





..		
 pic	feat(patterns)	2 years ago
 README.md	fix(patterns); typos	8 months ago
 app-card-delivery.jar	feat(patterns)	2 years ago
 app-ibank.jar	feat(patterns)	2 years ago

Домашнее задание к занятию «2.3. Patterns»

В качестве результата пришлите ссылку на ваш GitHub-проект в личном кабинете студента на сайте netology.ru.

Все задачи этого занятия нужно делать в разных репозиториях.

Важно: если у вас что-то не получилось, то оформляйте Issue по установленным правилам.

Важно: не делайте ДЗ всех занятий в одном репозитории! Иначе вам потом придётся достаточно сложно подключать системы Continuous Integration.

Как сдавать задачи

- Инициализируйте на своём компьютере пустой Git-репозиторий
- Добавьте в него готовый файл `.gitignore`
- Добавьте в этот же каталог код ваших авто-тестов
- Сделайте необходимые коммиты
- Добавьте в каталог `artifacts` целевой сервис (`app-card-delivery.jar` для первой задачи, `app-ibank.jar` для второй задачи - см. раздел Настройка CI)
- Создайте публичный репозиторий на GitHub и свяжите свой локальный репозиторий с удалённым
- Сделайте пуш (удостоверьтесь, что ваш код появился на GitHub)
- Удостоверьтесь, что на Аррвеуог сборка зелёная
- Поставьте бейдж сборки вашего проекта в файл README.md
- Ссылку на ваш проект отправьте в личном кабинете на сайте netology.ru
- Задачи, отмеченные, как необязательные, можно не сдавать, это не повлияет на получение зачета
- Если вы обнаружили подозрительное поведение SUT (похожее на баг), создайте описание в Issue на GitHub. Придерживайтесь схемы при описании.

Настройка CI

Настройка CI осуществляется аналогично предыдущему заданию (за исключением того, что файл целевого сервиса может называться по-другому). Для второй задачи вам также понадобится указать нужный флаг запуска для "тестового режима".

Задача №1 - Заказ доставки карты (изменение даты)

Вам необходимо автоматизировать тестирование новой функции формы заказа доставки карты:

Карта с доставкой!

Казань

Выберите ваш город

03.11.2019

Выберите дату встречи с представителем банка

Фамилия и имя

Петров Иван


Укажите точно как в паспорте

Мобильный телефон

+7 900 000 00 00


На указанный номер моб. тел. будет отправлен смс-код для подтверждения заявки на карту. Проверьте, что номер ваш и введен корректно.

☒ Я соглашаюсь с условиями обработки и использования моих персональных данных

 Мы гарантируем безопасность ваших данных

Запланировать

Требования к содержимому полей, сообщения и другие элементы, по словам Заказчика и Разработчиков, "такие же, мы ничего не меняли".

Примечание*: личный совет - не забудьте это перепроверить, никому нельзя верить .

Тестируемая функциональность: если заполнить форму повторно теми же данными за исключением "Даты встречи", то система предложит перепланировать время встречи:

?


Необходимо подтверждение

У вас уже запланирована встреча на другую дату. Перепланировать?

Перепланировать

×

После нажатия на кнопку "Перепланировать" произойдёт перепланирование встречи:



Успешно!

Встреча успешно запланирована на 03.11.2019

×

Важно: в этот раз вы не должны хардкодить данные прямо в тест! Используйте Faker, Lombok, Data-классы (для группировки нужных полей) и утилитный класс-генератор данных* - см. пример в презентации.

Утилитными называют классы, у которых приватный конструктор и статичные методы.

Обратите внимание, что Faker может генерировать не совсем в нужном для вас формате.

Задача №2 - Тестовый режим

Разработчики Интернет Банка изрядно поворочав предоставили вам тестовый режим запуска целевого сервиса, в котором открыта программная возможность создания Клиентов Банка, чтобы вы могли протестировать хотя бы функцию входа.

Важно: ваша задача заключается в том, чтобы протестировать функцию входа через Web интерфейс с использованием Selenium.

Для удобства вам предоставили "документацию", которая описывает возможность программного создания Клиентов Банка через API. Вот представленное ими описание (дословно):

```
Для создания клиента нужно делать запрос вида:

POST /api/system/users
Content-Type: application/json

{
  "login": "vasya",
  "password": "password",
  "status": "active"
}

Возможные значения поля статус:
* "active" - пользователь активен
* "blocked" - пользователь заблокирован

В случае успешного создания пользователя возвращается код 200

При повторной передаче пользователя с таким же логином будет выполнена перезапись данных пользователя
```

Давайте вместе разбираться. Мы уже проходили:

- клиент-серверное взаимодействие
- HTTP-методы и коды ответов
- формат данных - JSON
- REST-assured

Мы крайне настоятельно рекомендуем ознакомиться с документацией и примерами на [Rest Assured](https://rest-assured.io).

Подключается обычным образом в Gradle:

```
testImplementation 'io.rest-assured:rest-assured:4.3.0'
testImplementation 'com.google.code.gson:gson:2.8.6'
```

Библиотека Gson нужна для того, чтобы иметь возможность сериализовать (преобразовывать) Java-объекты в JSON.

Т.е. мы не руками пишем JSON, а создаём Data-классы, объекты которых и преобразуются в JSON.

Дальнейшее использование выглядит следующим образом:

```
// спецификация нужна для того, чтобы переиспользовать настройки в разных запросах
class AuthTest {
    private static RequestSpecification requestSpec = new RequestSpecBuilder()
        .setBaseUrl("http://localhost")
        .setPort(9999)
        .setAccept(ContentType.JSON)
        .setContentType(ContentType.JSON)
        .log(LogDetail.ALL)
        .build();

    @BeforeAll
    static void setUpAll() {
        // сам запрос
        given() // "дано"
            .spec(requestSpec) // указываем, какую спецификацию используем
            .body(new RegistrationDto("vasya", "password", "active")) // передаём в теле объект, который будет преобразован в JSON
            .when() // "когда"
            .post("/api/system/users") // на какой путь, относительно BaseUrl отправляем запрос
            .then() // "тогда ожидаем"
            .statusCode(200); // код 200 OK
    }
    ...
}
```

Это не лучший формат организации, будет лучше, если как в предыдущей задаче, вы вынесете это в класс-генератор, который по требованию вам будет создавать randomного пользователя, сохраняя его через API и возвращать вам в тест.

В логах теста вы увидите:

```
Request method: POST
Request URI: http://localhost:9999/api/system/users
Proxy: <none>
Request params: <none>
Query params: <none>
Form params: <none>
Path params: <none>
Headers: Accept=application/json, application/javascript, text/javascript, text/json
Content-Type=application/json; charset=UTF-8
Cookies: <none>
Multiparts: <none>
Body:
{
  "login": "vasya",
  "password": "password",
  "status": "active"
}
```

Для активации этого тестового режима при запуске SUT нужно указать флаг `-P:profile=test`, т.е.: `java -jar app-ibank.jar -P:profile=test`.

Важно: если вы не активируете тестовый режим, любые запросы на `http://localhost:9999/api/system/users` будут вам возвращать 404 Not Found.

Вам нужно самостоятельно изучить реакцию приложения на различные комбинации случаев (вспомните комбинаторику):

- наличие пользователя;
- статус пользователя;
- невалидный логин;
- невалидный пароль.

Дополнительно: оцените время, которое вы затратили на автоматизацию, и время, за которое вы проверили бы те же сценарии вручную, используя для тестирования интерфейса браузер и Postman для доступа к открытому API.

Приложите к решению задачи, в формате:

- Время, затраченное на ручное тестирование (минут): x
- Время, затраченное на автоматизацию (минут): y

© 2021 GitHub, Inc.


Terms

Privacy

Security

Status

Docs



Contact GitHub

Pricing

API

Training

Blog

About