Верно	#	Задача	Решение	Комментарии ученика	Комментарии тренера
Первый урок					
	Elements	Открываем http://arsbatyrov.ru/chrome/elements_dz Перед нами простой сайт заметок. Наша задача - протестировать его:			
		1. Найти место, где текст выйдет за рамки блока, если будет слишком длинным. **Результат:** скриншот с вышедшем за рамки текстом.			
		2. Найти и сделать видимым блок нотификации, проверить, адекватно ли он выглядит. Результат: скриншот видимой нотификации. Составить два CSS-локатора: a. Который бы подошел любой кнопке “Детали задания”. b. Который бы подошел любой кнопке “Удалить”. **Результат:** написать в решении текстом локаторы.			
		3. Задача со звездочкой. Составить универсальный XPath-локатор или CSS-локатор, который бы подошел элементам, содержащим текст ТОЛЬКО важных задач (они помечены лейблом “Важное!”). Т.е. в нашем случае элементам, содержащим тексты “Надо не забыть выполнить домашнее задание.” и “Надо изучить Bash, SQL, Chrome Devtools, ADB, Appium и Selenium, Git.”. **Результат:** текстовый файл с локатором.			
	Console	Открываем http://arsbatyrov.ru/chrome/console_dz Перед нами галерея с котиками, которые ищут свой дом. Наша задача - протестировать ее:			
		1. Найти JS-ошибку, которая не блокирует дальнейший функционал. **Результат:** написать текст ошибки в решении.			
		2. Найти JS-ошибку, которая блокирует функционал закрытия оверлея. **Результат:** написать текст ошибки в решении.			
		3. Нам надо протестировать кнопку “нажмите сюда” на блоке загрузки. Но этот блок пропадает очень быстро. Надо написать JS-код, который изменит поведение JS-функции, которая уничтожает этот блок. Функция называется иначе, чем в видео, а именно - destroyLoader. **Результат:** JS-код, который изменит поведение функции + там же ответить на вопрос - верно ли работает кнопка “нажмите сюда” на блоке загрузки или нет.			
		Задача со звездочкой. Для решение этого задания, возможно, понадобится изучение либо урока про вкладку Elements, либо Network, либо Source. JS умеет не только удалять часть HTML-кода, но и добавлять. Так, например, иногда JS дорисовывает нотификацию, а не делает ее просто видимой. Именно такая нотификация у нас есть на странице. Наша задача - протестировать ее. Проблема в том, что мы не знаем ни при каких условиях она показывается, ни название JS-функции, которая ее показывает. Надо изучить JS-код на странице, понять о какой функции идет речь и вызвать ее. **Результат:** скриншот с показавшейся нотификацией.			
		Открываем http://arsbatyrov.ru/chrome/source_dz Перед нами страница, которая получает картинку со стоков и показывает ее пользователю. Этот сервис нам предстоит протестировать:			
		1. Надо посмотреть, с какого стока (или с каких стоков) мы получаем картинки. **Результат:** написать в решении адресом/адреса.			
		2. Надо перегрузить заголовок страницы, чтобы он был не “<title>Source - Home work</title>”, а “<title>Source - локальная версия</title>”. Убедиться, что перезагрузки страницы появляется именно наша локальная версия. **Результат:** скриншот с заголовком страницы и открытым Chrome DevTools на вкладке Overrides.			

	Source	**3.** Создать snippet со следующим кодом: “function home_work() {$('.a1').removeClass ('a1').addClass('a2');}home_work();”. Запустить его. После, кликая по кнопке получения новой картинки, понять, что изменилось в поведении кнопки. **Результат:** написать текстом в решении ответ на вопрос - что изменилось.			
		4. Задача со звездочкой. Наш сервис устроен так, что нужная картинка всегда есть. Наша же задача - протестировать поведение сервиса в случае отсутствия картинки. Делать это будем путем перегрузки JS-файла, который получает путь до картинки и добавляет ее на страницу. Среди загруженных файлов надо найти “source_dz.js”. Там есть строка *<img class="card-img-top" src="{{path}}" alt="Card image cap">*. Именно вместо {{path}} JS-код подставляет путь до файла. Наша задача поменять эту строку так, чтобы путь до картинки стал неправильным. Далее сохранить изменения в Overrides и перезагрузить страницу. Затем нажать кнопку для получения картинки и посмотреть, как сервис отреагирует. **Результат:** написать в решении текст с измененной строкой.			
	Network	Открываем http://arsbatyrov.ru/chrome/network_dz Перед нами типичный общий чат. Написать сообщение в него может каждый, кто имеет специальную cookie с название chat_token. Наша задача его протестировать:			
		1. Изучить все типы запросов, которые уходят со страницы. Прислать список запросов с описанием того, для чего, по вашему мнению, каждый из них нужен, какой функционал не работал бы без этого запроса. **Результат:** текст с описанием.			
		2. Разработчик сказал, что каждый пользователь в чате будет помечен своим цветом. У нас нет других пользователей в чате, но мы можем имитировать другого пользователя при помощи утилиты CUrl. Надо скопировать из вкладки CUrl-запрос на добавление нового сообщения и изменить его так, словно запрос отправил другой пользователь. Затем надо отправить такой запрос и проверить, действительно ли сообщение отобразится новым цветом. **Результат:** скриншот с чата, где есть сообщения от двух разных пользователей.			
		3. Задача со звездочкой. Найти несколько способов воспользоваться XSS-уязвимостью. Описать каждый из них. **Результат:** описать в решении с описаниями.			
		Второй урок			
	Performance	Открываем http://arsbatyrov.ru/chrome/performance_dz Перед нами такое же JS-приложение, что мы видели на видео. Только кнопки перемешаны. Наша задача - при помощи вкладки Performance понять, какую нагрузку создает каждая из кнопок.			
		1. Надо для каждой кнопки указать тип нагрузки, которые превалирует: “рендеринг и отрисовка”, “скриптинг (долгая работа JS, например, для подсчетов чего-либо)”, “Idle (холостая работа в процессе ожидания)”. **Результат:** описать кнопку и типы нагрузки, которая она создает.			
		2. Надо скопировать профиль одной из кнопок - той, которая генерит больше всего нагрузки (определить в предыдущем задании). **Результат:** прислать файл-профиль.			
	Application	Открываем http://arsbatyrov.ru/chrome/application_dz Мы видим некий сайт, работа которого завязана на данные из Storage и Cookie.			
		1. Cookie могут выставляться как за счет JS, так и за счет сервера. В первом случае JS сам во время работы дает команду на выставление cookie. Во втором обязательно должен быть запрос к серверу, а в запросе в поле Response Headers должен быть заголовок Set-Cookie со значением cookie, который надо выставить. Наша задача - определить, какая кнопка выставляет cookie за счет ответа сервера, а какая - за счет работы самого JS. **Результат:** написать ответ, какая кнопка каким способом выставляет Cookie.			
		2. Есть кнопка, которая сохраняет введенные нами текст. Но мы не знаем куда, в cookie, local session или session storage. Наша задача - это понять, используя кладку Application. **Результат:** написать ответ на вопрос, куда сохраняется текст (в cookie, local session или session storage).			

		3. Описать, как бы вы действовали для решения предыдущей задачи при условии, что Application-вкладки не было бы. Возможно было бы определить, где именно сохраняется информация? Результат: написать ответ.			
		4. Задача со звездочкой. Даже на такой простой вкладке все равно затесалась XSS-уязвимость. Как ее воспроизвести? **Результат:** описание уязвимости.			
	Device	Открываем http://arsbatyrov.ru/chrome/device_dz Видим веб-приложение с меню и полями для ввода. Наша задача - протестировать это приложение:			
		1. Проверить отображение в портретном и ландшафтном режимах на девайсе с разрешением 1280 на 960. **Результат:** два скриншота, по одному для каждого из режимов.			
		2. Задача со звездочкой. Подключиться к устройству на Android. Открыть веб-приложение из задания в chrome на устройстве. Результат: скриншот экрана компьютера с открытым инспектором.			
	Security	На этот раз наше задание не связано с сайтом http://arsbatyrov.ru			
		1. Найти сайт, который защищен SSL-сертификатом. **Результат:** скриншот деталей его сертификата, который показывает Chrome DevTools.			
		2. Найти сайт, который не защищен SSL-сертификатом. **Результат:** скриншот предупреждения от Chrome о том, что сайту не стоит доверять свои данные.			
	Network Conditions	Открываем http://arsbatyrov.ru/chrome/networkcond_dz Мы видим сайт, работа которого зависит от параметра User Agent. Наша задача - протестировать его:			
		1. Надо найти как минимум один User Agent, при котором неверно определяется браузер или мобильная ОС. **Результат:** написать значение User Agent.			
		2. Надо найти как минимум один User Agent, при котором появляется JS-ошибка. **Результат:** написать значение User Agent.			
		3. Задача со звездочкой. И снова, куда же без полюбившейся нам XSS-уязвимости? Как ее воспроизвести? Результат: описание уязвимости.			