Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 21.Б08-мм

Покрытие бэкенда Desbordante тестами

Фофанов Максим Александрович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
Ассистент кафедры информационно-аналитических систем Г. А. Чернышев

Санкт-Петербург
2023

Оглавление

Введение

Тестирование является важным аспектом разработки ПО. Оно позволяет проверить, что приложение работает так, как ожидается, и обнаружить неисправности и ошибки. Тесты также могут помочь увеличить надежность приложения и упростить процесс его поддержки и обновления.

Существуют различные типы тестов, которые могут быть использованы в зависимости от целей тестирования и специфики приложения. Некоторые из самых распространенных типов — это юнит-тесты, интеграционные тесты, функциональные тесты и нагрузочные тесты.

Тестирование веб-приложений также требует использования специальных инструментов и фреймворков, которые помогают автоматизировать процесс тестирования и упрощают процесс написания кода.

Важно, чтобы тесты были написаны хорошо, чтобы они были релевантными и покрывали важные функции и сценарии использования. Также важно регулярно запускать тесты, чтобы обеспечить стабильность и целостность приложения во время его разработки.

При тестировании веб-приложений дополнительно необходимо убедиться в правильном функционировании связи сервер - клиент, так как от этого напрямую зависит работоспособность сервиса. Для этого существуют библиотеки позволяющие имитировать отправку и получение HTTP-запросов.

Данная работа - покрытие тестами бэкенда веб-приложения Desbordante. Desbordante — это высокопроизводительный профилятор данных способный найти множество различных закономерностей в данных с использованием большого количества алогоритмов.
Desbordante так же позволяет проводить очистку данных на основе работы этих алгоритмов.

1. Постановка задачи

Целью работы является покрытие бэкенда веб-приложения Desbordante[1] тестами. Для её выполнения были поставлены следующие задачи:

1. Ознакомиться с языком запросов к API GraphQL и составить краткий обзор его возможностей.
2. Ознакомиться с GraphQL-схемой Desbordante. Написать краткий обзор существующих запросов и типов.
3. Реализовать набор тестов для различных частей схемы:
   - Создание и обработка аккаунтов пользователей.
   - Фильтрация результатов выполнения задач.
   - Права доступа.
   - Резолверы для основных типов.
   - Валидация параметров для основных примитивов.
4. Произвести апробацию и анализ результатов.

[1] https://www.github.com/Mstrutov/Desbordante (дата доступа: 4 января 2023 г.)

2. Вводная информация

2.1. Desbordante

Desbordante — это высокопроизводительный профайлер данных, который способен обнаруживать различные паттерны в данных с помощью различных алгоритмов. Он также позволяет запускать сценарии очистки данных с использованием этих алгоритмов. Закономерности, которые могут присутствовать в данных, описываются с помощью различных примитивов. Сам по себе примитив — это некоторое описание правила, действующего над данными (или их частью), описанное математическими методами. Больше информации о проекте можно найти в блоге компании Юнидата [3].

2.2. GraphQL

GraphQL — это язык запросов к API. Он позволяет клиенту запрашивать точно то, что ему нужно, что делает его более гибким и эффективным вариантом по сравнению с REST API. GraphQL позволяет клиенту запрашивать вложенные данные и выполнять мутации. Он также имеет систему типов, которая используется для обеспечения безопасного выполнения на сервере. С помощью GraphQL клиент имеет больше контроля над получаемыми данными, что приводит к уменьшению ненужного передачи данных и ускорению запросов.

Запросы в GraphQL делятся на два типа: query и mutation. Query используются для чтения данных, а mutation для их записи. Так же GraphQL позволяет передавать запросам аргументы, причем часть аргументов может относиться к вложенному запросу. Так же для запрашиваемых полей можно вводить alias-ы чтобы решать конфликты имён.

Если несколько запросов делят одну похожую часть которая слишком мала для полноценного подзапроса — из неё можно сделать fragment. Фрагменты позволяют пользователю сделать выборку из полей конкретного типа и затем использовать её везде где необходимо.

```
query getHuman($id: ID) {
    human(id: $id) {
        name
        age
    }
}
```

Listing 1: Пример запроса типа query на GraphQL, где getHuman — это название запроса, id — принимаемый аргумент типа ID, query — тип запроса, name и age — возвращаемые поля.

Обычно запрос на GraphQL имеет следующую структуру: тип запроса (query или mutation), его название, список аргументов, название вызываемого запроса, список его аргументов, выборка полей, причем в этом месте может вызываться ещё один вложенный запрос который тоже может иметь свои параметры и так далее.

Помимо всего вышеперечисленного в GraphQL возможности для работы со сложными типами: перечислениями, объединениями (значения типа могут принимать один из нескольких содержащихся в объединении типов), а так же массивами в том числе и для кастомных типов.

Подробнее о преимуществах GraphQL над REST можно прочитать в статье Alik Send на Хабр [2]. Полная документация доступна на официальном сайте [5].

2.3. GraphQL-схема Desbordante

GraphQL-схема Desbordante состоит из нескольких частей:

1. Запросы связанные с авторизацией: createUser, logIn, logOut, refresh
2. Запросы связанные с созданием задач: createMainTaskWithDatasetChoosing, createSpecificTask, deleteTask
3. Запросы связанные с мониторингом информации в базе данных: datasetInfo, taskInfo, user и их версии для администрации: datasets, tasksInfo, users позволяющие ознакомиться сразу со всей информацией.

4. Остальные запросы связанные с получением фидбека, подтверждением адреса электронной почты и получением прав доступа.

Для удобства использования создано достаточно много кастомных типов которые наследуются от нескольких гипер-типов (как правило начинаются со слов Abstract или Base). Вот некоторые примеры таких типов:

- PrimitiveTaskConfig и ARTaskConfig, FDTaskConfig, CFDTaskConfig — типы описывающие настройки для соответствующих примитивов.
- TaskWithDepsResult и ARTaskResult, FDTaskResult, CFDTaskResult — типы описывающие результаты выполнения работы соответствующих им примитивов.
- FilteredDepsBase и FilteredFDs, FilteredCFDs, FilteredARs — типы описывающие результаты работы фильтров над соответствующими примитивами.
- Типы с названиями заканчивающимися на Answer, например CreateUserAnswer — это типы сообщающие об успешном выполнении какого-то процесса и содержащие информацию о нём.
- Типы заканчивающиеся на Error — это типы сообщающие об ошибке при выполнении какого-то процесса, содержащие текстовое описание и детали.

2.4. Обзор технологий тестирования веб-приложений

Так как бэкенд Desbordante написан на TypeScript, то был проведён обзор фреймфорков для тестирования JavaScript/TypeScript. Существует три популярных фреймворка под написание тестов для TypeScript: Jest, AVA, Mocha. Далее рассмотрим каждый из них:

- Jest — это фреймворк тестирования, разработанный и поддерживаемый Facebook. Он предлагает достаточно простой интерфейс и

предоставляет многих утилитных функций для написания тестов. Jest также достаточно быстр, благодаря использованию виртуального браузера и умного запуска, который перезапускает только те тесты, которые относятся к изменениям в вашем коде. Полную информацию можно найти на официальном сайте проекта [6].

- AVA — это библиотека тестирования, которая уделяет особое внимание многопоточному и асинхронному тестированию. Она запускает тесты одновременно в отдельных процессах, что может ускорить время их выполнения. AVA также имеет простой API и поддерживает функции, такие как охват кода и CI. Подробности можно найти в официальном репозитории [4].

- Mocha — это популярная и гибкая библиотека тестирования, которая позволяет выбирать собственную библиотеку проверок и запускать тесты в браузере или в командной строке. Она поддерживает асинхронное тестирование и имеет большую экосистему плагинов. С документацией и другими подробностями можно ознакомиться на официальном сайте [7].

В итоге все приведённые фреймворки подходят под поставленную задачу, но так как в проекте уже был ряд тестов написанных с помощью Jest, поэтому было решено остановиться на нём.

3. Обзор решения

Для анализа GraphQL-схемы была использована утилита GraphQL Voyager[2]. Реализованный функционал:

- Запускать GraphQL-запросы из файлов с расширением .graphql и получать результаты их выполнения.
- Получать доступ (токены access и refresh) для разных ролей (аноним, обычный пользователь, администратор).
- Создавать тестовый клиент для Apollo-server с возможностью прикладывать к запросу целевые header-ы.

Для некоторых из случаев описанных в постановке задачи были написаны специализированные инструменты:

- Для тестов фильтров — функции дающие возможность размещать и удалять из базы данных тестовые данные, а так же получать список зависимостей уже обработанных на сервере.
- Для тестов резолверов — функции для создания задач и получения из базы данных правильных параметров.

[2] https://www.ivangoncharov.github.io/graphql-voyager (дата доступа: 4 января 2023 г.)

4. Практическая часть

4.1. Реализация

В процессе реализации были, по возможности, применены все лучшие практики, в том числе указанные в книге Ли Копленда [1]. Была выбрана следующая организация кода по директориям: большие разделы указанные в постановке задачи были вынесены в отдельные папки, внутри которых поделены на файлы исходя из подзадач. Внутри файлов тесты были организованы с помощью логических блоков describe (группа тестов сходная по какому-то признаку) и it (конкретный тест). Для каждой группы тестов с помощью функций beforeAll и afterAll проводилось создание объектов в базе данных (если это необходимо) и их удаление соответственно. Общие функции были вынесены в файл util в общей директории, более специфичные инструменты в файлы util внутри соотвествующих папок.

4.2. Применение

Примером использования написанных тестов будет встроить их в CI и запускать каждый раз перед коммитом/отправкой данных в репозиторий.

5. Апробация

5.1. Исследовательские вопросы и метрики

При написаннии тестов для любого веб-приложения основными целями являются:

- Достигнуть максимального процента тестового покрытия.
- Выявить и устранить как можно больше проблем в процессе написания тестов.

В качестве метрик для оценки полученного результата были выбраны следующие две:

- Процент покрытия кода.
- Чистота кода[3].

5.2. Результаты

По результатам выполенения работы были получены проценты покрытия указаные в таблице приведённой ниже, а так же был обнаружен ряд проблем, а именно:

- Фильтры требующие наличия правой части работают некорректно (для примитивов FD и CFD).
- Фильтры работающие с левой частью (для примитива CFD), работают не корректно при условии, что в ней содержится несколько элементов.
- Некоторые поля в GraphQL-схеме могут содержать null, хотя не должны.
- Представления результатов работы примитивов в базе данных нуждается в переработке.

[3]Измеряется с помощью линтеров ESLint и Prettier

Помимо этого были устранены все ошибки на которые указывал линтер и большая часть предупреждений. Авторизация имеет такой маленький процент покрытия относительно других частей потому что в этот раздел входить ещё и отправка сообщений на почту, восстановление аккаунтов — функционал не входивший в задачи поставленные перед выполнением работы.

Часть схемы	Процент покрытия
Авторизация	34.00
Фильтры	89.45
Права доступа	100.00
Основные резолверы	100.00
Валидация параметров	100.00

Заключение

Результаты:

- Составлен краткий обзор возможностей языка запросов GraphQL.
- Составлен обзор GraphQL-схемы Desbordante.
- Реализованы тесты всех потенциально проблемных частей схемы.
- Предоставлен отчёт по тому что необходимо исправить или переработать.

Подробнее с кодом можно ознакомиться по ссылке:
https://github.com/vs9h/Desbordante/pull/68

Список литературы

[1] Copeland Lee. A Practitioner's Guide to Software Test Design. — 2004.

[2] Send Alik. Статья на Habr.ru про GraphQL. — URL: https://www.habr.com/ru/post/326986/ (дата обращения: 7 января 2023 г.).

[3] Георгий Чернышев Михаил Полынцов Никита Бобров. Примитивы Desbordante: Функциональные зависимости и их применение в эксплорации и очистке данных. — URL: https://www.habr.com/ru/company/unidata/blog/679390/ (дата обращения: 7 января 2023 г.).

[4] Официальный репозиторий AVA. — URL: https://www.github.com/avajs/ava (дата обращения: 7 января 2023 г.).

[5] Официальный сайт GraphQL. — URL: https://www.graphql.org (дата обращения: 7 января 2023 г.).

[6] Официальный сайт Jest. — URL: https://www.jestjs.io (дата обращения: 7 января 2023 г.).

[7] Официальный сайт Mocha. — URL: https://www.mochajs.org/ (дата обращения: 7 января 2023 г.).