

ТЗ на разработку

1. Создаем функцию `getStorageUsers`, которая запрашивает/создает в `localStorage` свойство `users`, в котором находится массив зарегистрированных юзеров.

1. Если в `localStorage` данного свойства нет, то создаем его и записываем в него исходный массив `USERS`.
2. Функция `getStorageUsers` возвращает массив зарегистрированных юзеров.
3. При заходе на страницу вызываем ее и полученный массив сохраняем в глобальную переменную `const storageUsers = getStorageUsers();`

Для страницы `login.html` пишем логику логина/регистрации пользователя.

2. Login form

1. Достаемся к форме `const LoginForm = document.querySelector('#LoginForm');` и навешиваем обработчик на событие `submit`.
2. Сохраняем введенные юзером значения полей:
 1. `let email = e.target.querySelector('input[data-name=«email»]').value;`
 2. `let password = e.target.querySelector('input[data-name=«password»]').value;`
3. В массиве `storageUsers` ищем пользователя у которого email равен значению с формы.
4. Если юзер с таким email в `localStorage` НЕ найден, то в форме логине достаемся к `div.error`, добавляем ему класс `active` и устанавливаем `innerHTML = 'Invalid email'`.
5. Если юзер с таким email в `localStorage` найден, то далее проверяем, равны ли поля `password`. Если поля НЕ совпадают, то в форме логине достаемся к `div.error`, добавляем ему класс `active` и устанавливаем `innerHTML = 'Invalid password'`.
6. Если юзер с таким email в `localStorage` найден и пароли совпадают, то данному юзеру, которого нашли в `localStorage`, меняем свойство `status` на `true` и в `localStorage` обратно отправляем свойство `users`.
`localStorage.setItem('users', JSON.stringify(storageUsers));` После этого редиректим юзера на страницу его аккаунта. `document.location.href = 'account.html';`
7. Так же, при сабмите формы нужно убирать для `div.error` класс `active`, если он ранее был добавлен.

3. Register form

1. Достаемся к форме `const RegistrationForm = document.querySelector('#RegistrationForm');` и навешиваем обработчик на событие `submit`.
2. Сохраняем введенные юзером значения полей:
 1. `let name = e.target.querySelector('input[data-name=«name»]').value;`
 2. `let email = e.target.querySelector('input[data-name=«email»]').value;`
 3. `let password = e.target.querySelector('input[data-name=«password»]').value;`
 4. `let passwordVerify = e.target.querySelector('input[data-name=«passwordVerify"]]').value;`
3. Если значения полей `password` и `passwordVerify` не равны, то в форме регистрации достаемся к `div.error`, добавляем ему класс `active` и устанавливаем `innerHTML = 'Password not matches!'`.
4. Если значения полей `password` и `passwordVerify` равны, то:
 1. В массиве `storageUsers` ищем пользователя у которого email равен значению с формы.

2. Если юзер с таким email в localStorage найден, то в форме регистрации достаемся к div.error, добавляем ему класс active и устанавливаем innerHTML = `User with email \${email} already exist!`.
3. Если юзер с таким email в localStorage НЕ найден, то сохраняем все введенные в форме значения (кроме passwordVerify) в объект user. Добавляем свойства status: true, favourites: []. let user = { name: name, email: email, password: password, status: true, favourites: [] } Запускаем его в массив storageUsers. Обратно отправляем в localStorage свойство users. localStorage.setItem(`users`, JSON.stringify(storageUsers)); После этого редиректим юзера на страницу его аккаунта. document.location.href = `account.html`;
4. Так же, при сабмите формы нужно убирать для div.error класс active, если он ранее был добавлен.

Делаем глобальную переменную **userInSession**, в которой будет храниться объект залогиненного пользователя.

Для этого: в массиве **storageUsers** ищем юзера, у которого свойство **status === true**.

const userInSession = storageUsers.find(user => user.status === true);

4. Работа с Шапкой сайта


1. Достаемся к блоку где выводится либо Log in, либо Имя залогиненного юзера. **const headerUser = document.querySelector(`#headerUser`);**
2. Достаемся к блоку где выводится сердечко (количество избранных товаров залогиненного юзера). **const headerFavourites = document.querySelector(`#headerFavourites`);**
3. Достаемся к кнопке Log out. **const headerLogout = document.querySelector(`#headerLogout`);**
4. Если **userInSession** существует, то:
 1. В блоке **headerUser**:
 1. Вместо текста Log in выводим имя залогиненного юзера
headerUser.innerHTML = userInSession.name;
 2. Вместо ссылки на login.html пишем ссылку на страницу account.html.
headerUser.href = `account.html`;
 2. В блоке **headerFavourites**:
 1. Вместо ссылки на login.html пишем ссылку на страницу favourites.html.
headerFavourites.href = `favourites.html`;
 2. В блок **const headerFavouritesCount = document.querySelector(`#headerFavouritesCount`);** (количество избранных товаров залогиненного юзера) выводим длину массива свойства favourites. headerFavouritesCount.innerHTML = userInSession.favourites.length;
 3. Кнопке **Log out** добавляем класс active для отображения.
headerLogout.classList.add(`active`);

5. Навешиваем обработчик на кнопку Log out.

1. Навешиваем обработчик на событие click по кнопке **Log out**:
 1. В массиве **storageUsers** ЗАНОВО ищем юзера, у которого status === true.
const userInSession = storageUsers.find(user => user.status === true);
 2. Для найденного юзера меняем свойство status на false. userInSession.status = false;
 3. В localStorage обратно отправляем свойство users.
localStorage.setItem(`users`, JSON.stringify(storageUsers));

4. После этого редиректим юзера на главную страницу index.html.
`document.location.href = `index.html`;`

6. Для страницы index логика вывода товаров.

1. Достаемся к блоку вывода всех товаров. **`const categoriesContainer = document.querySelector(`#categoriesContainer`);`**
2. Запускаем цикл `forEach` по исходному массиву `PRODUCTS`.
3. При работе с каждым элементом массива `PRODUCTS` определяем в какую section-катеорию его выводить. Если на странице еще не создана section для данной категории, то создам ее и выводим в **`categoriesContainer`**. После этого рендерим блок с товаром и выводим во все нужные section-категории.
4. Вывод каждого товара:
 1. Кнопка Сердечка 
 1. Если **`userInSession`** и товар в `favourites`, то картинка сердечка **`images/product__favourite—true.png`**
 2. Если **`userInSession`** и товар не в `favourites` ИЛИ если юзер НЕ залогинен, то картинка сердечка **`images/product__favourite.png`**
 3. При нажатии на сердечко:
 1. Если пользователь НЕ залогинен, то редиректим его на страницу `login.html`
 2. Если пользователь залогинен и товар ранее не был добавлен в `favourites`, то:
 1. Меняем картинку сердечка на `images/product__favourite—true.png`
 2. Добавляем `id` товара в массив `favourites` в `localStorage` для данного юзера.
 3. В шапке сайта в блоке с выводом количества избранных товаров меняем значение.
 3. Если пользователь залогинен и товар ранее был добавлен в `favourites`, то:
 1. Меняем картинку сердечка на `images/product__favourite.png`
 2. Удаляем `id` товара с массива `favourites` в `localStorage` для данного юзера.
 3. В шапке сайта в блоке с выводом количества избранных товаров меняем значение.

7. Для страницы favourites.html логика работы с блоком Favourite Items.

1. Если **`userInSession`** существует, то:
 1. Достаемся к таблице избранных товаров юзера **`const favouriteTable = document.querySelector(`#favouriteTable`);`**
 2. Если у залогиненного юзера в свойстве `favourites` находится НЕ пустой массив, то проходимся по нему и каждый товар выводим в таблицу **`favouriteTable`** в виде `tr`. Пример данных в массиве `favourites`: `[10, 12, 1]`, где каждый элемент массива соответствует `id` товара в **ИСХОДНОМ МАССИВЕ `PRODUCTS`**.
3. Кнопка Сердечка  При нажатии на сердечко:
 1. Удаляем `id` товара с массива `favourites` в `localStorage` для данного юзера.
 2. В шапке сайта в блоке с выводом количества избранных товаров меняем значение.
 3. Удаляем `tr` товара