|  |  |
| --- | --- |
| 1. Валидация формы «Редактировать профиль» | |
| Валидируйте форму «Редактировать профиль». Для этого заготовьте элементы ошибок по макету в «Фигме». | 1) Класс . popup\_\_input-error\_active добавлен для поля ввода, чтобы менять цвет подчёркивания.  2) Span ошибки тоже нужно добавить, видимо. Добавляем полю ввода id, чтобы потом был доступ к спану. А спану присваеваем класс, связаный с id поля ввода - полеввода-эррор.  3) На все поля сделать такие спаны нужно. Сделано.  4) Задаём стили для этих спанов в активном состоянии form\_\_input-error\_active: цвет, размер шрифта и прочее. Будем этот класс присваивать для отображения ошибки. |
| Если поле формы «Редактировать профиль» не прошло валидацию, под ним должен появиться красный текст ошибки. | 1) Форме добавили novalidate.  2) Добавлять функции. |
| Настройки валидации такие:  Оба поля обязательные. | Обоим полям ввода уже был задан атрибут required. |
| В поле «Имя» должно быть от 2 до 40 символов. | Есть. |
| В поле «О себе» должно быть от 2 до 200 символов. | Есть. |
| Оба поля могут содержать только латинские и кириллические буквы, знаки дефиса и пробелы. Это нужно проверить с помощью регулярных выражений и вывести кастомное сообщение об ошибке. | 1) Добавили pattern. Кастомное сообщение об ошибке добавим при помощи data-\* атрибута.  2) В коде не забыть использовать setCustomValidity. |
| Во всех остальных случаях используйте стандартные браузерные тексты ошибок. |  |
| Если хотя бы одно из полей не прошло валидацию, кнопка «Сохранить» должна быть неактивной. Если оба поля прошли — активной. Цвета неактивных кнопок возьмите из макета. |  |
| Важно: при открытии модального окна редактирования профиля в поля формы подставляются валидные данные пользователя. Если открыть модальное окно редактирования профиля, ввести невалидные данные в поля ввода и закрыть окно, то при повторном открытии и заполнении данных формы профиля необходимо вызвать очистку ошибок валидации, которые могли остаться с прошлого открытия. |  |
| 2. Валидация формы «Новое место» | |
| Для этого заготовьте элементы ошибок по макету в «Фигме». |  |
| Если поле формы не прошло валидацию, под ним должен появиться красный текст ошибки. |  |
| Настройки валидации такие:  Оба поля обязательные. |  |
| В поле «Название» должно быть от 2 до 30 символов. | + |
| В поле «Ссылка на картинку» должен быть URL. |  |
| Поле «Название» может содержать латинские и кириллические буквы, знаки дефиса и пробелы. Нужно проверить с помощью регулярных выражений и вывести кастомное сообщение об ошибке. | + |
| В поле «Ссылка на картинку» должен быть URL. |  |
| И снова используйте стандартные браузерные тексты ошибок, кроме проверки регулярным выражением. |  |
| Если хотя бы одно из полей не прошло валидацию, кнопка «Сохранить» должна быть неактивной. Если оба поля прошли — активной. Цвета неактивных кнопок те же. |  |
| Важно: после добавления карточки на страницу форма очищается, при повторном открытии модального окна поля формы должны быть пустыми. Поэтому при очистке формы добавления карточки нужно вызвать задание неактивного состояния кнопке отправки, чтобы она не оставалась активной при повторном открытии модального окна с пустыми полями ввода. |  |
| 3. Требования к коду валидации | |
| Разбейте код валидации на функции. Подробнее об этом говорится в теме «Валидация форм». | + |
| Сделайте функцию enableValidation ответственной за включение валидации всех форм. | + |
| Пусть она принимает все нужные функциям классы и селекторы элементов как объект настроек. Все необходимое про объекты вы узнали ещё в предыдущем спринте.  // включение валидации вызовом enableValidation  // все настройки передаются при вызове  enableValidation({  formSelector: '.popup\_\_form',  inputSelector: '.popup\_\_input',  submitButtonSelector: '.popup\_\_button',  inactiveButtonClass: 'popup\_\_button\_disabled',  inputErrorClass: 'popup\_\_input\_type\_error',  errorClass: 'popup\_\_input-error\_active'  }); | +  Спросить у Алисы как вызвать именно вот так:  enableValidation({  formSelector: '.popup\_\_form',  inputSelector: '.popup\_\_input',  submitButtonSelector: '.popup\_\_button',  inactiveButtonClass: 'popup\_\_button\_disabled',  inputErrorClass: 'popup\_\_input\_type\_error',  errorClass: 'popup\_\_input-error\_active'  }); |
| В случае, если в поля «Имя» или «Название» введён любой символ, кроме латинской буквы, кириллической буквы и дефиса, вывести кастомное сообщение об ошибке: "Разрешены только латинские, кириллические буквы, знаки дефиса и пробелы". Текст ошибки разместить в data-\* атрибуте поля ввода. |  |
| Создайте функцию clearValidation, которая очищает ошибки валидации формы и делает кнопку неактивной. Эта функция должна принимать как параметры DOM-элемент формы, для которой очищаются ошибки валидации и объект с настройками валидации. Используйте функцию clearValidation при заполнении формы профиля во время её открытия и при очистке формы добавления карточки.  // очистка ошибок валидации вызовом clearValidation  clearValidation(profileForm, validationConfig); |  |
| Вынесите функциональность валидации форм в файл validation.js. Чтобы было чуточку понятнее — пример выше, вызов функций enableValidation и clearValidation должен находиться в файле index.js. А все другие функции, включая декларирование функции enableValidation и валидации форм, — в отдельном файле validation.js. |  |
|  |  |
| 1. Валидация формы «Редактировать профиль» | |
| Когда нет символов, совсем ничего не ввели – нужно сообщение «Вы пропустили это поле» и нижнее красное подчёркивание. | 1. Находим класс поля - popup\_\_input\_type\_name.  2. Находим какой класс рекомендовали для элемента ввода (input) и какой – для элемента с сообщением об ошибке. Для input рекомендуется id input\_type\_name, а для элемента с сообщением об ошибке рекомендуется класс input\_type\_name-error.  3. Если у свойства validity элемента input значение false, то будем input добавлять специальный класс popup\_\_input\_type\_error. Класс этот описали (без рамки и красное подчёркивание).  4. Находим где он присваивается. Нашли, в специальном объекте конфигурации. Этот класс универсальный. Подойдёт и для имени, и для описания и т.д. Проверяем как работает.  4.1. Где красная рамка у input? Ладно, пока непонятно. Пусть будет так.  5. Стиль сообщения об ошибке нужно описать. У span соответствующего будет класс popup\_\_input-error\_active. Сделано.  6. Когда вообще ничего не ввели, нужно отображать сообщение «» |
| Создайте функцию clearValidation, которая очищает ошибки валидации формы и делает кнопку неактивной. Эта функция должна принимать как параметры DOM-элемент формы, для которой очищаются ошибки валидации и объект с настройками валидации. Используйте функцию clearValidation при заполнении формы профиля во время её открытия и при очистке формы добавления карточки.  // очистка ошибок валидации вызовом clearValidation  clearValidation(profileForm, validationConfig); | Сделано через GPT. |
| Вынесите функциональность валидации форм в файл validation.js. Чтобы было чуточку понятнее — пример выше, вызов функций enableValidation и clearValidation должен находиться в файле index.js. А все другие функции, включая декларирование функции enableValidation и валидации форм, — в отдельном файле validation.js. |  |
| 3. Загрузка информации о пользователе с сервера | |
| Информация о пользователе должна подгружаться с сервера. Чтобы осуществить это, сделайте GET-запрос на URL. cohortId замените на идентификатор вашей группы:  GET https://nomoreparties.co/v1/:cohortId/users/me  При запросе не забудьте передать токен. Если запрос прошёл успешно, в ответе вы получите объект пользователя:  {  "name": "Jacques Cousteau",  "about": "Sailor, researcher",  "avatar": "https://pictures.s3.yandex.net/frontend-developer/ava.jpg",  "\_id": "e20537ed11237f86bbb20ccb",  "cohort": "cohort0"  } | Всё ясно, сделано. |
| Используйте свойства name, about и avatar в соответствующих элементах шапки страницы. Свойство \_id — идентификатор пользователя, в данном случае вашего. | Найти, где они у меня вообще в коде.  Так, видимо нужно из этого объекта вставлять значения вот в эти узлы:  const nameProfileNode = document.querySelector('.profile\_\_title');  const descriptionProfileNode = document.querySelector('.profile\_\_description');  Так, а с аватаром чуть сложнее.  Давай сначала name и about используем, а потом перейдём к avatar. Есть.  Теперь нужно понять как у узла с классом profile\_\_image сменить background-image. Сам код добавил, но если убрать из css фоновое изображение, то узел остаётся пустым. С сервера не подгружается картинка. Пока оставляю. Наверное, просто удалили оттуда? |
|  |  |
| 4. Загрузка карточек с сервера | |
| Начальные карточки должны подгружаться с сервера. Для этого сделайте GET-запрос:  GET https://nomoreparties.co/v1/cohortId/cards | Есть. |
| В ответ придёт JSON с массивом карточек, которые загрузили студенты вашей группы: | Да, приходит. |
| Используйте этот массив при отображении предзагруженных карточек, а от предыдущего способа отображения первоначальных карточек избавьтесь.  У каждой карточки есть свойства name и link — это заголовок и ссылка на картинку — они понадобятся при отображении каждой отдельной карточки. | Ага, там есть те же свойства, что и у массива initialCards. Получается, можно его также использовать? Кажется, да. Давай сначала попробуем. Не получилось. Мне кажется нужно саму процедуру добавления карточек добавить в промис. Пробуем. Ну да, так и есть. Работает, правда некоторые картинки не подгружаются. Но это видимо из-за проблемы с источником. |
| Как видите, у карточки также есть идентификатор — свойство \_id. Сейчас он вам не нужен, но скоро понадобится. | Ок |
| Обратите внимание, что для правильного отображения состояния кнопок лайка и удаления карточек необходимо знать \_id пользователя. | Да, понятно. |
| Его мы получаем только в ответе на запрос данных пользователя, и отображать карточки на странице следует только после его получения. Поэтому для загрузки данных пользователя и карточек необходимо воспользоваться методом Promise.all(). В него передается массив промисов, которые должны быть выполнены, т.е. наши запросы, а в блок .then мы попадем когда оба запроса будут выполнены. Тем самым это гарантирует, что у нас будет сразу и массив карточек и \_id пользователя для их отображения. | GPT написал код. Принципиально понятно, что происходит. Синтаксис, конечно, пока не запомнил сам. |
| 5. Редактирование профиля | |
| Отредактированные данные профиля должны сохраняться на сервере. Для этого отправьте запрос методом PATCH:  PATCH https://nomoreparties.co/v1/cohortId/users/me  В заголовках запроса, кроме токена, необходимо отправить Content-Type, а в теле — JSON с двумя свойствами — name и about. Значениями этих свойств должны быть обновлённые данные пользователя. Вот пример такого запроса:  fetch('https://nomoreparties.co/v1/cohortId/users/me', {  method: 'PATCH',  headers: {  authorization: 'c56e30dc-2883-4270-a59e-b2f7bae969c6',  'Content-Type': 'application/json'  },  body: JSON.stringify({  name: 'Marie Skłodowska Curie',  about: 'Physicist and Chemist'  })  }); | Это всё чудесно, только в метод как вставить введённые пользователем данные? Где они у нас вообще?  Ага, сейчас есть функция changeProfileInfo, которая никак не связывается с сервером, просто на странице меняет данные. Получается её нужно переписать?  Так, а если я отправлю на сервер новые данные, у меня userPromise их оттуда подгрузит?  Сначала переписываю changeProfileInfo, пусть она отправляет данные на сервер.  Теперь ещё надо сделать, чтобы эти данные были в форме, когда открываешь редактировать.  Нашёл, сделал. |
| Если обновление прошло успешно, в теле ответа от сервера вы получите обновлённые данные пользователя:  {  "name": "Marie Skłodowska Curie",  "about": "Physicist and Chemist",  "avatar": "https://pictures.s3.yandex.net/frontend-developer/common/ava.jpg",  "\_id": "e20537ed11237f86bbb20ccb",  "cohort": "cohort0",  }  Метод PATCH обычно используют для обновления сущностей, уже существующих на сервере. Обновление информации о пользователе — именно такой случай: пользователь уже есть, нужно просто обновить его данные. | А где я этот ответ увижу? Не знаю, но вроде работает.  Нашёл. Во вкладке разработчика же в разделе Сеть. |
| 6. Добавление новой карточки | |
| Чтобы добавить на сервер новую карточку, отправьте POST-запрос:  POST https://nomoreparties.co/v1/cohortId/cards  В заголовках запроса, кроме токена, необходимо отправить Content-Type, а в теле — JSON с двумя свойствами — name и link. В name должно быть название создаваемой карточки, а в link — ссылка на картинку. | Сначала надо найти, где вообще у нас функционал отправки карточки.  Ага, функция addCard. Кажется, весь функционал, что есть, должен остаться, он фактически добавляет карточку на страницу. И ещё нужно добавить функционал отправки на сервер.  Не проверяю. Всё по аналогии с редактированием профиля сделано. Хотя можно проверить, ладно.  Хм… После перезагрузки не отображается картинка… Нет её на сервере. Ну и когда на странице отрисовалась, не вижу ответа от сервера.  Получилось. Отлично. Просто метод забыл исправить. Метод был Patch, а нужен Post. |
| 7. Отображение количества лайков карточки | |
| У каждой карточки есть свойство likes — оно содержит массив пользователей, лайкнувших карточку:  {  "likes": [],  ...другие данные карточки  } | Да, видел. Когда возвращается массив карточек с сервера, там есть отдельный массив с лайками. |
| Сделайте так, чтобы на каждой карточке было написано, сколько у неё лайков. Для этого придётся сверстать дополнительный элемент. Его дизайн есть в Figma. | Нужно найти, где сердечко в html, вставить туда div, div присвоить класс.  Стоп, сначала где сердечко? Сердечко нашли, сам элемент с количеством добавили, теперь нужно в него прописать лайки. Это нужно сделать, когда создаётся карточка. Карточки мы уже запрашивали с сервера, оттуда и взять инфу.  Сделано. |
| 8. Удаление карточки | |
| Прежде чем браться за работу с API, исправьте элемент карточки. Сделайте так, чтобы иконка удаления была только на созданных вами карточках, так как удалять чужие карточки нельзя. | Тогда при добавлении кнопки удаления нужно проверять чья карточка.  В какой момент добавляется кнопка удаления?  Всё нашёл. Проверяем теперь id пользователя, добавившего карточку, прежде чем добавить кнопку удалить. |
| После этого реализуйте функциональность удаления карточки. Карточка должна удаляться, если в попапе удаления карточки пользователь нажал «Да».  Чтобы удалить карточку, отправьте DELETE-запрос:  DELETE https://nomoreparties.co/v1/cohortId/cards/cardId  Вместо cardId в URL нужно подставить параметр \_id карточки, которую нужно удалить. \_id каждой карточки есть в её JSON:  {  "likes": [],  "\_id": "5d1f0611d321eb4bdcd707dd", — вот он  ...другие данные карточки  }  В результате запрос на удаление этой карточки должен выглядеть так:  DELETE https://nomoreparties.co/v1/cohortId/cards/5d1f0611d321eb4bdcd707dd | Дак этот попап ещё нарисовать нужно. Давай просмотрим css попапа, который редактирует профиль.  Сам div с этим попапом в структуру добавил, но нужно ещё работать и работать.  Как его увидеть? Как у нас вообще появляется попап. По нажатию на кнопку. И что там происходит? Смотрим. Так, понял. По нажатию добавляется класс popup\_is\_opened.  Габариты посмотреть где и как указываются. Наверное именно специфический класс попапа. А нет, специфический – для обращений. А ещё внутри дива есть popup\_content. Может там?  Само содержимое есть, стили прописал. Теперь нужно добавить, чтобы появлялся попапэтот, когда происходит нажатие на кнопку удаления карточки. Есть.  Теперь функционал кнопки Да: нужно чтобы она удаляла карточку. Перенести удаление с нажатия на корзинку на эту кнопку. GPT попросил. Сделано.  И только вот теперь можно добавить удаление карточки на сервере. Для запроса попробовать шаблонные строки использовать?  Почему-то ошибку сервер возвращает, что только свои посты могу удалять. GPT доделал мне код. |
| 9. Постановка и снятие лайка | |
| Чтобы лайкнуть карточку, отправьте PUT-запрос:  PUT https://nomoreparties.co/v1/cohortId/cards/likes/cardId  Чтобы убрать лайк, нужно отправить DELETE-запрос с тем же URL:  DELETE https://nomoreparties.co/v1/cohortId/cards/likes/cardId  Вместо cardId в URL нужно подставить свойство \_id соответствующей карточки.  В ответе придёт обновлённый JSON с карточкой. Массив лайков в нём будет уже обновлён.  При постановке и снятии лайка сердечко должно менять цвет, а счётчик лайков увеличиваться или уменьшаться.  Чтобы изменить количество лайков, нужно отправить на сервер запрос с соответствующим методом. Рекомендуем брать количество лайков из ответа сервера, а не из вёрстки. Иначе могут возникнуть ошибки, когда два пользователя одновременно лайкнут одну карточку. | Найти функцию, которая обрабатывает клик по сердечку. Она должна:  1) проверить лайкнута ли карточка пользователем,  2) изменить цвет сердечка на противоположный,  если лайка не было  3) отправить PUT-запрос на сервер,  если был  3) отправить DELETE запрос на сервер,  4) из пришедшего ответа выбрать количество лайков и обновить его на карточке.  Функцию likeCard нужно переписать. 1 и 2 уже делаются.  Где взять id карточки для передачи лайка? Объект кнопки у нас содержится в event.target. А как целиком карточку получить? Разобрался. Лайки отправляются. Только при отрисовке страницы у сердечка нет цвета, оно не помнит, что я его лайкнул. Надо найти, где отрисовывается элемент и добавить там проверку лайка от меня. Так, сейчас правильно отображается лайк при загрузке.  Почти всё сделано, только ещё нужно при отправке PUT и DELETE запроса тут же на страницу отрисовывать количество лайков. И брать это количество с сервера.  Жесть, очень сложно. Потратил на это примерно 2-3 часа. |
| 10. Обновление аватара пользователя | |
| Чтобы сменить аватар, отправьте такой PATCH-запрос:  PATCH https://nomoreparties.co/v1/cohortId/users/me/avatar  В теле запроса передайте JSON с единственным свойством — avatar. Это свойство должно хранить ссылку на новый аватар. Если отправить не ссылку, сервер вернёт ошибку. | Получается осталось только сделать функционал обработки клика.  Нужно найти кнопку, которая сохраняет данные пользователя. Ей нужно добавить свой класс и добавить обработчик. Думаю так. Почти всё сделано.  Нужно теперь ещё сделать так, чтобы при отрисовке страницы фото профиля загружалось с сервера. Супер, работает. |
| При наведении указателя мыши на аватар, на нём должна появляться иконка редактирования: | Начинаем с этого. Слушатель нужно добавить? Типа mousover? Или при помощи псевдообъекта? Да, сначала :hover пробуем. И при наведении добавится оверлей? Или как? Смотрю что в фигма.  Так, оверлей нужно взять тот, что использовали при открытии попапа. Только карандаш прозрачный у меня… Ладно, оставляю. Иконка ж появляется. |
| А при клике — открываться форма. Эту форму нужно сделать. В ней должно быть одно поле — ссылка на новый аватар. Иконка редактирования аватара и форма загрузки есть в Figma. | Дальше вот это надо делать. Куда слушатель клика добавить? Весь div сделать кнопкой? Нет, оставляю div как есть, просто ему клик добавили. Теперь отрисовываем форму.  Где у нас там формы вообще хранятся?  Всё нашёл, при нажатии открывается попап, куда можно загрузить аватар. |
| Хоть в этой форме и всего одно поле, но его тоже необходимо валидировать:  Поле обязательное  Поле содержит именно URL в корректном формате | Надо понять что это значит. Я этот попап сделал из попапа, который добавляет новую карточку. Да, проверка работает. Это значит, что вся моя функция валидации и эту форму тоже обрабатывает и там настраивает валидацию? Думаю да. Классы-то правильно прописаны. Так что здесь ничего не делаю. |
| 11. Улучшенный UX всех форм | |
| Поработайте над UX. При редактировании профиля уведомите пользователя о процессе загрузки, поменяв текст кнопки на: «Сохранение...», пока данные загружаются: | Ищу где участок кода, который отвечает за редактирование данных в профиле. Нашёл, changeProfileInfo функция.  Так, секунду. Давай подумаем как не писать для каждой формы код для смены Сохранить на Сохранение… 3 из 4 попапа будут так делать, у всех этих 3 кнопок есть общий класс: popup\_\_button. Надо посмотреть как на этот класс вешается обработчик и переместить изменение текста с Сохранить на Сохранение… туда. Смотрю.  Пока не вижу тут места для оптимизации. Потому что слушателем вешается функция – на каждую кнопку своя.  Тогда просто редактируем ChangeProfileInfo. Код скорректировал. Давай исправим что попадает в форму редактирования профиля в качестве плэйсхолдера. Где оно вообще. Исправил. Всё работает. |
| Сделайте то же самое для формы добавления новой карточки | addCard, вроде. Поеблись тут, конечно. Но вроде всё работает. |
| и обновления аватара. | Надо закрытие формы в промис тогда поставить. Так, что нужно:  0) смена текста с Сохранить на Сохранение…  1) закрытие формы в промис перенести,  2) обновление аватара на странице – после ответа с сервера, получается тоже в промисе.  Действителен ли url проверки у меня нет. |