Лекция. Тестирование веб-приложений

# Веб приложение.структура веб-приложения.

**Веб приложение** — [клиент-серверное](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_%E2%80%94_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80) приложение, в котором [клиент](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) взаимодействует с [веб-сервером](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80) при помощи [браузера](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80).

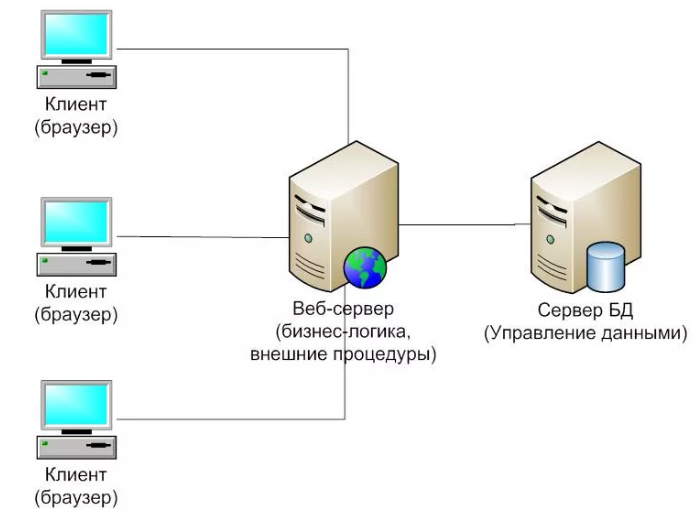
Чем отличается веб-приложение от сайта?

 Сайт это в первую очередь что-то информационное и статичное: визитка компании, сайт рецептов, городской портал или вики. Набор подготовленных заранее HTML-файлов, которые лежат на удаленном сервере и отдаются браузеру по запросу.  
  
Сайты содержат различную статику, которая как и HTML-файл не генерируется на лету. Чаще всего это картинки, CSS-файлы, JS-скрипты, но могут быть и любые другие файлы: mp3, mov, csv, pdf.  
  
А веб-приложение — это технически более сложное приложение. Тут HTML-страницы генерируются на лету в зависимости от запроса пользователя. Почтовые клиенты, соцсети, поисковики, интернет-магазины, онлайн-программы для бизнеса, это все веб-приложения.

Веб-приложение состоит из клиентской и серверной частей, тем самым реализуя [технологию «клиент-сервер»](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80).

Клиентская часть реализует [пользовательский интерфейс](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) (англ. User interface - UI), формирует запросы к серверу и обрабатывает ответы от него. Еще эту клиентскую часть называют Frontend (фронтенд).

Серверная часть получает запрос от клиента, выполняет вычисления, после этого формирует [веб-страницу](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0) и отправляет её клиенту по сети с использованием [протокола](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) [HTTP](https://ru.wikipedia.org/wiki/HTTP). Еще серверную часть называют Backend (бэкенд).



В чем особенности:

Сбой может произойти в любой части приложения. При этом со стороны клиента мы видим только симптомы дефекта, но не ее саму

Так же есть особенности при синхронизации данных, времени. Часто ошибки возникают на стыке разных часовых поясов.

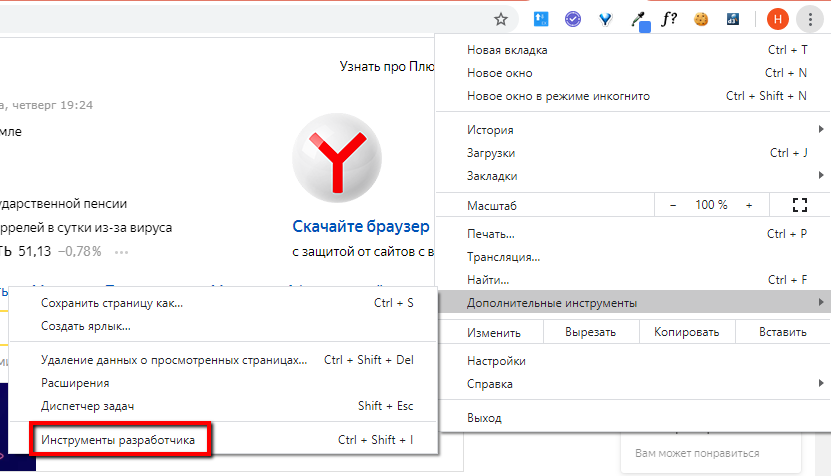
# Взаимодействие клиента и сервера

Взаимодействие происходит по протоколу http – протокол передачи гипертекста.

Самый простой способ увидеть http сообщение использовать инструмент разработчика, который сейчас встроен почти в каждый браузер. Рассмотрим на примере Chrome.

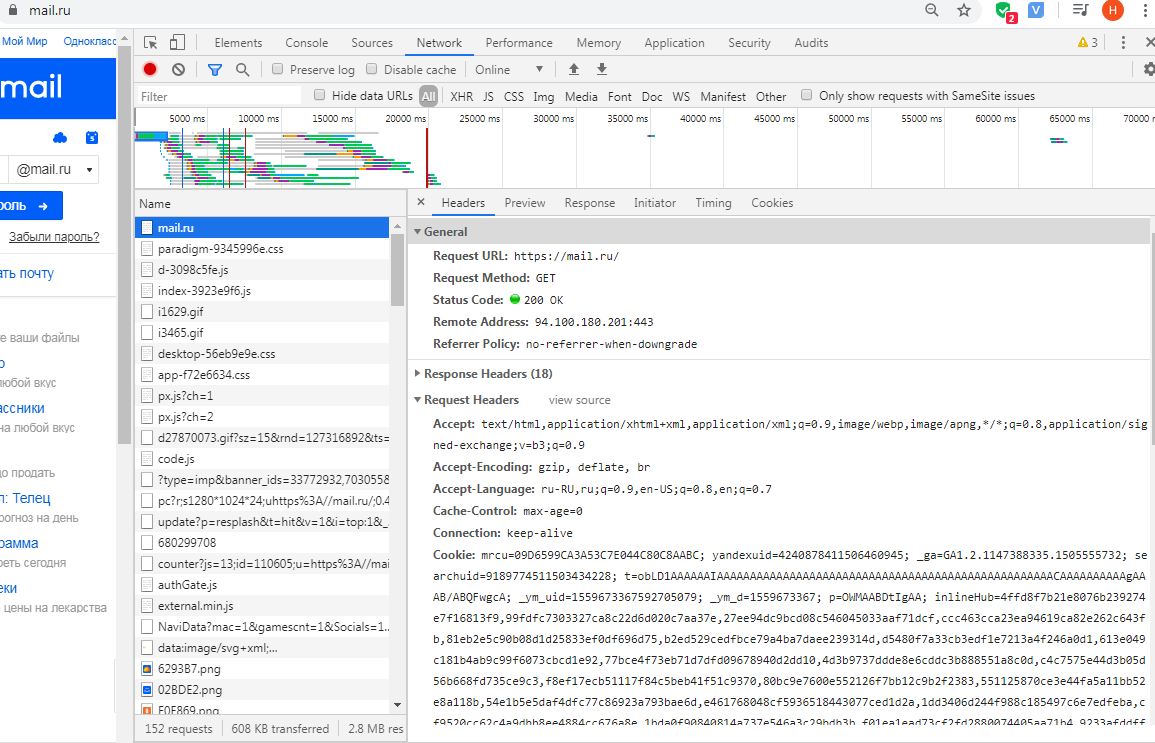
Chrome – один из самых популярных браузеров мира. Не менее популярен и для разработки. Был специально создан пакет утилит для разработки веб-приложений – Chrome Dev Tools. Этот инструмент позволяет работать с веб-приложением на профессиональном уровне, упрощает и помогает при разработке веб страниц. Он имеет обширный функционал, по этой причине его используют и при ручном тестировании, и при автоматизированном.

Инструмент разработчика можно открыть с помощью меню Дополнительные инструменты – инструменты разработчика или нажать на F12 или ctrl+shift+i



Перейдем на mail.ru.

Вкладка Network



## Структура http – сообщения:

GET / HTTP / 1.1

Host: mail.ru

Connection: keep - alive

Cache - Control: max - age = 0

Upgrade - Insecure - Requests: 1

User - Agent: Mozilla / 5.0(Windows NT 6.1; Win64; x64)AppleWebKit / 537.36(KHTML, like Gecko)Chrome / 80.0.3987.122 Safari / 537.36

Sec - Fetch - Dest: document

Accept: text / html, application / xhtml + xml, application / xml;

q = 0.9, image / webp, image / apng, \* /\*;q=0.8,application/signed-exchange;v=b3;q=0.9

Sec-Fetch-Site: none

Sec-Fetch-Mode: navigate

Sec-Fetch-User: ?1

Accept-Encoding: gzip, deflate, br

Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7

Cookie: mrcu=09D6599CA3A53C7E044C80C8AABC; yandexuid=4240878411506460945; \_ga=GA1.2.1147388335.1505555732; searchuid=9189774511503434228; t=obLD1AAAAAAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAAgAAAB/ABQFwgcA; \_ym\_uid=1559673367592705079; \_ym\_d=1559673367; p=OWMAABDtIgAA; inlineHub=4ffd8f7b21e8076b239274e7f16813f9,99fdfc7303327ca8c22d6d020c7aa37e,……….

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

1. Стартовая строка ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Starting line*) — определяет тип сообщения;
2. Заголовки ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Headers*) — характеризуют тело сообщения, параметры передачи и прочие сведения;
3. Тело сообщения ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Message Body*) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Тело сообщения может отсутствовать, но стартовая строка и заголовок являются обязательными элементами.

**Из примера:**

* Стартовая строка (Start line): GET / HTTP / 1.1 , где
  + GET – метод запроса;
  + HTTP / 1.1 – версия протокола.
* Заголовки (Headers):
  + Host: mail.ru – адрес корня сайта - host uri (унифицированный идентификатор ресурса)
  + И др.
* Тело сообщения в данном примере отсутствует

## Методы http-запросов

**GET** – Используется для запроса содержимого указанного ресурса. Параметры передаются в URI. Ответное сообщение содержит ресурс.

**POST** - Применяется для передачи пользовательских данных заданному ресурсу. Например, в блогах посетители обычно могут вводить свои комментарии к записям в HTML-форму, после чего они передаются серверу методом POST и он помещает их на страницу. Параметры запроса передают в URI или в заголовках запроса. Данные передаются в теле запроса. Тело ответного сообщения содержит результат выполнения запроса.

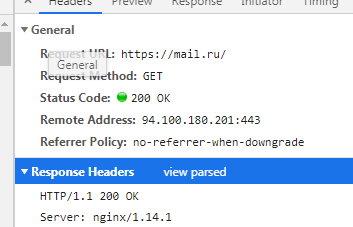
**PUT** - Применяется для загрузки содержимого запроса на указанный в запросе URI.

**DELETE** - Удаляет указанный ресурс.

И другие. Подробнее тут <https://ru.wikipedia.org/wiki/HTTP>.

## Коды статусов

Стартовая строка метода содержит код статуса выполнения запрос



Например:

200 - ОК, Все хорошо

404 – Not found («не найдено»)

503 - Service Unavailable («сервис недоступен»)

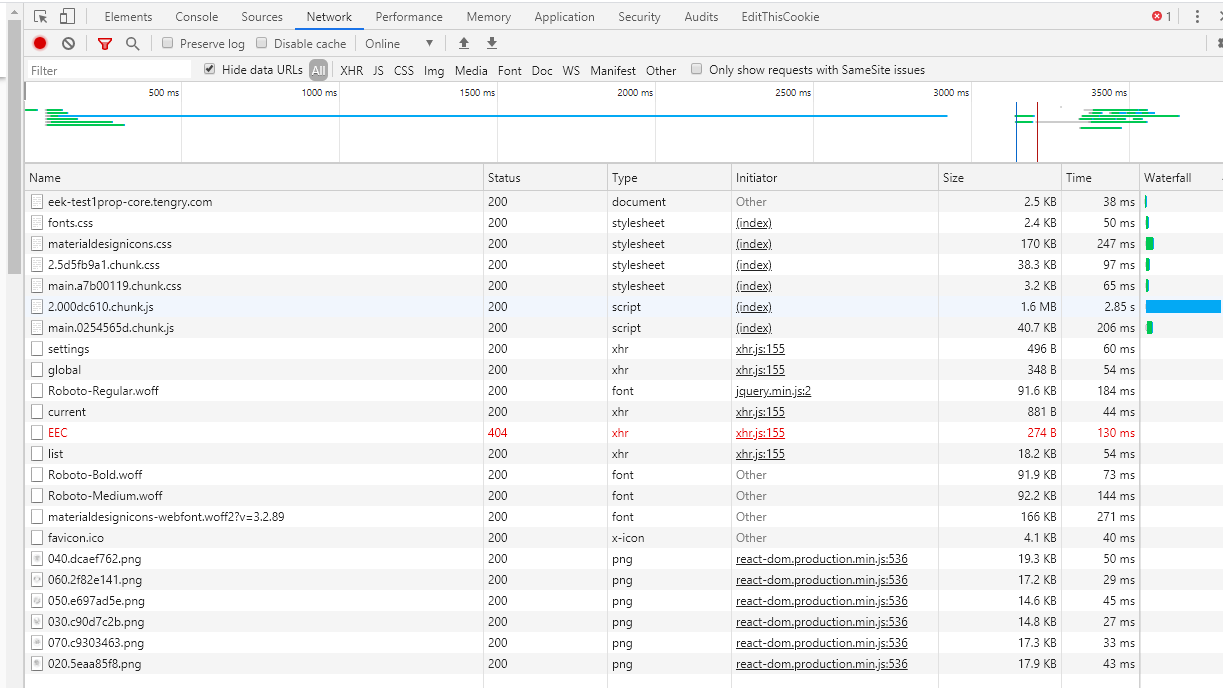
И многие другие. Все статусы указаны здесь <https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%BA%D0%BE%D0%B4%D0%BE%D0%B2_%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D1%8F_HTTP>

Для удобной работы с http запросами существует несколько программ. Например, Postman – его будем рассматривать \ использовать на другой лекции.

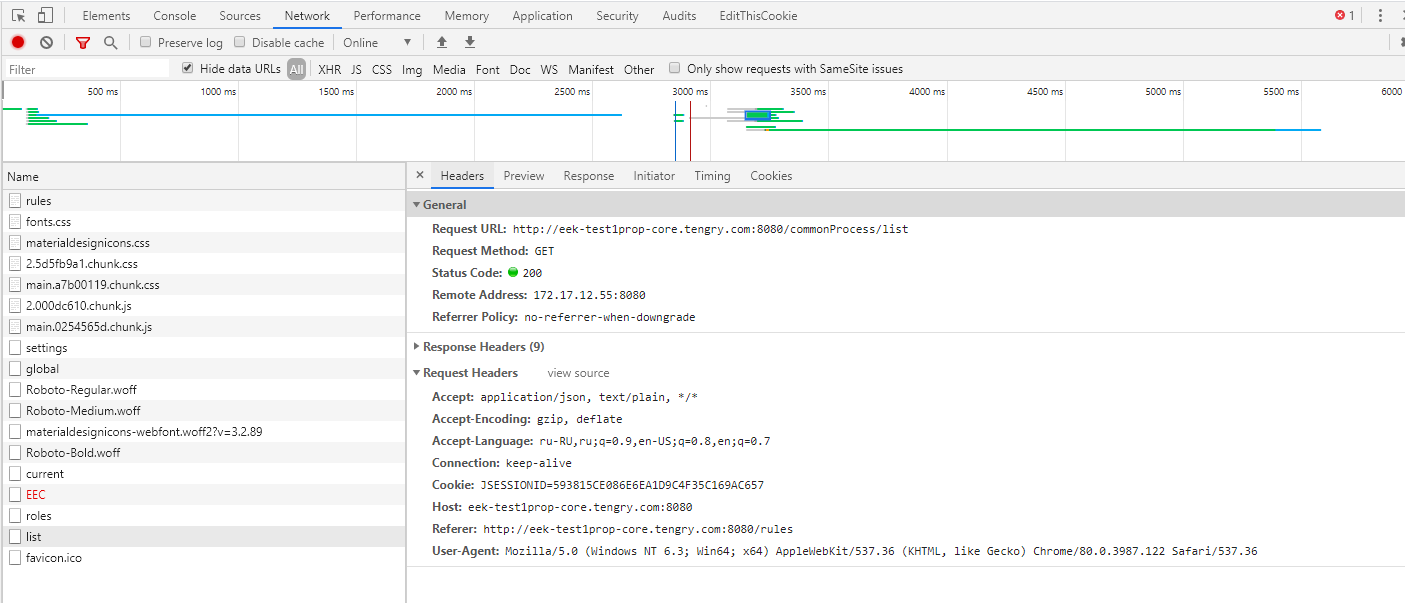
**Набор запросов формирует интерфейс взаимодействия приложения (API) и сторонних программ.**

## Вкладка network

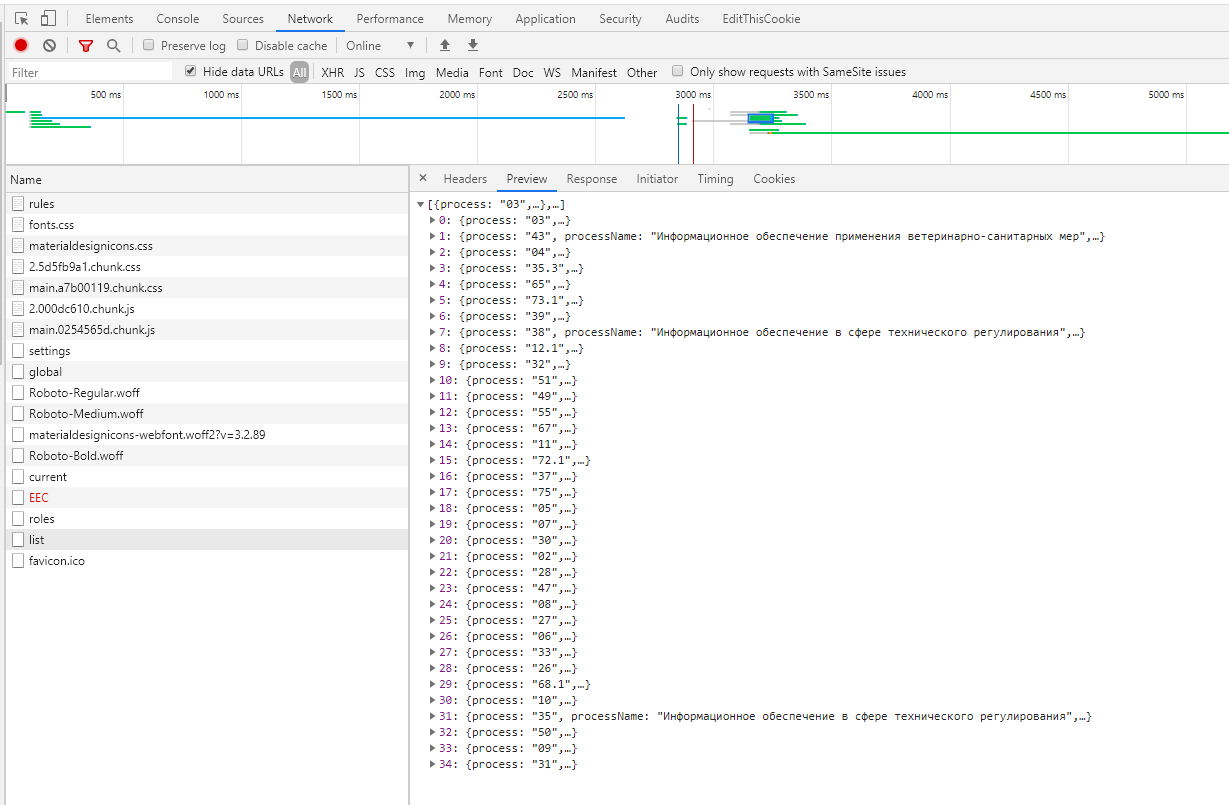
На вкладке netrwork так можем увидеть в списке сразу коды статусов запросов. Запросы, на которые вернулись ошибки помечаются красным



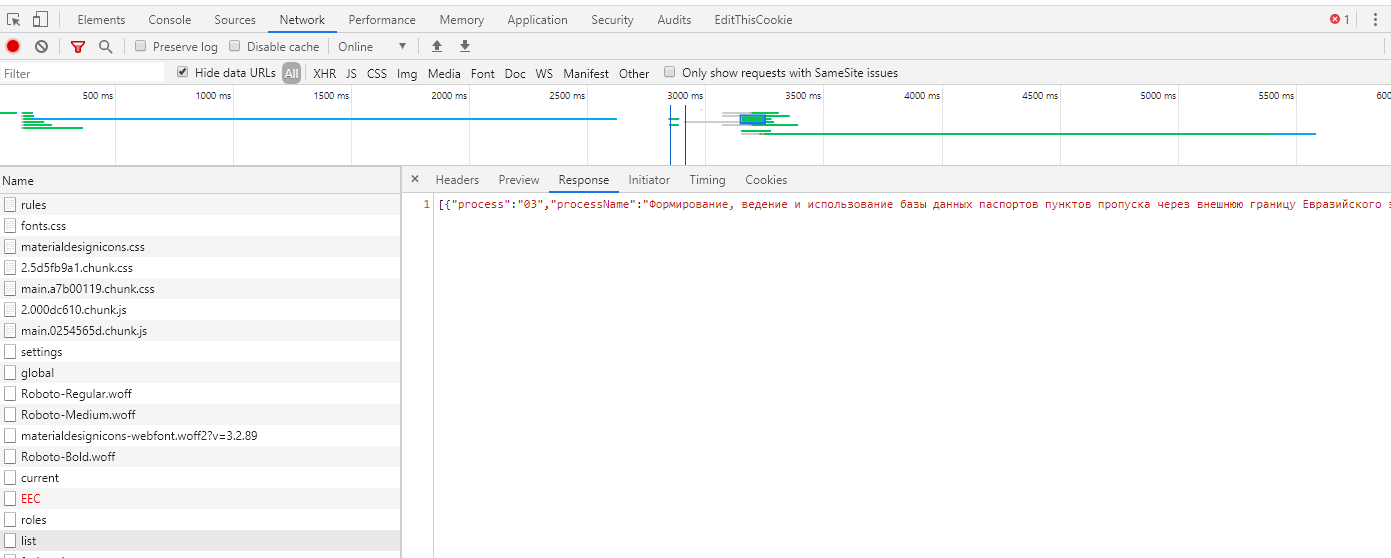
При просмотре записи, на которую возвращается ответ в general видим request с результатом запроса



На вкладке preview видимо превью ответа



На вкладке response- видим сам ответ



## Клиентская часть

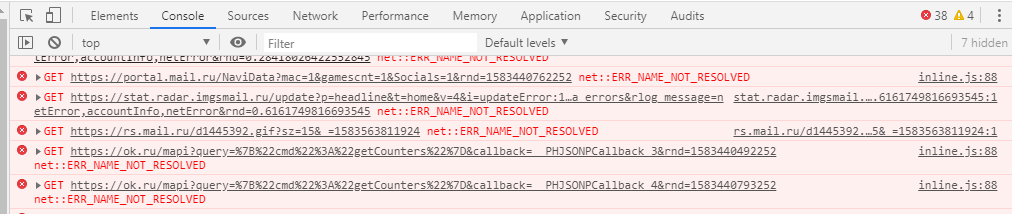
Клиентская часть приложения — это скрипты, написанные на языке программирования Javascript (JS) и исполняемые в браузере пользователя.

Клиент получает ответ от сервера и для того, чтобы показать страницу с данными клиент строит DOM.

DOM (document object model) — это структура HTML-страницы. Работа с DOM — это поиск, добавление, изменение, перемещение и удаление HTML-тегов.

При этом структура может меняться динамически с помощью JS скриптов. JS обеспечивает интерактивность на стороне клиент, например, анимацию, валидацию.

Ошибки JS скрипта можно увидеть в console браузера.



AJAX (asynchronous javascript and XML) - с помощью этой технологии браузер и сервер общаются незаметно для пользователя.

Пример: выход из личного кабинета через 5мин.

Операции на клиенте:

* Отправка запроса
* Получения данных от сервера
* Анализ данных и формирование DOM
* Загрузка ресурсов: изображений,стилей,js
* Выполнение JS кода – изменение DOM,загрузка доп.ресурсов.

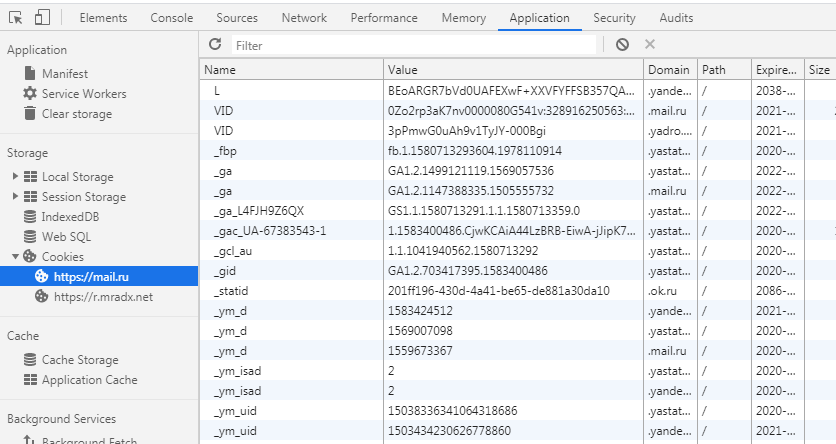
## Что еще необходимо знать для тестирования

### Cookie

Cookie – Небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя.

Обычно в Cookie хранится информация по идентификации компьютера клиента, а так может быть тема оформления, сохраненные настройки и т.д.

Cookie можно просмотреть в инструменте разработчика на вкладке Application



### CASH

Кэширование (Caching) – сохранение данных на стороне сервера или клиента.

На сервере как правило сохраняются редко изменяемые или часто используемые данные. Например, элементы отображения, изображения, некоторые страницы целиком или частично.

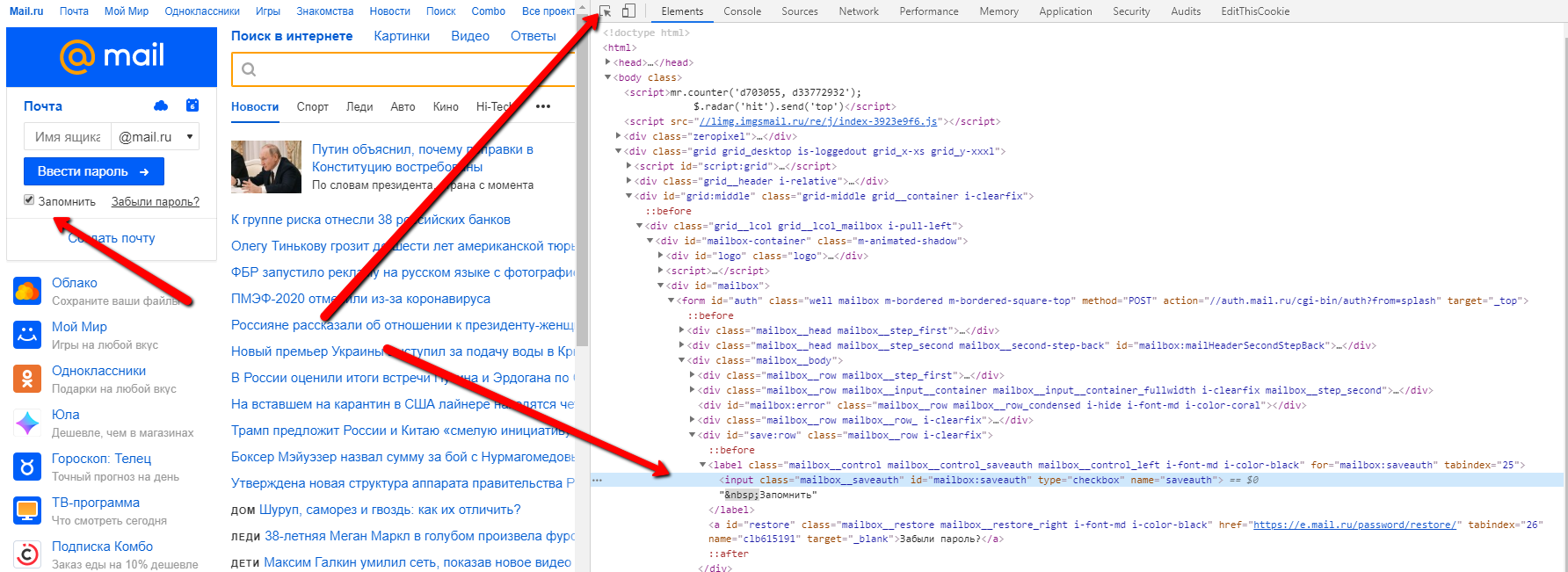
На клиенте кэшируется содержимое веб страницы и js скрипты.

Если разработчик исправил дефект в js скрипте, то для того, чтобы увидеть эти правки обновить страницу недостаточно, необходимо очистить кэш.

### Вкладка Elements

– отображает страницу в html.

При нажатии на Inspect и выборе элемента на странице мы можем просмотреть код этого элемента.



Чаще эту вкладку используют для автоматизации тестирования – поиск локаторов.

### Валидации

На форме могут присутствовать текстовые поля, даты, чек-боксы и т.д.

Валидации могут быть реализованы на фронте, т.е на клиентской части, так и на сервере.

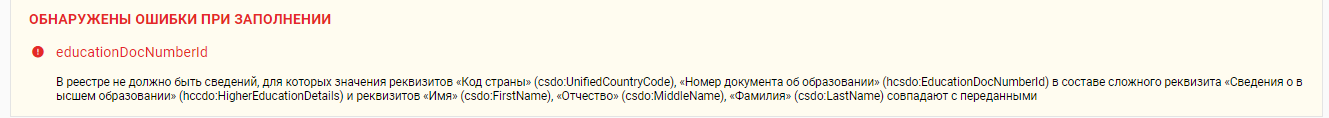
Как правило сервер «не доверяет» клиенты, поэтому на сервере реализуют так же все валидации, которые реализовываются на клиенте.

Валидации на клиентской части:

* Проверки обязательности полей
* Проверки ограничение ввода
* Проверки стандартов



Валидации на сервере происходят после отправки запроса на сервер, помимо того, что сервер выполняет те же проверки, что и клиент, так же проверять может уникальность записи, проверки пароля при входе, проверка необходимых прав.



## Browser

Каким образом разные браузеры влияют на тестирование?

То как страница будет отображаться в браузере зависит от браузерного движка. Движок – программа, преобразующая DOM в изображение на экране. У большинства браузеров свой движок. Поэтому некоторые страницы в разных браузерах отображается по-разному.

**Т.е. при заведении дефектов важно указывать браузер.**

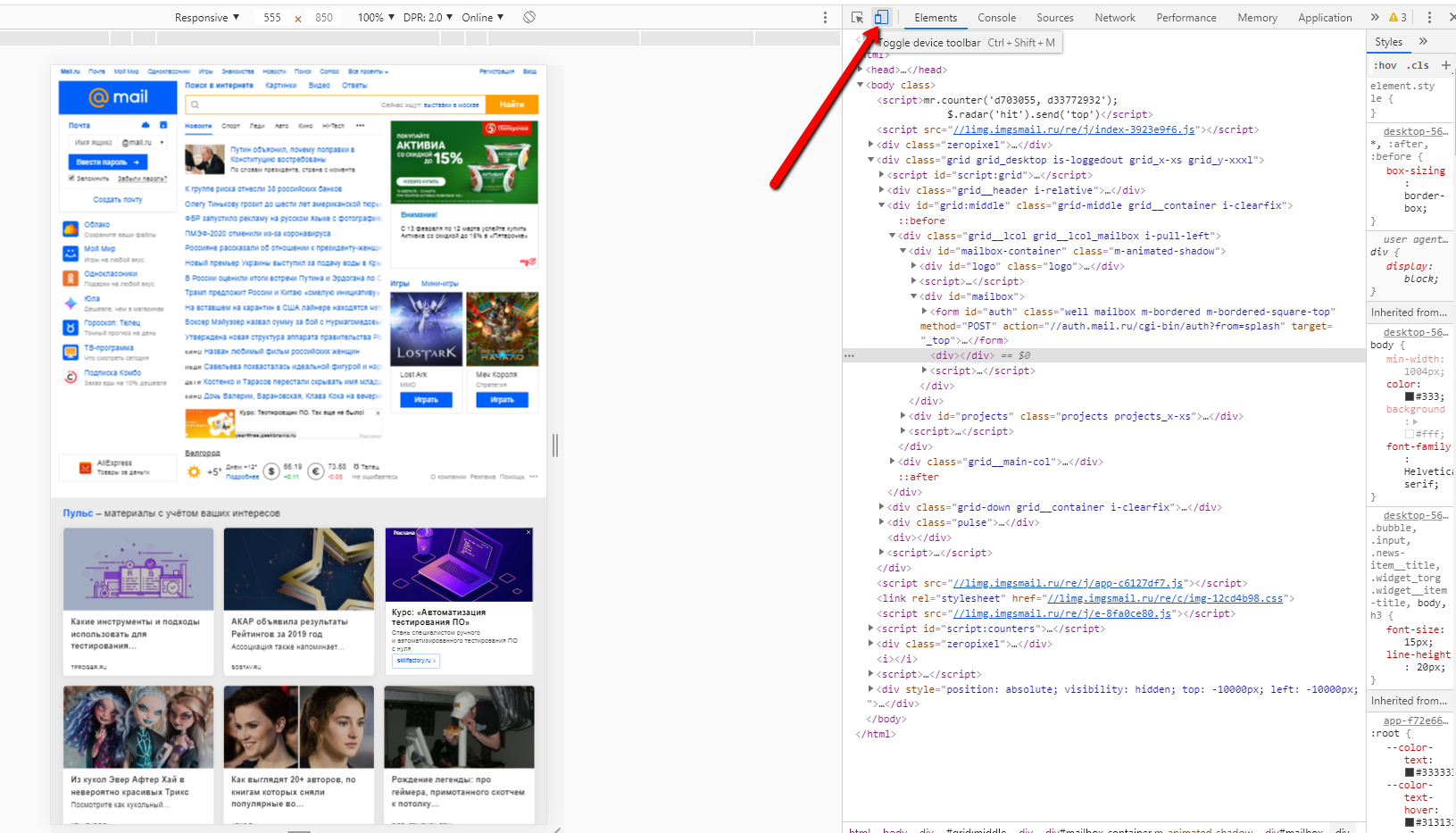
### Макет

Макет страницы подстраивается под разные экраны:

* Размер окна может быть разный
* Разные устройства: планшеты, смартфоны

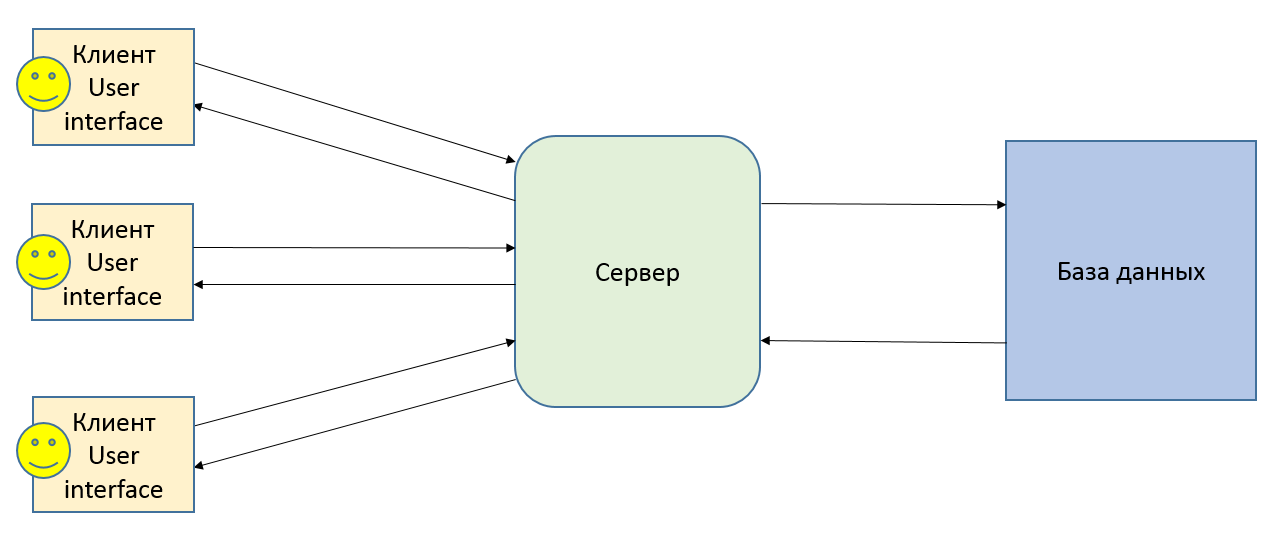
**Т.е. при заведении дефектов важно указывать размер окна.**

В chore devtools справа от Inspect присутствует кнопка переключения на панель инструментов устройства, с помощью которой мы можем проверить отображение страницы на мобильном устройстве.



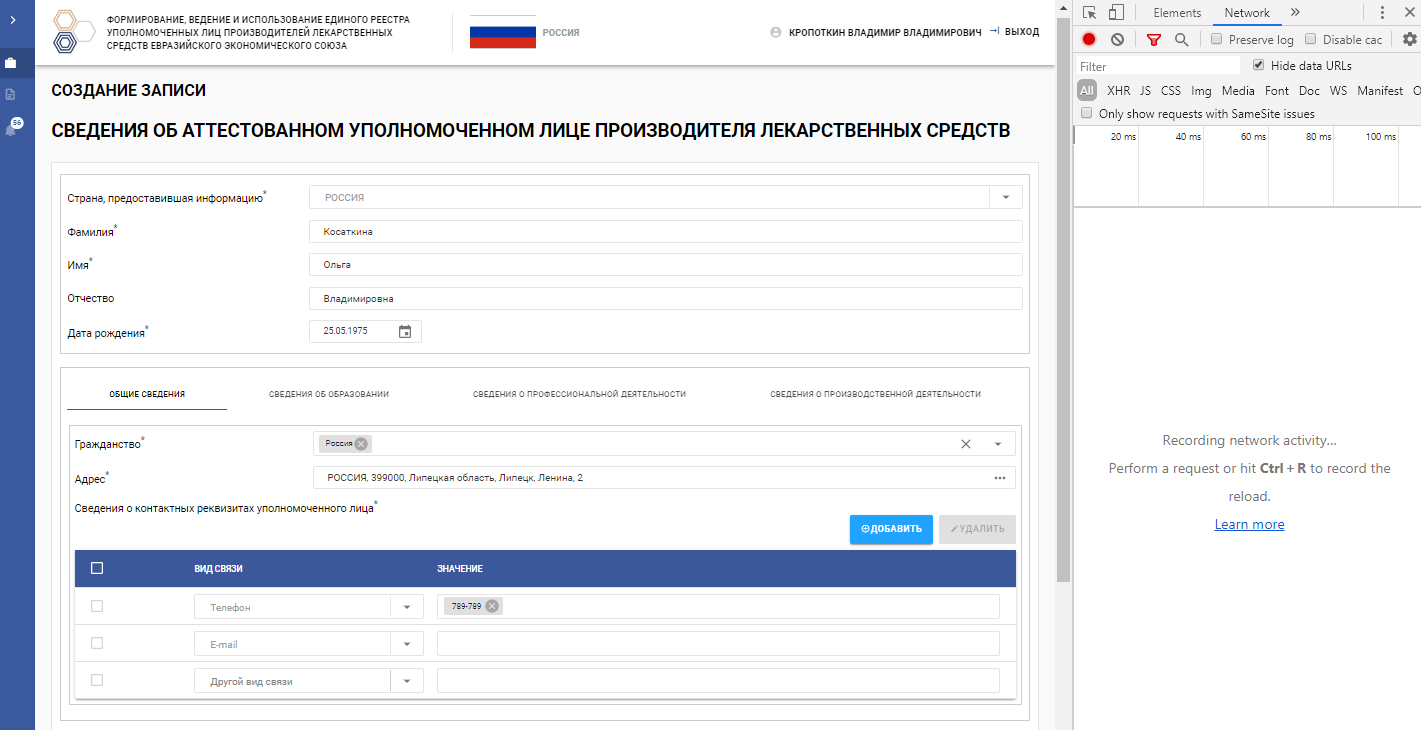
## Взаимодействие клиента, сервера и БД

Рассмотрим пример взаимодействия клиента, сервера и БД веб приложения.



Есть веб-приложение, в рамках которого пользователю предоставляется возможность вести реестр с данными об уполномоченных лицах производителе лекарственных средств. Это значит, что пользовать умеет создавать новую запись в реестре, изменять ее и исключать.

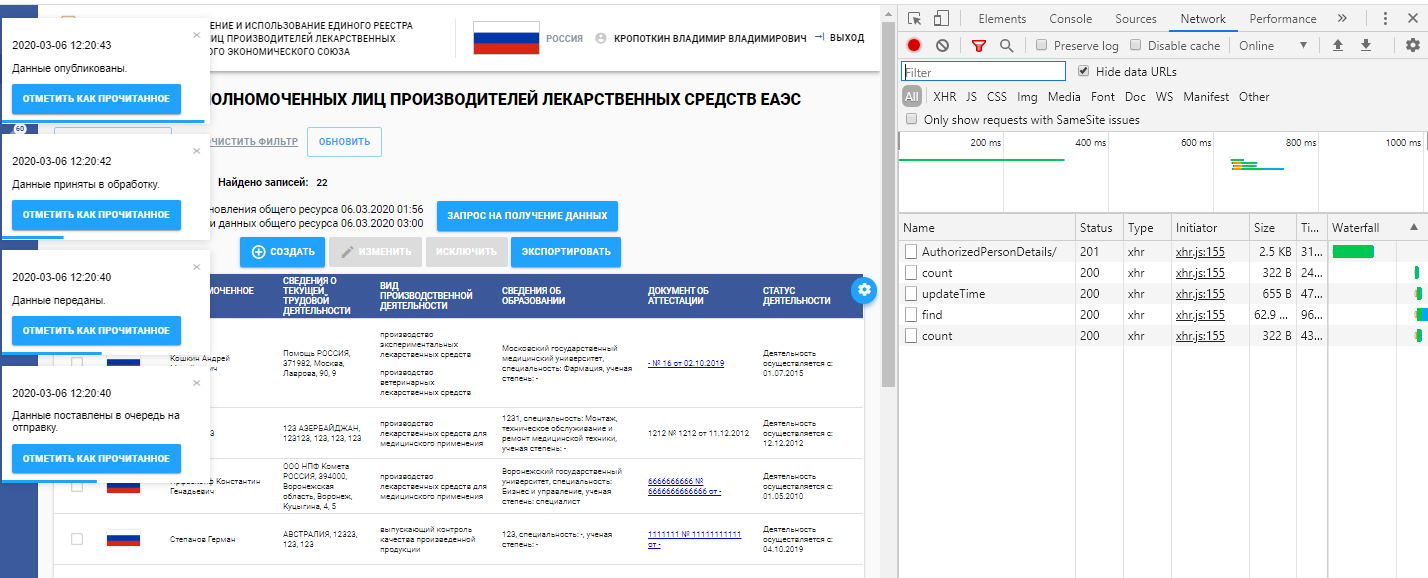
На самом деле функциональность данного приложения намного шире, но мы будем рассматривать функцию создания записи в реестре.

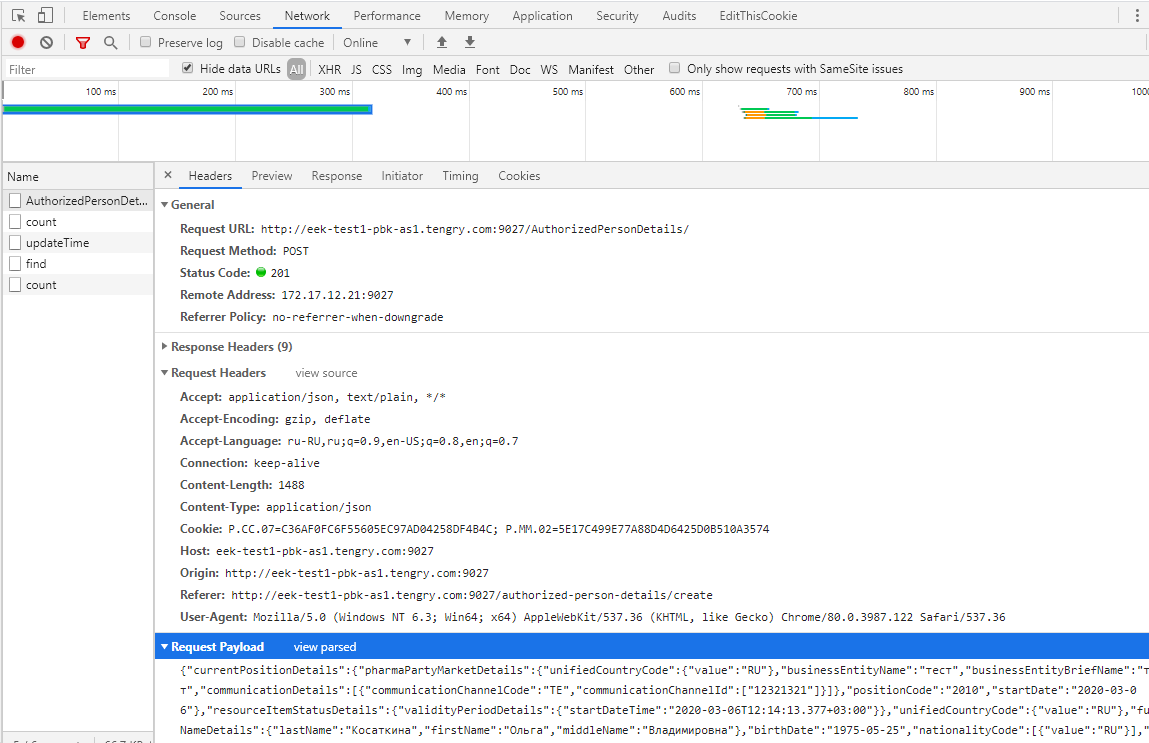


С помощью кнопки «Создать» открываем форму создания записи, заполняем ее данными.

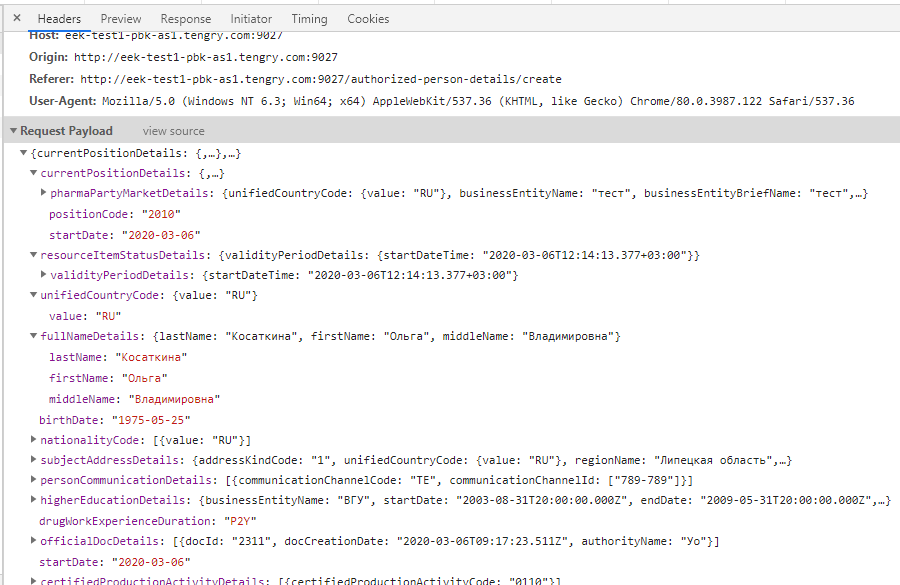
Вкладка network в ChromeDevTools должна быть открыта.

Нажимаем кнопку «Сохранить и отправить» с помощью которой формируется POST запрос на сервер, в теле которого содержится json с ранее введенными на форме данными. Тело запроса на сервере проходит проверки. В нашем примере запрос вернул статус код 201 ОК.





Мы можем сравнить информацию, введенную на форме с телом Post –запроса



После успешных проверок данные сохраняются в БД. В нашем случае используется MongoDB. Просматривать данные в БД будем с помощью инструмента RObo3T.

В Бд мы можем увидеть и проверить как данные были сохранены

