**Проект №5 — Front-end тестирование**

Привет, участник Школы 21! В этом проекте мы наконец познакомимся с такими понятиями, как Front-end и Back-end, разберёмся с тем, как устроены современные веб-приложения и как их необходимо тестировать. 😇

**Instructions**

Новая команда и новый групповой проект! Распределяйте задачи в команде так, чтобы добиться максимальной эффективности. Тимлида также должен отслеживать процесс выполнения заданий.

Напоминаем, что все созданные отчёты и файлы вам нужно будет загрузить в папку src/ в корне проекта (обязательно в ветку *develop*).

**Contents**

1. [Chapter I](https://edu.21-school.ru/project/62668/task#chapter-i)  
   1.1. [Общая инструкция](https://edu.21-school.ru/project/62668/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/62668/task#chapter-ii)  
   2.1. [Веб-приложения](https://edu.21-school.ru/project/62668/task#%D0%B2%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F)  
   2.2. [Клиент-серверное взаимодействие](https://edu.21-school.ru/project/62668/task#%D0%BA%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%BE%D0%B5-%D0%B2%D0%B7%D0%B0%D0%B8%D0%BC%D0%BE%D0%B4%D0%B5%D0%B9%D1%81%D1%82%D0%B2%D0%B8%D0%B5)  
   2.3. [Front-end и Back-end](https://edu.21-school.ru/project/62668/task" \l "front-end-%D0%B8-back-end)  
   2.4. [Задание №1](https://edu.21-school.ru/project/62668/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-1-mpa-spa-pwa)  
   2.5. [Задание №2](https://edu.21-school.ru/project/62668/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-2-%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0)
3. [Chapter III](https://edu.21-school.ru/project/62668/task#chapter-iii)  
   3.1. [Основные Frontend технологии](https://edu.21-school.ru/project/62668/task#%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D0%B5-frontend-%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D0%B8)  
   3.2. [HTML](https://edu.21-school.ru/project/62668/task#html)  
   3.3. [JavaScript](https://edu.21-school.ru/project/62668/task" \l "javascript)  
   3.4. [XPath](https://edu.21-school.ru/project/62668/task" \l "xpath)  
   3.5. [Задание №3](https://edu.21-school.ru/project/62668/task#%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-3-xpath)
4. [Chapter IV](https://edu.21-school.ru/project/62668/task" \l "chapter-iv)  
   4.1. [UX/UI](https://edu.21-school.ru/project/62668/task" \l "uxui)  
   4.2. [Задание №4](https://edu.21-school.ru/project/62668/task" \l "%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-4-ui-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)

**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 таким же учащимися, как и ты сам. У них есть чек-лист, который поможет им качественно выполнить приемку 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;
* твое 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**

**Веб-приложения**

Несколько раз в этом курсе упоминалось слово "веб-приложение", но чем же веб-приложения отличаются от обычных сайтов? Какие бывают веб-приложения? Давайте разбираться!

Сначала разберёмся в том, что такое сайт. **Сайт**, или **веб-сайт**, — это одна или несколько логически связанных между собой веб-страниц. Сайты могут содержать контент разного формата: текст, картинки, музыка и прочее. Зачастую сайты не так связаны с сервером, как веб-приложения, они не предоставляют излишней интерактивности, их задача — выводить блоки текста с различными медиа-файлами.

В свою очередь, веб-приложения — это клиент-серверные приложения, доступ к которым осуществляется при помощи браузера. Они позволяют пользователям вводить, получать и манипулировать данными. Грубо говоря, веб-приложения — это сайт, в который добавили различные кнопки, поля и формы для ввода данных.

Основные отличия веб-сайта от веб-приложения:

* Веб-сайт является просто источником информации, в то время как веб-приложение работает в интерактивном режиме;
* Пользовательский интерфейс веб-приложения намного сложнее, чем интерфейс веб-сайта;
* Веб-приложение может быть составной частью сайта либо отдельным ресурсом;
* Разработка веб-сайта требует гораздо меньше времени, чем разработка веб-приложения.

**Клиент-серверное взаимодействие**

Вы, пользуясь веб-приложением, постоянно взаимодействуете с сервером. Допустим, регистрация в какой-нибудь социальной сети непременно приведёт к появлению ваших данных на сервере. Так и любое приложение, размещённое в сети Интернет, основывается на связке "клиент-сервер". И это не только электронные формы. Даже простое "перелистывание" страниц некоторого сайта в Интернете — пример клиент-серверного взаимодействия, ведь страницы хранятся не на каждом компьютере, а подгружаются извне.

Что же понимается под "клиентом" и "сервером"? 🤔

**Клиент** — в нашем случае это компьютер, оснащённый специальным программным обеспечением, которое позволяет пользователю отправить запрос к другой машине и получить ответ. Код, выполняемый на стороне клиента, чаще всего называют клиентским кодом. Он обеспечивает создание пользовательского интерфейса. Здесь, когда речь заходит о браузере, важную роль играет язык программирования JavaScript и библиотеки расширения.

**Сервер** — это компьютер, который оснащён специальным программным обеспечением, которое позволяет решить задачи предоставления пользователю доступа к некоторым услугам и ресурсам, которыми владеет и управляет данный сервер. Код, выполняемый на стороне сервера, называют серверным кодом или серверным сценарием. Он обеспечивает обработку данных. Здесь важную роль играют серверные языки программирования. Примерами таких языков являются PHP и Python.

Таким образом, базовый концепт заключается в том, что клиент посылает запрос на сервер, а сервер ему отвечает (с запросам и протоколами взаимодействия мы обязательно познакомимся в следующем проекте😉).

Из преимуществ такого подхода можно отметить:

* Низкие требования к устройствам клиентов
* Гибкая архитектура

Но также существуют и недостатки:

* Высокая стоимость оборудования
* Сложное обслуживание

Обычно все говорят о двух "сторонах" веб-приложения, однако, в алгоритме подобного взаимодействия довольно активно принимает участие еще одна "сторона" — **база данных**. Вся динамическая информация приложения (учетные, пользовательские данные и пр) хранится именно здесь. Базам данных посвещён целый проект в нашем курсе, потому позже мы их обязательно обсудим!

**Front-end и Back-end**

Вот мы и дошли до этих терминов. В IT сфере их можно услышать почти везде, где идёт речь о разработке веб-приложений. Так что же мы понимаем под словами Front-end и Back-end? На самом деле, всё очень просто.

**Front-end (frontend, фронтенд)** — это клиентская часть веб-приложения, пользовательский интерфейс и связанные с ним компоненты. То есть фронтенд — всё, что браузер может читать, выводить на экран и/или запускать.

**Back-end (backend, бэкенд)** — это внутренняя, скрытая от пользователя часть веб-приложения. То есть бэкенд — всё, что работает на сервере, "не в браузере".

**Задание №1. MPA, SPA, PWA**

MPA, SPA, PWA... Кажется, мы совсем запутались в этих аббревиатурах... Сможете нам помочь? У нас возникло много-много вопросов 😔.

Создайте файл exercise1.md, в котором:

* расшифруйте три аббревиатуры: MPA, SPA и PWA, дайте этим терминам определения и опишите их различия, недостатки и преимущества;
* приведите по 3 различных примера к каждому термину (с ссылками на реальные приложения);
* ответьте на вопросы:
  + Зачем вообще нужны PWA? Почему нельзя ограничиться SPA или MPA?
  + Какие технологии используются для реализации PWA?
  + Как можно понять (и можно ли), что перед нами PWA, а не другие виды приложений?
  + Чем отличается между собой тестирование SPA и MPA?
  + Как тестировать PWA?

**Задание №2. Архитектура**

Создайте файл exercise2.md, в котором дайте ответы на следующие вопросы:

* Какие существуют уровни веб-приложения?
* За что отвечает каждый из них?
* Что такое монолитная и микросервисная архитектуры веб-приложения?
* Каковы различия между монолитом и микросервисами?
* Почему не все приложения построены на микросервисной архитектуре?
* Каковы особенности тестирования монолитных и микросервисных веб-приложений?

**Chapter III**

**Основные Frontend технологии**

Для разработки фронтенд части приложения нужны разнообразные инструменты, которые помогут создать структуру, красиво оформить продукт и сделать его интерактивным. Среди подобных инструментов нужно отметить:

* HTML
* CSS
* JS (JavaScript)

**HTML**

**<strong>HTML (HyperText Markup Language)</strong> — это язык гипертекстовой разметки текста. HTML-код используется для структурирования и отображения веб-страницы и её контента. Чтобы посмотреть на HTML-код любой страницы в Интернете, просто нажмите на клавишу**и перейдёте во вкладку F12. Также можно разом посмотреть код всей страницы: для этого надо нажать правой кнопкой мыши на пустую область сайта, а после на кнопку "Посмотреть код страницы".Elements

HTML-код состоит из ряда элементов. Когда вы заходите на сайт, браузер подгружает HTML-файл с информацией о структуре и контенте веб-страницы. HTML как бы выстраивает визуальный фундамент сайта, он указывает, где будут располагаться элементы, какой у них будет базовый дизайн, откуда брать стили для элементов и скрипты (обычно их пишут на JavaScript). Элементы же состоят из тегов и контента.

Конечно, тема HTML очень обширная, её попросту нельзя раскрыть за один проект. Существуют целые курсы по изучению HTML и вёрстки сайтов. Поэтому подробно рассматривать данную тему мы не станем, но изучим основные теги на примере построения XPath'ов.

**CSS**

**CSS (Cascading Style Sheets, каскадные таблицы стилей)** — это код, который используется для стилизации веб-страницы. Он позволяет "украсить" ваши HTML-элементы, по умолчанию у которых почти никаких стилей нет. CSS позволяет реализовать анимации, переходы, реакции элементов на движение мыши и прочее.

**JavaScript**

А вот **JavaScript** — это полноценный язык программирования, который можно встроить в веб-страницу. Этот язык позволяет реализовывать сложные вещи в веб-приложении, например, отображать периодически обновляемый контент, интерактивные графики и так далее.

*P.S.: очень советуем изучить вышеперечисленные технологии подробнее, ведь вам ещё не раз придётся с этим столкнуться!*

**XPath**

**XPath (XML Path Language)** — это язык запросов к элементам XML-документа. Представьте, что в документе Word вы сочетанием клавиш Ctrl + F вызываете функцию поиска контента. Вписывая что-то в поле ввода, вы можете осуществить поиск элементов в вашем документе. То же самое можно делать и в коде HTML, только тут функционал поиска расширяется как раз за счёт XPath'ов, при помощи которых вы можете либо точно указать позицию элемента, либо найти элементы, относящиеся к конкретному классу и многое-многое другое.

Самый простой способ получить XPath — это во вкладке Elements нажать правой кнопкой на элемент и нажать на "Copy full XPath", тем самым скопировав *абсолютный* XPath. Выглядеть он будет примерно следующим образом: /html/body/div[11]/div/div. Означает это то, что наш элемент (конечный блок div) располагается в другом блоке, который располагается 11-ым по счёту блоком в третьем блоке div, который, в свою очередь, располагается в body, который находится в главном теге html. Но в подобных абсолютных указателях есть два огромных недостатка:

1. Это совершенно неудобно читать и не сразу понятно, о каком элементе идёт речь.
2. Если HTML-код данной страницы хотя бы немного поменяется, например, добавится новый блок и тем самым сдвинет наш элемент выше или ниже, то такой XPath будет указывать уже на совсем другой элемент.

Для *относительного* XPath путь начинается с середины структуры HTML DOM. Он начинается с двойной косой черты (//), что означает, что он может искать элемент в любом месте на веб-странице. Далее используется имя тега нужного нам узла, указывается атрибут узла и значение этого атрибута: //tagname[@Attribute='value']. Например, представим, что мы пишем XPath до поля ввода Email адреса в простом веб-приложении, тогда он может быть написан двумя способами:

* /html/body/div/div/form/table/tbody/tr/td/input
* //input[@id='email']

Второй XPath выглядит гораздо лучше первого, не так ли?

**Задание №3. XML, HTML и XPath**

Создайте файл exercise3.1.md, в него добавьте ответы на вопросы:

* Что такое язык разметки?
* Какие вы знаете языки разметки? Опишите каждый из них.
* Что такое XML?
* Что такое теги, атрибуты и элементы в XML?
* Чем XML отличается от HTML?
* Что такое DTD? Расшифруйте и приведите примеры использования в различных документах.
* Из чего состоит HTML-документ?

А теперь перейдите на сайт [СберСтрахования](https://online.sber.insure/store/propertyins/). Ознакомьтесь с основным функционалом приложения. Создайте файл exercise3.2.md, в котором опишите XPath'ы на первых двух страницах ("1. Выбор полиса" и "2. Оформление") для:

* Каждой кнопки;
* Каждого поля для ввода текста;
* Каждого чекбокса;
* Каждого датапикера;
* Логотипа "СБЕР СТРАХОВАНИЕ";
* Слайдера в блоке выбора суммы;
* Хедера "Что будет застраховано?";
* Текстовых блоков: "Мебель, техника и ваши вещи", "Падение летательных аппаратов и их частей";
* Каждой колонки в списка, который находится под хедером "Страховая защита включенная в программу".

*P.S.: XPath'ы нужно писать для каждого элемента отдельно*

Далее при помощи вкладки Elements отредактируйте какой-либо атрибут трех любых элементов на странице (шрифт, цвет, расположение, прочее — на усмотрение команды). Сделайте скрины этих элементов до и после редактирования, присвойте им названия: 1-before.png — до, 1-after.png — после (в итоге получится 6 скринов).

**Chapter IV**

**UX/UI**

Работая с фронтенд-составляющей приложения, можно часто встретиться с такими аббревиатурами, как UX и UI. Есть даже UI и UX виды тестирования.

Аббревиатура UI расшифровывается как user interface — это оформление сайта: сочетания цветов, шрифты, иконки и кнопки.

UX — user experience, то есть "пользовательский опыт". Простыми словами, это то, каким образом пользователь взаимодействует с интерфейсом и насколько приложение для него удобно.

Если функциональное тестирование — это тестирование ПО для проверки реализуемости функциональных требований, то **UI-тестирование** — это тестирование ПО с целью проверки всех визуальных индикаторов и иконок, меню, переключателей, текстовых полей, флажков и так далее.

**UX-тестирование (юзабилити-тестирование)** отличается от других видов тестирования, которые мы рассматривали ранее, тем, что оно подразумевает проведение проверки на реальных пользователях. Потому проводить UX-тестирование в данном проекте мы не будем, но от UI-тестирования не откажемся!

**Задание №4. UI-тестирование**

Ознакомьтесь с технологией проведения UI-тестирования. В TestIT создайте общее рабочее пространство, в нём создайте проект и назовите его "СберСтрахование". Добавьте тест-кейсы, которые позволяют провести полное ручное UI-тестирование двух страниц [СберСтрахования](https://online.sber.insure/store/propertyins/): "1. Выбор полиса" и "2. Оформление".

После выполнения задания выгрузите полученные тест-кейсы в формате xlsx, назовите файл exercise4.1.xlsx и загрузите его в папку /src.

Добавьте в созданное рабочее пространство новый проект. Теперь вам необходимо создать тест-кейсы, которые позволят провести полное ручное UI-тестирование официального сайта [TestIT](https://testit.software/) (проверять нужно только главную страницу и два первых раздела: "О продукте" и "Цена", включая подразделы, но не затрагивая поддомены и другие сайты). Файл с выгруженными тест-кейсами назовите exercise4.2.xlsx и загрузите его во всю ту же папку /src!

**Double-check**

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

exercise1.md

exercise2.md

exercise3.1.md

exercise3.2.md

1-before.png

1-after.png

2-before.png

2-after.png

3-before.png

3-after.png

exercise4.1.xlsx

exercise4.2.xlsx

💡 [Нажми здесь](https://forms.gle/KVQB1xuhVAu3ZU4V7) **чтобы отправить обратную связь по проекту**.