (RedditDataChild) та повертатиме список двійок:

```
-- Getting list of title and link pairs
getListFromChild :: [RedditDataChild] → [(T.Text, T.Text)]
getListFromChild [] = []
getListFromChild ((RedditDataChild(MainData title permalink)) : xs) =
    (title, permalink) : getListFromChild xs
```

Рис. 2.13 Отримуємо список двійок із RedditDataChild

Після цього ми можемо створити ще одну допоміжну функцію, щоб нам не доводилось кожного разу витягати RedditDataChild з RedditResponse. Ця допоміжна функція виглядає отак:

Тепер ми маємо можливість виділити потрібні нам дані за допомогою типу
RedditResponse.

```
getTitlesAndLinks :: RedditResponse → [(T.Text, T.Text)]
getTitlesAndLinks (RedditResponse (RedditData ch)) = getListFromChild ch
```

Рис. 2.14 Отримуємо список двійок із RedditResponse

2.4 Обробка даних за допомогою створеного типу RedditResponse

Щоб обробити дані, викличемо функцію eitherDecode:

```
-- Parsing the json data with RedditResponse type
let jsonData = response ^. responseBody
let parsedJson = eitherDecode jsonData :: Either String RedditResponse
```

Рис. 2.15 Обробка r/haskell

Тепер ми отримаємо Left String у змінній parsedJson якщо щось піде не так, або ж Right RedditResponse якщо все пройшло успішно. Цей результат потрібно обробити:

```
case parsedJson of
    -- If something goes wrong with parsing, we print the error message
    Left err → print err
    -- If everything is ok, print the info
    Right resp → do
        let titlesAndLinks = getTitlesAndLinks resp
        printRedditInfo titlesAndLinks
```

Рис. 2.16 Обробка результату декодування r/haskell

Ми також створили додаткову функцію printRedditInfo, щоб мати можливість вивести отримані дані у консоль.

2.5 Збереження у файл

На даний момент, всі отримані дані обробляються та виводяться в консоль. Але хотілося б не запускати програму кожного разу, коли нам потрібно подивитись на ці дані. Щоб цього уникнути, необхідно виконати збереження даних у файл. Для початку, створимо функцію що зберігатиме в файл кожну двійку з нашого масиву як окремий рядок:

Потім, викликаємо цю функцію в разі успішної обробки наших JSON-даних:

```
-- Saves titles and links to the file
saveRedditInfo :: [(T.Text, T.Text)] → String → IO ()
saveRedditInfo [] _ = do print "Nothing to save. [saveRedditInfo]"
saveRedditInfo ll file = do
    M.forM_ ll (BL.appendFile file . BL.append "\n" . BLU.fromString . show)
```

Рис. 2.17 Додаємо в файл кожну двійку

```
case parsedJson of
    -- If something goes wrong with parsing, we print the error message
    Left err → print err
    -- If everything is ok, print the info and put it into file
    Right resp → do
        let titlesAndLinks = getTitlesAndLinks resp
        printRedditInfo titlesAndLinks
        BL.writeFile resultPosts ""
        saveRedditInfo titlesAndLinks resultPosts
```

Рис. 2.18 Використовуємо збереження в файл

Ми отримали файл, у якому зберігаються всі корисні для нас дані.

2.6 Остання ітерація програми

У якості останньої модифікації ми додамо до кожного посилання префікс, потрібний для того щоб ці посилання стали повними. Інакше, формат таких посилань виглядатиме як “r/haskell/comments/...”.

```
reddit :: T.Text
reddit = "https://www.reddit.com"
```

Рис. 2.19 Префікс для посилань

Модифікуємо нашу функцію getListFromChild:

Тепер наш застосунок зберігатиме всі посилання у повній формі.

```haskell
-- Getting list of title and link pairs
getListFromChild :: [RedditDataChild] → [(T.Text, T.Text)]
getListFromChild [] = []
getListFromChild ((RedditDataChild(MainData title permalink)) : xs) =
    (title, (T.append reddit permalink)) : getListFromChild xs
```

Рис. 2.20 Остання модифікація

Висновки

У результаті проведення дослідження можна зробити висновок, що бібліотека Aeson є дійсно зручним та швидким інструментом для роботи із JSON у мові програмування Haskell. При роботі з цією бібліотекою слід завжди пам’ятати про рамки типізації, які встановлює Haskell, тим самим забезпечуючи безпечну та безперебійну роботу даної бібліотеки.

Було розглянуто процес розробки власного застосунку, що завантажує та виконує обробку даних з веб-ресурсу Reddit. Даний застосунок можна легко використовувати та за потреби розширювати.

Список використаних джерел

1. JSON [Електронний ресурс]
   Режим доступу: https://www.json.org/json-en.html
2. Kurt Will. Get programming with Haskell. - 2018. - 616 с.
3. Aeson: Fast JSON parsing and encoding. - Data.Aeson [Електронний ресурс]
   Режим доступу: https://hackage.haskell.org/package/aeson-1.4.7.1/docs/Data-Aeson.html
4. Aeson: the tutorial [Електронний ресурс]
   Режим доступу: https://artyom.me/aeson
5. A wreq tutorial [Електронний ресурс]
   Режим доступу: http://www.serpentine.com/wreq/tutorial.html
6. wreq: An easy-to-use HTTP client library. - Network.Wreq [Електронний ресурс]
   Режим доступу: https://hackage.haskell.org/package/wreq-0.5.3.2/docs/Network-Wreq.html
7. reddit.com: api documentation [Електронний ресурс]
   Режим доступу: https://www.reddit.com/dev/api#listings
8. reddit: the front page of the internet [Електронний ресурс]
   Режим доступу: https://www.reddit.com/
9. Prelude – The University of Glasgow, 2001 – 653 с.

Додаток А. Лістинг програмного коду

Основний файл Main.hs

```
module Main where


import Data.Text as T

import Data.ByteString.Lazy as BL

import Data.ByteString.Lazy.Char8 as BC

import GHC.Generics

import System.Directory

import Control.Monad


import Data.Aeson

import Network.Wreq

import Control.Lens



import Lib


main :: IO ()

main = do

    BC.putStrLn "\n\n ==== Hello from Aeson project! Everything compiled good.
==== \n\n"
```

```
let resultPosts = "r_haskell_hot_posts.txt"

-- Defining get request options and getting the response
let opts = defaults & param "limit" .~ ["25"]
response <- getWith opts "https://www.reddit.com/r/haskell/hot.json"
BC.putStrLn "Request returned status code:"
print $ response ^. responseStatus . statusCode

-- Parsing the json data with RedditResponse type
let jsonData = response ^. responseBody
let parsedJson = eitherDecode jsonData :: Either String RedditResponse

case parsedJson of
    -- If something goes wrong with parsing, we print the error message
    Left err -> print err
    -- If everything is ok, print the info and put it into file
    Right resp -> do
        let titlesAndLinks = getTitlesAndLinks resp
        BL.writeFile resultPosts ""
        printRedditInfo titlesAndLinks
        saveRedditInfo titlesAndLinks resultPosts
```

Lib.hs

```
module Lib (
        RedditResponse
      , getTitlesAndLinks
      , printRedditInfo
      , saveRedditInfo
      ) where

import System.IO
import Data.Text as T
import Data.ByteString.Lazy as BL
import Data.ByteString.Lazy.UTF8 as BLU
import GHC.Generics

import Control.Monad as M

import Data.Aeson



reddit :: T.Text
reddit = "https://www.reddit.com"

---------- Data types to parse JSON ----------
```

```
-- MainData json type contains all the information about the post
-- Although, of all info we only need the title and permalink params
data MainData = MainData
              { title :: T.Text
              , permalink :: T.Text
              } deriving (Show, Generic)
instance FromJSON MainData


data RedditDataChild = RedditDataChild
                     { rdcData :: MainData
                     } deriving Show
instance FromJSON RedditDataChild where
    parseJSON (Object v) =
        RedditDataChild <$> v .: "data"

data RedditData = RedditData
                { children :: [RedditDataChild]
                } deriving (Show, Generic)
instance FromJSON RedditData

data RedditResponse = RedditResponse
```

```
                { rrData :: RedditData
                } deriving Show
instance FromJSON RedditResponse where
    parseJSON (Object v) =
        RedditResponse <$> v .: "data"


------------------ Functions to work with these data types ------------------

-- Getting list of title and link pairs
getListFromChild :: [RedditDataChild] -> [(T.Text, T.Text)]
getListFromChild [] = []
getListFromChild ((RedditDataChild(MainData title permalink)) : xs) =
    (title, (T.append reddit permalink)) : getListFromChild xs

getTitlesAndLinks :: RedditResponse -> [(T.Text, T.Text)]
getTitlesAndLinks (RedditResponse (RedditData ch)) = getListFromChild ch

-- Prints titles and links to the console
printRedditInfo :: [(T.Text, T.Text)] -> IO ()
printRedditInfo [] = do print ""
printRedditInfo ll = do
    M.forM_ ll (print)
```

```
-- Saves titles and links to the file
saveRedditInfo :: [(T.Text, T.Text)] -> String -> IO ()
saveRedditInfo [] _ = do print "Nothing to save. [saveRedditInfo]"
saveRedditInfo ll file = do
    M.forM_ ll (BL.appendFile file . BL.append "\n" . BLU.fromString . show)
```