УДК 004

ПАРАЛЛЕЛЬНАЯ РАЗРАБОТКА КОМПОНЕНТОВ НЕМОНОЛИТНЫХ КЛИЕНТ-СЕРВЕРНЫХ ВЕБ ПРИЛОЖЕНИЙ.

Ковалев В. Е., студент гр. ИТб-161, 4 курс.
Научный руководитель Ванеев О. Н., кандидат технических наук, доцент
Кузбасский государственный технический университет
имени Т. Ф. Горбачева
г. Кемерово

Введение

Разработка почти всех информационных систем в настоящее требует целой команды разработчиков

Не всегда вся команда может работать абсолютно слаженно, что может выливаться в большие временные затраты на постоянные лишние действия, которые тратятся на интеграцию компонентов системы.

Так, например, веб сервисы, состоящие из нескольких частей:

- Пользовательская часть (клиент/front/фронтенд)
- Серверная часть (сервер/back/бэкенд)

Половина команды работает с одной частью, другая с другой.

Общение между такими компонентами обычно происходит посредством REST API.

Такой обмен данными должен быть строго фиксирован, данные должны передаваться в фиксированном формате и содержать определенный набор параметром.

Небольшие изменения в обмениваемых данных на одной стороне могут повлечь за собой сильные изменения на другой.

Работа в таком режиме делает обе части сильно зависимыми. Из этого следует, что часто части команды фронтендеров приходится ждать того, что другая часть команды реализует новую функцию, которая им очень нужна.

Дополнительная сложность состоит и в том, что необходимо скоординировать работу всей команды еще до начала самой работы.

То есть необходимо найти способ изолировать обе части, но, чтобы в итоге всё заработало безо всяких исправлений.

В качестве решения этой проблемы предлагается подход, на основе использования спецификации REST API с помощью фреймворка Swagger. Данный подход, который будет освещен в данной статье.

Постановка задачи

Для решения проблем параллельной работы над взаимосвязанными компонентами системы веб приложений были поставлены следующие задачи:

1. Поиск технологий автоматизации разработки клиент-серверной архитектуры
2. Разработка процесса изолированной разработки компонентов
3. Разработка процесса изолированного создания компонентов.
4. Реализация найденного подхода в написании выпускной работы
5. Сделать выводы о преимуществах разработанного процесса.

Решение

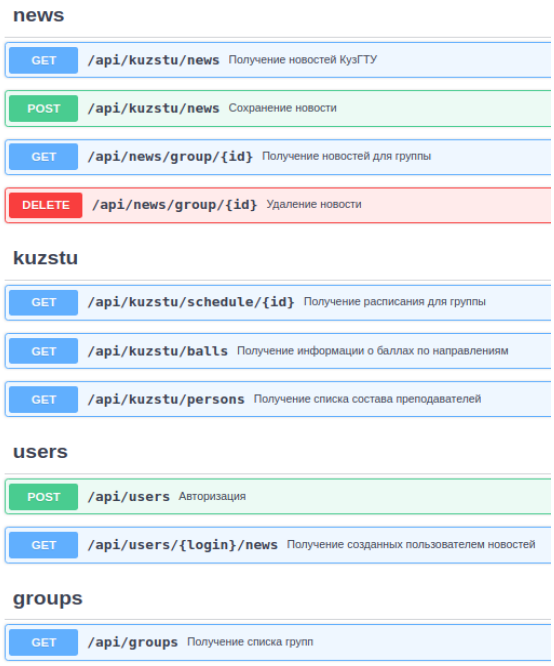
Поиск технологий надолго не затянулся. У меня уже был опыт работы со спецификациями REST API. Такие спецификации могли бы стать единственной точкой соприкосновения компонентов системы. Для написания спецификации я воспользовался фреймворком Swagger.

Swagger – это фреймворк для спецификации RESTful API. Его особенность заключается в том, что он дает возможность не только интерактивно просматривать спецификацию, но и отправлять запросы.

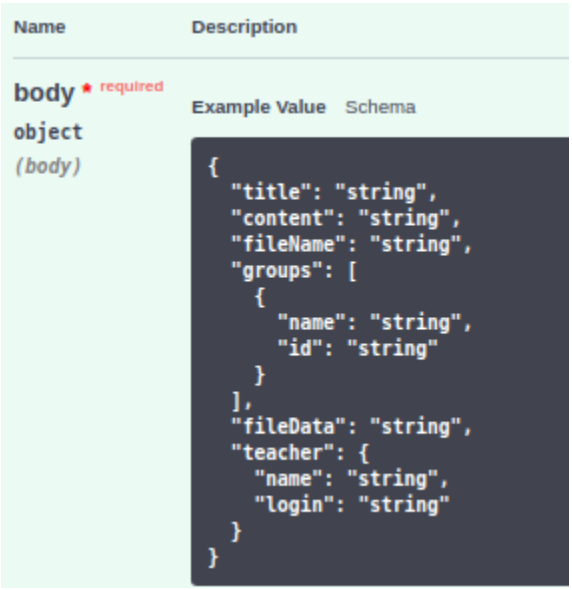
Swagger позволяет не только составлять спецификацию, но и генерировать непосредственно клиента или сервер по спецификации API Swagger.

Далее необходимо было заняться написанием самой спецификации. Для написания потребовалось небольшое количество времени. Все было написано в формате YAML.

Получилось нечто подобное:



Каждый из методов подробно можно рассмотреть при нажатии на него.



Затем лишь осталось запустить фейковый сервер, который следовал бы этой спецификации.

После запуска было показано следующее.

```
auravadima@pc:~/kafedra_backend$ prism mock -d swagger.yml
[10:48:58 PM] › [CLI] …  awaiting  Starting Prism…
[10:48:58 PM] › [CLI] i  info      GET        http://127.0.0.1:4010/api/kuzstu/news
[10:48:58 PM] › [CLI] i  info      POST       http://127.0.0.1:4010/api/news
[10:48:58 PM] › [CLI] i  info      GET        http://127.0.0.1:4010/api/news/group/facere
[10:48:58 PM] › [CLI] i  info      DELETE     http://127.0.0.1:4010/api/news/group/blanditiis
[10:48:58 PM] › [CLI] i  info      GET        http://127.0.0.1:4010/api/kuzstu/schedule/unde
[10:48:58 PM] › [CLI] i  info      GET        http://127.0.0.1:4010/api/kuzstu/balls
[10:48:58 PM] › [CLI] i  info      GET        http://127.0.0.1:4010/api/kuzstu/persons?dept_id=539
[10:48:58 PM] › [CLI] i  info      POST       http://127.0.0.1:4010/api/users
[10:48:58 PM] › [CLI] i  info      GET        http://127.0.0.1:4010/api/users/autem/news
[10:48:58 PM] › [CLI] i  info      GET        http://127.0.0.1:4010/api/groups
[10:48:58 PM] › [CLI] ▶  start     Prism is listening on http://127.0.0.1:4010
```

Сервер запустился, методы работают, осталось просто начать использовать их на фронте

При попытке получить данные по маршруту /api/kuzstu/schedule/123 получаем результат вида:

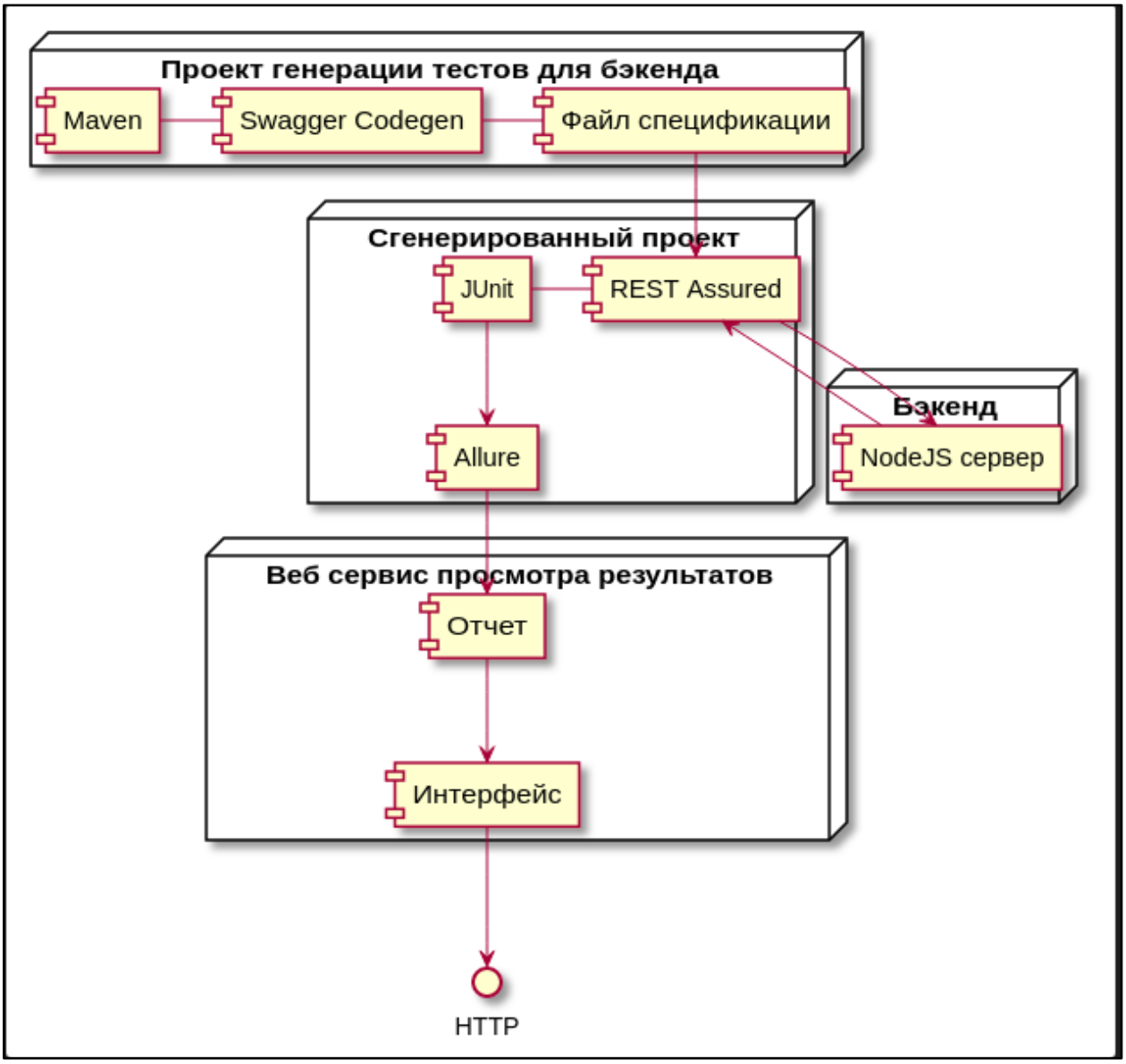

```
[
    {
        "id": "commodo tempor veniam",
        "education_group_name": "enim minim elit reprehenderit in",
        "education_group_id": "mollit",
        "day_number": "ea",
        "lesson_number": "commodo",
        "place": "proident officia dolor aliqu",
        "subgroup": "comm",
        "teacher_id": "proident",
        "teacher_name": "consequat nostrud sed sunt",
        "subject": "Lorem",
        "type": "aute ad",
        "week_type": "sed Lorem velit exercitation dol"
    },
    {
        "id": "aliquip Ut tempor",
        "education_group_name": "elit irure adi",
        "education_group_id": "ut non laboris sunt et",
        "day_number": "ipsum exercitation",
```


Все работает, как и предполагалось.

Далее настало время для бэка.

Ранее на работе я разработал проект, который мог по данной спецификации составлять готовый проект для тестирования того, что реализовано на бэке.

Примерная структура и основа работы выглядит так.



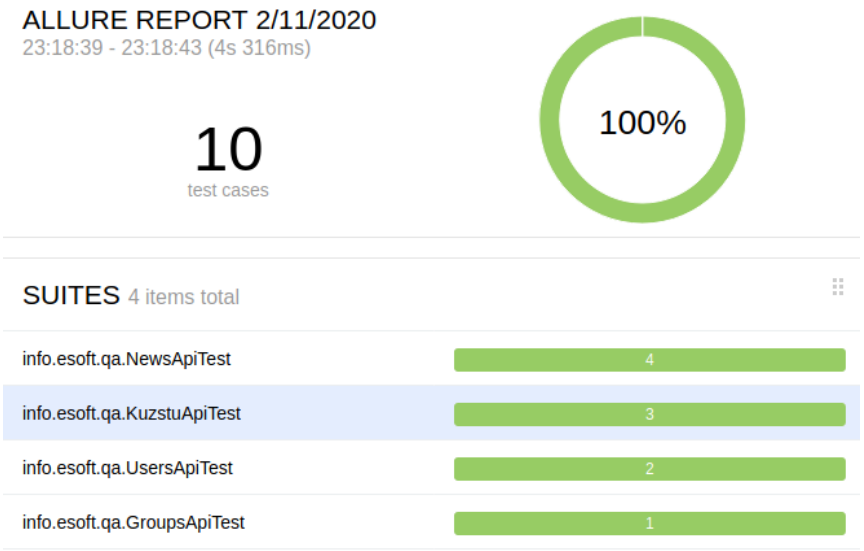
Подготовив файл-спецификацию и прогнав через утилиту, был получено решение, которое позволяло по нажатию одной кнопки узнать, что работает неправильно и что еще нужно сделать.

Сгенерировав тесты для своего бэкенда и запустив их, получаем результат:

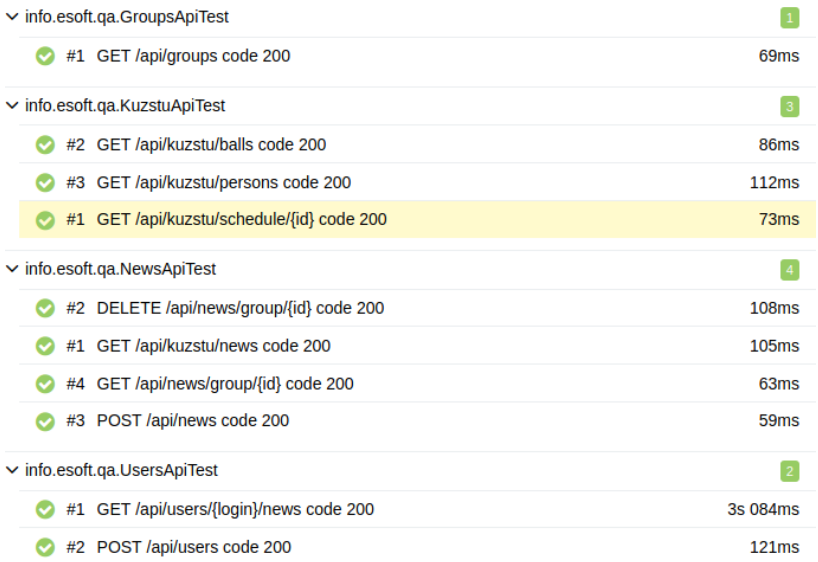
```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 10, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time:  19.602 s
[INFO] Finished at: 2020-02-11T23:18:44+07:00
[INFO] ------------------------------------------------------------
```

Но и это не все.

Также после выполнения тестов мы получаем подробный отчет благодаря фреймворку allure



Который подробно показывает результаты каждого из тестов



Данный подход не совсем подходит для разработчика одиночки, так как он не может заниматься сразу двумя проектами. Но для команды разработчиков это становится решением, так как убирает из работы лишнюю часть, в которой происходит интеграция готовых компонентов системы.

Вывод

Как показывает практика, команды разработчиков, использующие данный подход, сокращают время разработки примерно на 30%.

Плюс к этому всегда есть готовые тесты, написание которых для большого проекта занимает порядка 15-20 часов рабочего времени. Данный проект увеличивает производительность разработки тестов в 450-600 раз.

Разработка отдельных компонентов для достижения максимального результата должна вестись обособленно друг от друга.

Список литературы:

1. OpenAPI Specification[Электронный ресурс]: https://swagger.io/specification/
2. JUnit User Guide [Электронный ресурс]: https://junit.org/junit5/docs/current/user-guide/
3. Rest Assured Getting Started [Электронный ресурс]: https://github.com/rest-assured/rest-assured/wiki/GettingStarted
4. Maven documentation [Электронный ресурс]: https://maven.apache.org/guides/index.html