Общая информация

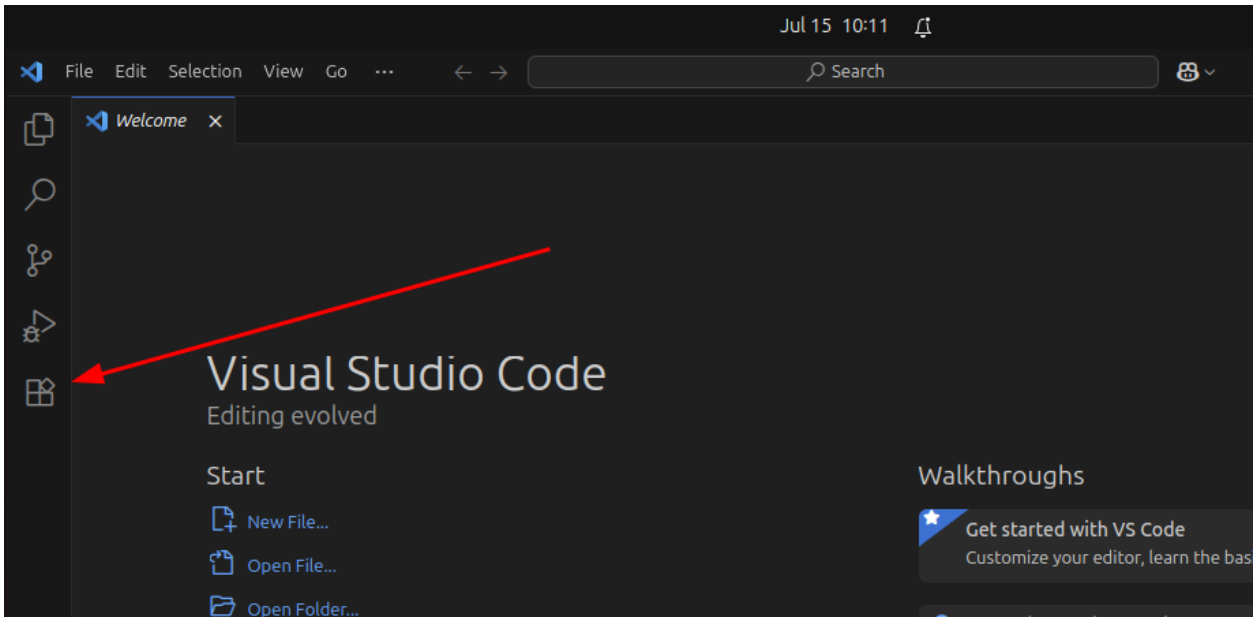
Верстку проверить на валидаторе: невалидная верстка очень плохо конвертируется в компоненты.

https://validator.w3.org/

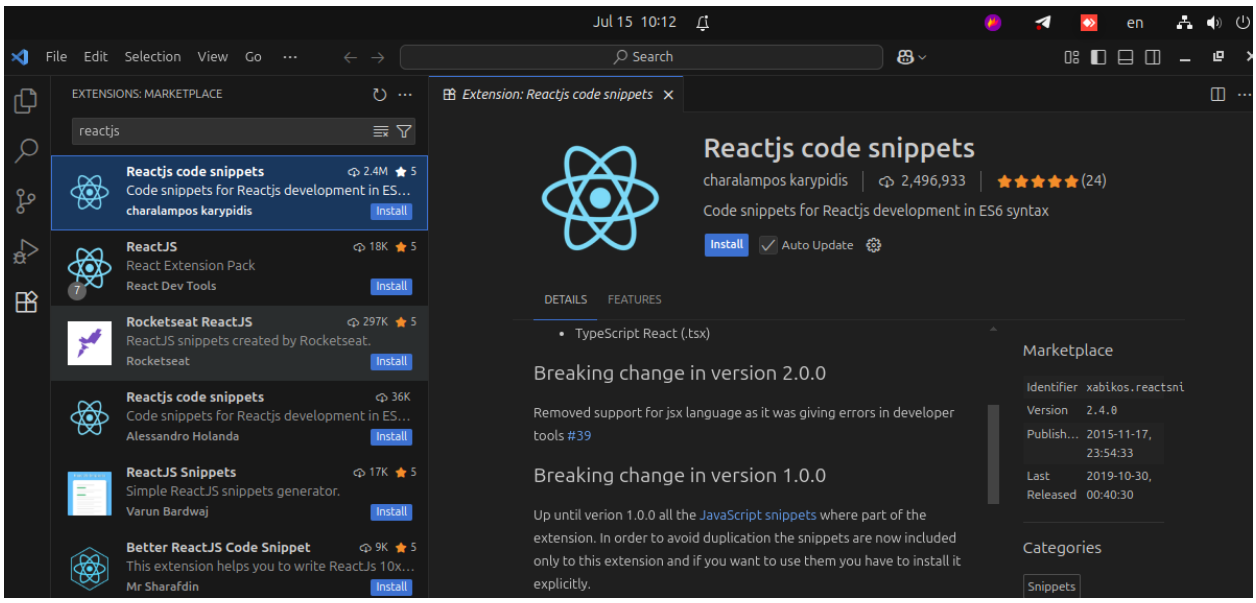
Конвертер для перевода HTML в JSX:

https://transform.tools/html-to-jsx

Установка плагина



В строке поиска пишем reactjs.



Находим именно этот плагин: от charalampos karypidis.

Константы

export const LOGIN = "/login";

export const ACCOUNT = "/account";

export const CHANGE_EMAIL = "/email";

export const CHANGE_PHONE = "/phone";

export const PETS = "/pets";

export const CAROUSEL = "/slider";

export const SEARCH = "/search";

export const ADVANCED_SEARCH = "/order"

export const SERVER_URL = "https://pets.сделай.site"

export const API_URL = SERVER_URL + "/api";

export const API_URL_REGISTRATION_PATH = API_URL + "/register";

export const API_URL_SUBSCRIPTION_PATH = API_URL + "/subscription";

export const API_URL_LOGIN_PATH = API_URL + "/login";

export const API_URL_USERS_PATH = API_URL + "/users";

export const API_URL_USERS_POSTS = API_URL_USERS_PATH +"/orders"; // Объявления, добавленные пользователем.

export const API_CHANGE_EMAIL_URL = API_URL_USERS_PATH + CHANGE_EMAIL;

export const API_CHANGE_PHONE_URL = API_URL_USERS_PATH + CHANGE_PHONE;

export const API_PETS_URL = API_URL + PETS;

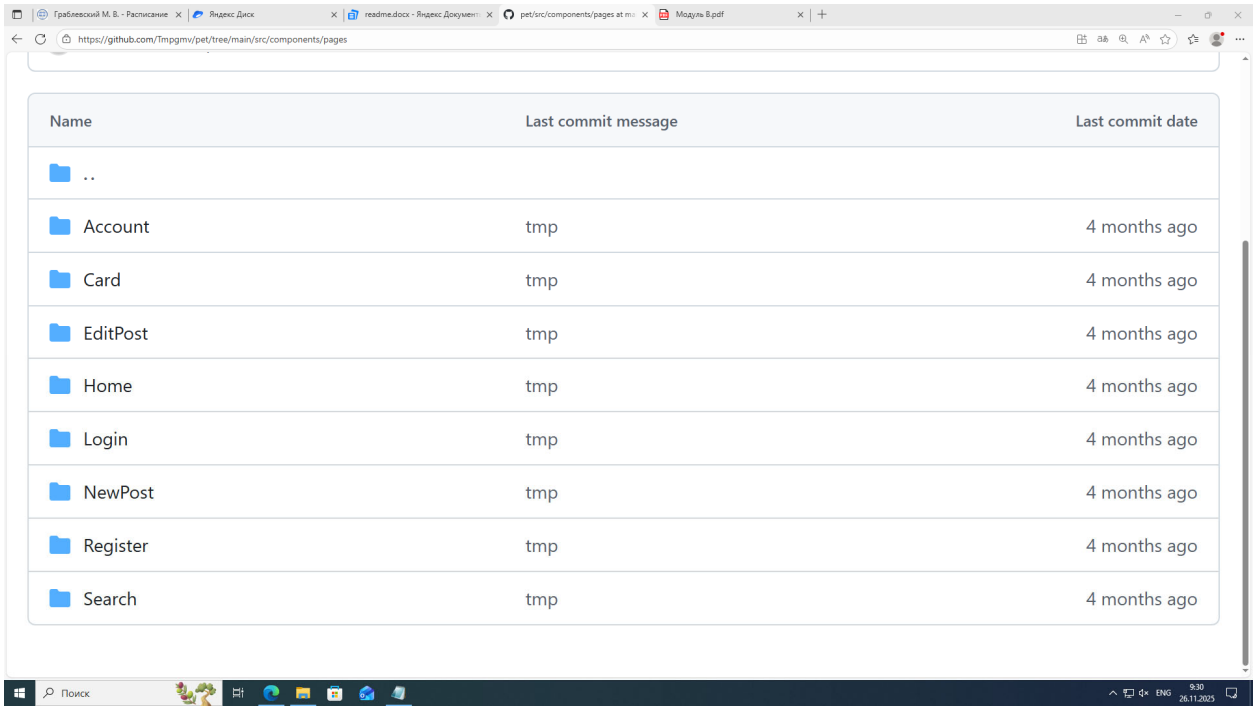
export const API_CAROUSEL_URL = API_PETS_URL + CAROUSEL;

export const API_SEARCH_URL = API_URL + SEARCH;

export const API_ADVANCED_SEARCH_URL = API_URL + SEARCH + ADVANCED_SEARCH;

export const DEBUG = false; // Самим спрограммировать, чтобы нативные проверки не выполнялись. Т.е. отсылать на сервер все, что есть. Так можно сэкономить время на отладку программы.

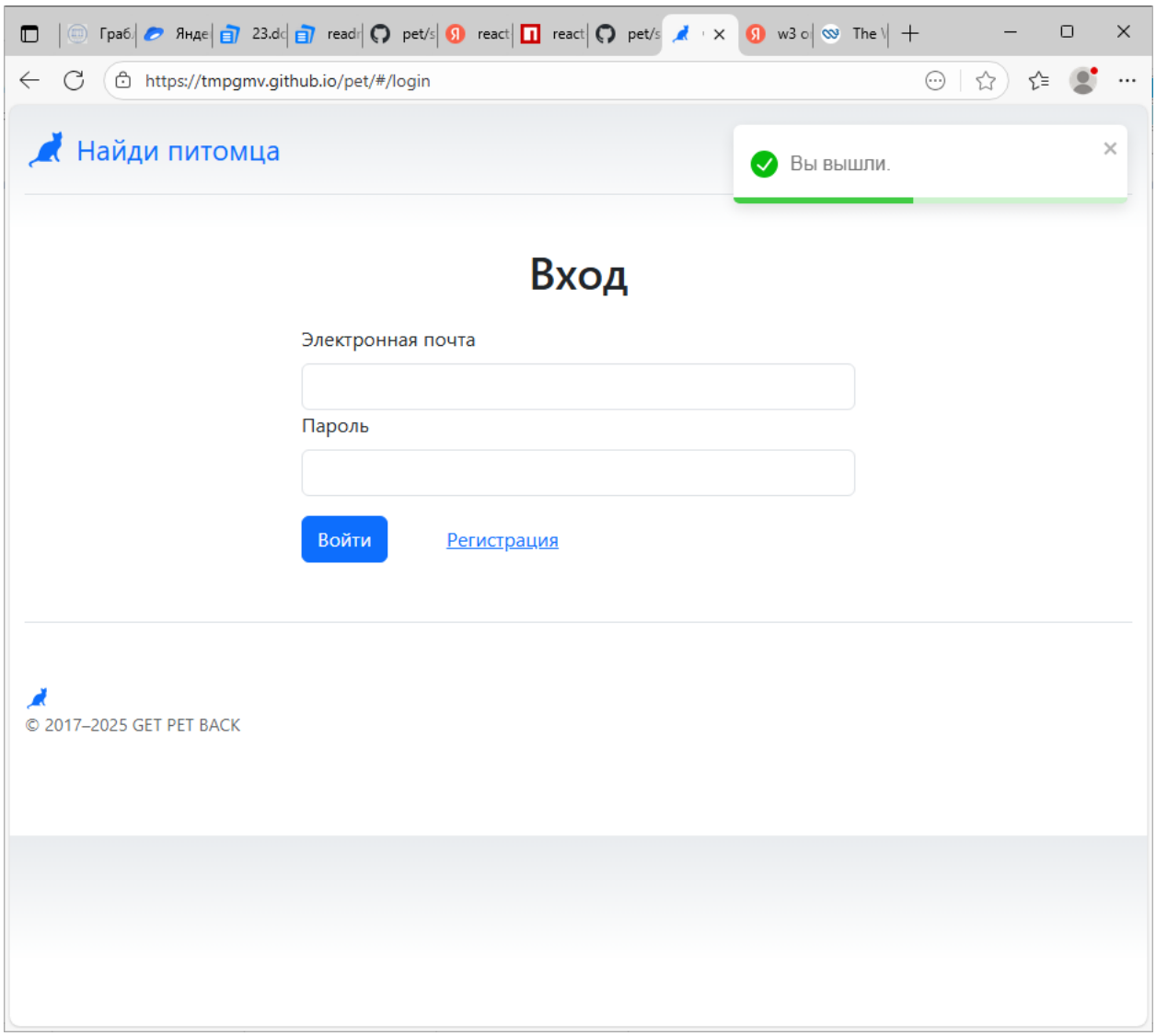
Страницы



Стиль

Сообщения пользователю

https://www.npmjs.com/package/react-toastify



Отладка

1. Если в JavaScript-коде написать

debugger;

То дебаггер остановится в этом месте. Вместо точек останова.

2. Полей много. У многих – нативная валидация. При отладке удобно отправлять на сервер, что есть. И смотреть, что отвечает сервер. В противном случае придется заполнять формы – время тратить.

import { DEBUG } from "./constants";

```
/* Пример применения:


*/
export function attr(props) {
  let result = DEBUG ? props.debugValue : props.prodValue;
  return result;
}
```

Логин

```
import { API_URL_USERS_PATH, ACCOUNT, API_URL_LOGIN_PATH } from "../../../general/constants";


function saveToken(token) {
  localStorage.setItem("token", token);
}


function Form() {
    let formData = $("#" + formId).serialize();
    function saveUserInfo(data) {
      localStorage.setItem("name", data.name);
      localStorage.setItem("phone", data.phone);
      localStorage.setItem("email", data.email);
      localStorage.setItem("registrationDate", data.registrationDate);
      localStorage.setItem("ordersCount", data.ordersCount);
      localStorage.setItem("petsCount", data.petsCount);
    }
```

Применение.

```
function requestUserInfo(token) {
  $.ajax({
    url: API_URL_USERS_PATH,
    method: "GET",
    beforeSend: (xhr) => {
      xhr.setRequestHeader("Authorization", "Bearer " + token);
    },
    dataType: "json",
  })
    .done((data) => {
      saveUserInfo(data);
      navigate(ACCOUNT, {
        state: {
          toast: {
            type: "success",
            message: "Вы вошли.",
          },
          from: location,
        },
      });
    })
    .fail((jqXHR) => {
      throw new Error(
        "Не удалось получить с сервера данные о пользователе!"
      );
    });
}

$.ajax({
  url: API_URL_LOGIN_PATH,
```

```
    method: "POST",
    data: formData,
    dataType: "json",
  })
    .done(function (dataJson, textStatus, jqXHR) {
      let token = dataJson.data.token;
      saveToken(token);
      requestUserInfo(token);
    })
    .fail(function (jqXHR, textStatus, errorThrown) {

      notifyFailure();

      let responseJson = jqXHR.responseJSON;

      let errors = responseJson.error.errors;

      $.each(errors, function (key, data) {
        let unitedErrorText = data.join();
        $("#" + key).addClass("is-invalid");
        let selector = "#" + key + "Error";
        $(selector).text(unitedErrorText);
      });

    });
}

return (
  <form
    id="login-form"
    method="post"
```

```
      className="col-12 col-md-6 mx-auto mt-4"
      onSubmit={(event) => handleSubmit(event, "login-form")}
    >
      <EmailInput />
      <PasswordInput />

      <div className="d-flex align-items-center mt-3">
        <Button btnText="Войти" />

        <Link to="/register" className="ms-5 text-primary">
          Регистрация
        </Link>
      </div>
    </form>
  );
}
```

jQuery Autocomplete

https://jqueryui.com/autocomplete/

```
import "jquery-ui/ui/widgets/autocomplete";

/*

ВАЖНО: при работе с jQuery UI autocomplete необходимо

скрытьв вспомогательные сообщения.

см. в style.css ui-helper-hidden-accessible
```

```
*/

/* Глобально скрыть jQuery UI autocomplete live region
Иначе где-то - предположительно, в левом нижнем углу - будут появляться
надписи наподобие:

“X results are available, use up and down arrow keys to navigate.”

“No search results.”
*/
.ui-helper-hidden-accessible {
  display: none !important;
}
```

Запросить доступные варианты

```
        <input
          id={INPUT_ID}
          type="text"
          className="form-control"
          placeholder="Кого ищем?"
          name="query"
          aria-describedby="button-addon2"
          required={required}
          defaultValue={query ? query : undefined}
          onChange={(e) => {
            const value = e.target.value;
            if (value.length > 3) {
              setTimeout(() => requestAvailableVariants(value), 1000);
            }
          }}
```

```
        />

 const INPUT_ID = "quick-search-input";
const [availableVariants, setAvailableVariants] = useState([]);



 function refreshAvailableVariants() {
   $("#" + INPUT_ID).autocomplete({
     source: availableVariants,
     minLength: 4,
   });
 }



 function requestAvailableVariants(query = null) {
   let url = API_SEARCH_URL;
   if (query) url += "?query=" + query;

   $.ajax({
     url: url,
     method: "GET",
     dataType: "json",
   })
     .done((dataJson) => {
       const descriptions = dataJson.data.orders.map((item) => item.description);
       const uniqueDescriptions = [...new Set(descriptions)];
       setAvailableVariants(uniqueDescriptions);
     })
     .fail(() => {
```

```
      toast.error("Не удалось получить с сервера данные для поисковых подсказок!", {
        toastId: "availableVariantsError",
      });
    });
  }
```

Новое объявление

/*

Если флаг register не взведен, просто емейла недостаточно

для отнесения объявления к пользователю. Это будет анонимное объявление.

Т.е. как свое ни один пользователь не сможет его увидеть (отредактировать / удалить).

Если флаг Register взведен, то требуются пароль и подтверждение.

Если флаг Register после этого снят, то можно допустить логическую ошибку, скрыв блок. Т.е. форма по факту

будет заполнена, но скрыта. Это будет логическая ошибка, потому что сервер не ждет флаг Register. Т.е. флаг

будет на сервер отправлен, но это лишняя информация, ее сервер просто отбросит.

Вывод: будет логической ошибкой, если сервер получит пароли, а пользователь не хотел регистрироваться.

Правильный прием: средствами JavaScript надо обязательно удалить поля password и password_confirmation.

Не очистить эти поля, а именно удалить.

Потому что если просто очистить, эти поля на сервер будут отправлены пустыми.

Если пароли на сервер отправлены, то будет попытка аутентификации.

Т.е. и зарегистрированный пользователь тоже должен отправить пароль.

Причина: Bearer + токен на сервер не отправляются.

Если пользователь не аутентифицирован, будет выполнена попытка

зарегистрировать пользователя. И сразу отнести объявление к нему.

*/

```
<form
  id={formId}
  method="post"
  encType="multipart/form-data"
  className="row g-3 mt-2 col-12 col-md-6 mx-auto"
  onSubmit={(event) => handleSubmit(event)}
>
```

https://learn.javascript.ru/formdata

```
function handleSubmit(event) {
  event.preventDefault();
  clear();
  let theForm = document.getElementById(formId);
  let formData = new FormData(theForm);
  let request = $.ajax({
    url: API_PETS_URL,
    method: "POST",
    data: formData,
    beforeSend: function (xhr) {
      xhr.setRequestHeader("Authorization", "Bearer " + TOKEN);
    },
    contentType: false,
    processData: false,
    dataType: "json",
  });
```