**Проект №6 — Back-end тестирование №1.**

Привет, участник Школы 21!🤗 В предыдущем блоке мы познакомились с ручным тестированием пользовательских интерфейсов, но этим, конечно же, ничего не ограничивается! Современные веб-приложения, как ты уже знаешь, постоянно взаимодействуют с серверами, которые тоже нужно уметь тестировать. В этом блоке ты познакомишься с API, сетевыми протоколами, HTTP-сообщениями, а также поработаешь со сваггером.

**Instructions**

Напоминаем, что все отчёты по результатам выполнения заданий тебе нужно оформлять в файлах с расширением .md. Если они уже созданы, то пересоздавать или удалять их не нужно (просто отредактируй этот файл). Все созданные отчёты и файлы тебе нужно будет загрузить в папку src/ в корне проекта (обязательно в ветку *develop*).

**Contents**

1. [Chapter I](https://edu.21-school.ru/project/62669/task#chapter-i)  
   1.1. [Общая инструкция](https://edu.21-school.ru/project/62669/task#%D0%BE%D0%B1%D1%89%D0%B0%D1%8F-%D0%B8%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D0%B8%D1%8F)
2. [Chapter II](https://edu.21-school.ru/project/62669/task#chapter-ii)  
   2.1. [Общее определение](https://edu.21-school.ru/project/62669/task#%D0%BE%D0%B1%D1%89%D0%B5%D0%B5-%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5)  
   2.2. [Что такое API и как он работает](https://edu.21-school.ru/project/62669/task#%D1%87%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-api-%D0%B8-%D0%BA%D0%B0%D0%BA-%D0%BE%D0%BD-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%D0%B5%D1%82)  
   2.3. [Способы работы API](https://edu.21-school.ru/project/62669/task#%D1%81%D0%BF%D0%BE%D1%81%D0%BE%D0%B1%D1%8B-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B-api)  
   2.4. [Сетевые протоколы](https://edu.21-school.ru/project/62669/task#%D1%81%D0%B5%D1%82%D0%B5%D0%B2%D1%8B%D0%B5-%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB%D1%8B)  
   2.5. [Задание №1](https://edu.21-school.ru/project/62669/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-1-http-%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F)  
   2.6. [Основные методы запросов](https://edu.21-school.ru/project/62669/task#%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5-%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B-%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%B2)  
   2.7. [Задание №2](https://edu.21-school.ru/project/62669/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-2-crud)  
   2.8. [Коды ответов](https://edu.21-school.ru/project/62669/task#%D0%BA%D0%BE%D0%B4%D1%8B-%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D0%BE%D0%B2)  
   2.9. [Задание №3](https://edu.21-school.ru/project/62669/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-3-%D0%BA%D0%BE%D0%B4%D1%8B-%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D0%BE%D0%B2)
3. [Chapter III](https://edu.21-school.ru/project/62669/task#chapter-iii)  
   3.1. [Тестирование API](https://edu.21-school.ru/project/62669/task#%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-api)  
   3.2. [Маршруты и эндпоинты](https://edu.21-school.ru/project/62669/task#%D0%BC%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D1%8B-%D0%B8-%D1%8D%D0%BD%D0%B4%D0%BF%D0%BE%D0%B8%D0%BD%D1%82%D1%8B)  
   3.3. [Задание №4](https://edu.21-school.ru/project/62669/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-4-%D1%8D%D0%BD%D0%B4%D0%BF%D0%BE%D0%B8%D0%BD%D1%82%D1%8B)  
   3.4. [Swagger](https://edu.21-school.ru/project/62669/task" \l "swagger)  
   3.5. [Задание №5](https://edu.21-school.ru/project/62669/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-5-%D0%B7%D0%BD%D0%B0%D0%BA%D0%BE%D0%BC%D1%81%D1%82%D0%B2%D0%BE-%D1%81-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D0%BE%D0%BC-swagger)

**Chapter I**

**Общая инструкция**

Методология Школы 21 может быть не похожа на тот образовательный опыт, который случался с тобой ранее. Её отличает высокий уровень автономии: у тебя есть задача, ты должен её выполнить. По большей части тебе нужно будет самому добывать знания для её решения. Второй важный момент — это peer-to-peer обучение. В образовательном процессе нет менторов и экспертов, перед которыми ты защищаешь свой результат. Ты это делаешь перед такими же учащимися, как и ты сам. У них есть чек-лист, который поможет им качественно выполнить приёмку вашей работы.

Роль Школы 21 заключается в том, чтобы обеспечить через последовательность заданий и оптимальный уровень поддержки такую траекторию обучения, при которой ты не только освоишь hard skills, но и научишься самообучаться.

* Не доверяй слухам и предположениям о том, как должно быть оформлено ваше решение. Этот документ является единственным источником, к которому стоит обращаться по большинству вопросов;
* твоё решение будет оцениваться другими учащимися;
* подлежат оцениванию только те файлы, которые ты выложил в GIT (ветка develop, папка src);
* в твоей папке не должно быть лишних файлов — только те, что были указаны в задании;
* не забывай, что у вас есть доступ к интернету и поисковым системам;
* обсуждение заданий можно вести и в Rocket.Chat;
* будь внимателен к примерам, указанным в этом документе — они могут иметь важные детали, которые не были оговорены другим способом;
* и да пребудет с тобой Сила!

**Chapter II**

**Общее определение**

В прошлом проекте мы упоминали, что клиент взаимодействует с сервером с помощью запросов. Но куда эти запросы отправляются и кем обрабатываются? Для этих задач и придумали **API**.

**API (Application Programming Interface, программный интерфейс приложения)** — это механизмы, которые позволяют двум программным компонентам взаимодействовать друг с другом, используя набор определений и протоколов. Например, система ПО некоторого агентства содержит ежедневно обновляющиеся данные, тогда мобильное приложение этого агентства будет "общаться" с этой системой через API и показывать ежедневные обновления данных на телефоне.

**Что такое API и как он работает**

Акроним API дословно можно расшифровать как программный интерфейс приложения. В контексте API слово "приложение" относится к любому ПО с определённой функцией. Интерфейс можно рассматривать как сервисный контракт между двумя приложениями, который определяет, как они взаимодействуют друг с другом, используя запросы и ответы. А информацию о структуре этих запросов и ответов можно найти в документации API.

Архитектура API обычно объясняется с точки зрения клиента и сервера. Приложение, отправляющее запрос, называется **клиентом**, а приложение, отправляющее ответ, называется **сервером**. Таким образом, в примере выше база данных агентства — это сервер, а мобильное приложение — это клиент.

**Способы работы API**

Существует четыре различных способа работы API в зависимости от того, когда и почему они были созданы.

**SOAP API**

**SOAP** — *Simple Object Access Protocol*, то есть простой протокол доступа к объектам. SOAP подразумевает использование *WSDL (Web Services Description Language)* — язык описания веб-сервисов и доступа к ним, основанный на языке XML. То есть клиент и сервер обмениваются сообщениями посредством XML. Это менее гибкий API, который был популярен в прошлом.

**RPC API**

Такие API называются системой **удалённого вызова процедур**. Клиент выполняет функцию (или процедуру) на сервере, и сервер отправляет результат обратно клиенту.

**Websocket API**

**Websocket API** — это ещё одна современная разработка web API, которая использует объекты JSON для передачи данных. WebSocket API поддерживает двустороннюю связь между клиентскими приложениями и сервером. Сервер может отправлять сообщения обратного вызова подключённым клиентам, что делает его более эффективным, чем REST API.

**REST API**

На сегодняшний день это самые популярные и гибкие API-интерфейсы в Интернете. Клиент отправляет запросы на сервер в виде данных. Сервер использует этот клиентский ввод для запуска внутренних функций и возвращает выходные данные клиенту.

**REST (*Representational State Transfer*) API** — архитектурный стиль взаимодействия сайтов и веб-приложений с сервером. REST определяет набор функций, таких как GET, POST, DELETE и т. д., которые клиенты могут использовать для доступа к данным сервера. Клиенты и серверы обмениваются данными по протоколу HTTP.

Главная особенность REST API заключается в том, что такая передача выполняется без сохранения состояния. Это означает, что серверы не сохраняют клиентские данные между запросами. Важно отметить, что клиентские запросы к серверу в данном случае аналогичны URL-адресам, которые мы вводим в браузере для посещения веб-сайта, а ответ от сервера представляет собой простые данные без типичного графического отображения веб-страницы.

RESTful веб-сервисы, как правило, гораздо проще реализовать, чем, например, веб-сервисы на основе SOAP:

* REST обычно использует JSON, который легче анализировать и обрабатывать;
* В случае SOAP вам необходимо определить свой сервис с использованием WSDL;
* При обработке и анализе сообщений SOAP-XML возникают большие накладные расходы.

**Сетевые протоколы**

REST API, как было сказано выше, являются самыми популярными интерфейсами в Интернете. Они работают на базе протокола HTTP (HTTPS). Однако есть и другие сетевые протоколы.

**HTTP (HyperText Transfer Protocol)** — протокол для передачи информации (гипертекста), на базе которого функционируют все сегодняшние сайты. В его возможности входит процесс запрашивания необходимых данных у виртуально удаленной системы (файлы, веб-страницы и прочее).

**HTTPS (HyperText Transfer Protocol Secure)** — расширение протокола HTTP, поддерживающее шифрование данных, создавался с целью повышения безопасности.

**TCP (Transmission Control Protocol)** — один из основных протоколов передачи данных интернета, который является связующим звеном для установки качественного соединения между двумя устройствами, передачи данных и верификации их получения.

**IP (Internet Protocol)** — протокол, в функции которого входит корректность доставки сообщений по выбранному адресу. При этом информация делится на пакеты, которые могут поставляться по-разному.

**TCP/IP** — совокупность протоколов передачи информации. TCP/IP – это особое обозначение всей сети, которая функционирует на основе протоколов TCP, а также IP.

**UDP (User Datagram Protocol)** — протокол, управляющий передачей данных, но данные не проходят верификацию при получении. Этот протокол функционирует быстрее, чем протокол TCP.

**FTP (File Transfer Protocol)** — протокол передачи *файлов* по сети. FTP-server в таком случае — это компьютер, предназначенный для хранения файлов.

**ICMP (Internet Control Message Protocol)** — протокол, который входит в стек протоколов TCP/IP. В основном ICMP используется для передачи сообщений об ошибках и других исключительных ситуациях.

**POP3, IMAP, SMTP** — это протоколы предназначенные для работы с почтой.

Из всех этих протоколов больше всего нас будут интересовать протоколы HTTP и HTTPS.

**Задание №1. HTTP-сообщения**

Пользуясь дополнительными источниками, попробуй найти информацию про HTTP-сообщения: что это такое, каких типов они бывают и какова их структура. Создай файл exercise1.md, запиши туда всю найденную информацию и озаглавь полученный текст как ## HTTP-сообщения.

Также в этом файле опиши основные HTTP методы, при помощи которых создаются запросы к серверу (не меньше 7), указав, для чего используется каждый из них. Данный раздел текста озаглавь как ## HTTP методы

**Задание №2. CRUD**

Создай файл excercise2.md, в который запиши определение акронима CRUD, а также что он означает в контексте REST API.

**Коды ответов**

Код ответа (статус-код) показывает, успешно ли был выполнен определённый HTTP запрос. Коды можно разделить на 5 классов:

* Информационные: 100 — 199
* Успешные: 200 — 299
* Перенаправления: 300 — 399
* Клиентские ошибки: 400 — 499
* Серверные ошибки: 500 — 599

**Задание №3. Коды ответов**

Перед выполнением этого задания познакомься с основными кодами ответов!

Представь, что ты тестируешь *GET-метод*, который нужен для запроса списка работников банка. В ходе тестирования ты можешь получить один из трёх вариантов ответа сервера:

1. Код ответа: 201. Тело ответа:

[

{

"name": "Малышева Екатерина Матвеевна",

"birthday": "1995-06-01",

"post": "Бухгалтер"

},

{

"name": "Капустин Роман Артёмович",

"birthday": "1991-01-18",

"post": "Финансовый аналитик"

},

{

"name": "Касьянов Ярослав Ярославович",

"birthday": "1989-07-29",

"post": "Кредитный эксперт"

},

{

"name": "Белова Елизавета Руслановна",

"birthday": "1997-04-13",

"post": "Аудитор"

},

{

"name": "Романов Константин Александрович",

"birthday": "2001-12-14",

"post": "Кассир"

},

...

]

1. Код ответа: 400. Тело ответа:

{

"message": "Неверные данные для авторизации."

}

1. Код ответа: 500. Тело ответа:

{

"message": "Неверно составлен запрос."

}

Как ты думаешь, все ли ответы имеют верный статус-код? Если нет, то какой код должен быть в этих ответах? Запиши это в файл exercise3.md и озаглавь полученный раздел как ## Коды ответов.

**Chapter III**

**Тестирование API**

**API Testing** — это тип тестирования программного обеспечения, во время проведения которого проверяются интерфейсы прикладного программирования (API). Целью API-тестирования является проверка функциональности, надёжности, производительности и безопасности интерфейсов программирования. В тестировании API вместо использования стандартных пользовательских входов (клавиатуры) и выходных данных используется программное обеспечение, предназначенное для отправки вызовов в API, получения выходных данных и записи ответа системы. Тесты API в отличие от GUI тестирования не концентрируются на внешнем виде приложения.

Пирамида тестов Майка Кона, которая представлена ниже, помещает тесты API на сервисный уровень (интеграционный). Это значит, что около 20% (или более) всех тестов должны быть сосредоточены на уровне API (точный процент зависит от наших потребностей).

При наличии модульных тестов, покрывающих весь код, API тестирование позволяет проверить надёжность взаимодействия между клиентом и сервером.

**Маршруты и эндпоинты**

Когда мы говорим про API, часто можно встретить такие понятия как маршрут и эндпоинт. Рассмотрим их подробнее:

* **Маршрут (route)** — это "имя", которое отсылает работу API к определённым эндпоинтам. Можно сказать, что маршрут — это URL, к которому можно обратиться разными HTTP методами. Таким образом, маршрут может иметь несколько эндпоинтов.
* **Эндпоинт (endpoint)** — это само обращение к маршруту отдельным HTTP методом. Эндпоинты выполняют конкретную задачу, принимают параметры и возвращают данные клиенту.

**Задание №4. Эндпоинты**

А теперь представь, что тебе нужно придумать 4 эндпоинта для приложения, которое позволяет работать с рецептами новогодних блюд (просматривать, добавлять и прочее). Использовать нужно все методы CRUD. Какие это будут эндпоинты? Запиши их в файл exercise4.md (в нём уже есть несколько примеров 😉).

**Swagger**

Перед тестированием API нужно знать, какие методы реализованы на сервере, как и по какому URL'у к ним можно обратиться. Иными словами, нам нужна документация. **Swagger** — это набор инструментов, который позволяет автоматически описывать API на основе его кода или набора правил в формате JSON-файла.

**OpenAPI** — спецификация, по которой работает Swagger, но иногда название OpenAPI применяют при описании продукта.

**Swagger UI** — это инструмент, который визуализирует документацию, представляет её в более простом для понимания виде. Более того, она становится не просто визуальной, но и интерактивной, то есть с ней можно взаимодействовать - без написания кода, просто с помощью интерфейса.

**Задание №5. Знакомство с функционалом Swagger**

Перейди в [Swagger тестового API](https://fakerestapi.azurewebsites.net/index.html). Это Fake REST API, который был разработан специально для ознакомления со Swagger UI. В файле exercise5.md опиши все возможные запросы (27 эндпоинтов). Каждое описание эндпоинта должно содержать:

1. HTTP-метод;
2. Полный URL запроса;
3. Заголовки запроса;
4. Тело запроса (при наличии);
5. Статус-код ответа;
6. Тело ответа (при наличии). Если тело большое, то только его часть;

Для GET-запроса с указанием {id} в запросе, а также для каждого PUT- или POST- запроса требуется описать два случая:

* успешный, когда сервер обрабатывает полученные данные *без ошибки*
* провальный, когда сервер обрабатывает полученные данные *с ошибками* (к примеру, возращает 400 ошибку)

*P.S.: итого нужно описать 44 отправленных запроса* 🤫.

**Double-check**

Перед загрузкой выполненного проекта в репозиторий перепроверь наличие всех необходимых файлов, которые требовалось создать во время его выполнения:

exercise1.md

exercise2.md

exercise3.md

exercise4.md

exercise5.md

💡 [Нажми здесь](https://forms.gle/37mimBmMrATvwHMb8) **чтобы отправить обратную связь по проекту**.