

Гайд по установке Svelte.

В данном руководстве мы установим Svelte с связке с Routify и напишем небольшое приложение, которое будет отсылать запрос на тестовый сервер в интернете и получать ответ от него.

Онлайн копия данного руководства (возможно с изменениями) находится по адресу:
https://github.com/ravsii/svelte-installation-guide-ru/blob/main/GUIDE_RU.md

Большинство кода в примерах вырезано ввиду его объёмности, однако присутствует в полном объёме в репозитории документа.

Содержание

- Гайд по установке Svelte.
 - Содержание
 - Что это и почему
 - А зачем Routify?
 - Установка
 - Установка зависимостей
 - Установка Svelte
 - Опционально
 - Тестовый запуск
 - Tailwind CSS
 - Разработка
 - Для пользователей VSCode
 - Как общаться с другими сервисами (Spring в нашем случае)
 - Структура проекта
 - Файлы `_layout.svelte`
 - Компоненты
 - Отправка данных от ребёнка к родителю
 - Вложенные layout
 - Директивы `bind` , `on`
 - Привязка данных к переменным
 - Обработчики событий `on`
 - Отправляем запрос

- CORS
 - Полезные ссылки

Что это и почему

Svelte это фронтэнд фреймворк, который позволяет очень просто и быстро разрабатывать веб-приложения. Если вы слышали о таких вещах как React, AngularJS, то это из этой же серии.

По моему скромному мнению Svelte намного приятнее и удобнее React, а Angular в целом сейчас используют гораздо реже. Он довольно простой в освоении, хоть и имеет комьюнити поменьше, что немножко влияет на количество доступной информации и количество модулей от сторонних разработчиков.

А зачем Routify?

Svelte, в отличии от своих аналогов, представляет только [реактивность](#), работы с компонентами и всё что непосредственно связано с тем, за что отвечает браузер. Однако серверные задачи (такие как создание дополнительных страниц на сайте или роутинг) он выполнять не умеет. Для этого существуют фреймворки, которые реализуют серверную часть для него.

Их список:

- [Svelte Kit](#) – официальный фреймворк
- [Sapper](#) - предшественник Svelte Kit
- [Routify](#) - фреймворк от сторонних разработчиков, который специализируется на роутинге.

Routify довольно прост, так как делает работу со страницами простой в использовании. Он так же даёт возможность использовать вложенные layout (об этом позже). Больше нам от серверной части на фронтэнде ничего не требуется, так что нашим требованиям он отвечает.

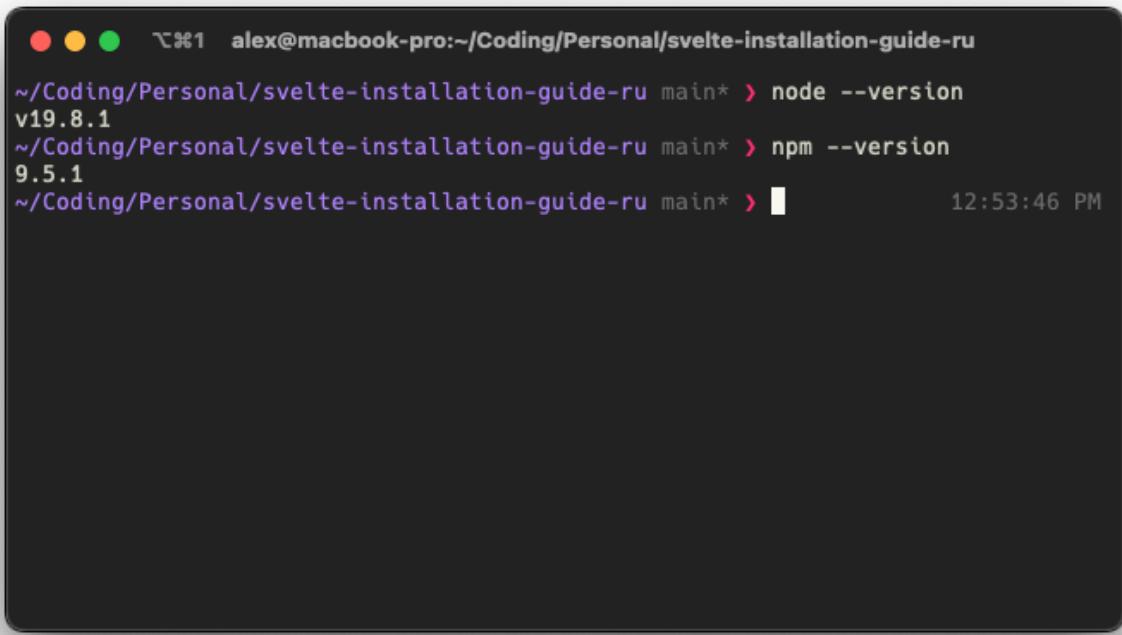
Установка

Установка зависимостей

Скачиваем NodeJS с [официального сайта](#), LTS версии будет достаточно, но можно и самую последнюю версию.

Если установка прошла успешно, то вы должны увидеть следующий результат в консоли.

Зайти в консоль на windows можно комбинацией клавиш *Win+R* а затем *cmd.exe* , либо *Shift+ПКМ* внутри любой папки по пустому месту, а затем *Открыть окно PowerShell* либо что-то подобное, если у вас *Windows 7* либо ниже.



A screenshot of a dark-themed terminal window on a Mac. The title bar shows the user's name and the current directory: `alex@macbook-pro:~/Coding/Personal/svelte-installation-guide-ru`. Below the title bar, there are three command-line entries: `node --version` (output: v19.8.1), `npm --version` (output: 9.5.1), and a partially visible command starting with `~/.Coding/Personal/svelte-installation-guide-ru main*`. In the bottom right corner of the terminal window, the time is displayed as 12:53:46 PM.

Версия не обязательно должна совпадать с той что на скриншоте.

Если нет - то гуглите текст ошибки, скорее всего до вас кто-то уже сталкивался с подобным.

Установка Svelte

Так как Svelte мы будем устанавливать в Routify - мы можем использовать уже готовый "стартер" проект, который они предоставляют в [официальном репозитории](#).

Создаём папку, где у нас будет хранится проект, заходим туда через консоль и ВВОДИМ:

```
npm init routify
```

```
npm init routify
~/Coding/Personal/svelte-installation-guide-ru main* > npm init routify
Need to install the following packages:
  create-routify@1.3.2
Ok to proceed? (y) █
```

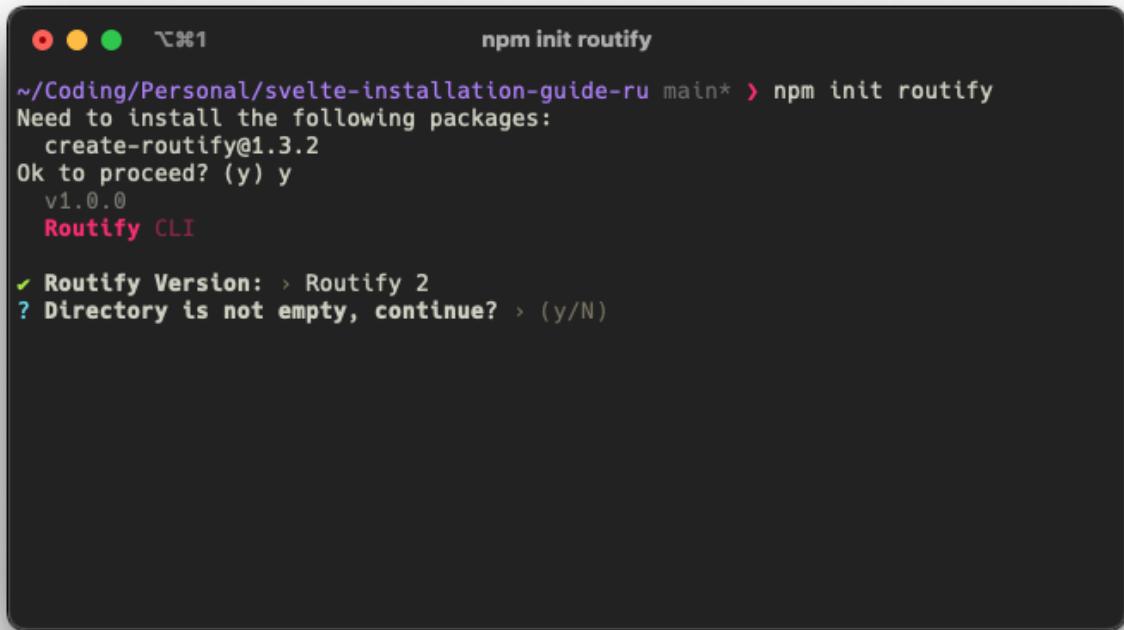
Соглашаемся (Пишем: "у" английскую).

```
npm init routify
~/Coding/Personal/svelte-installation-guide-ru main* > npm init routify
Need to install the following packages:
  create-routify@1.3.2
Ok to proceed? (y) y
v1.0.0
Routify CLI

? Routify Version: > - Use arrow-keys. Return to submit.
> Routify 2
  Routify 3 [BETA]
```

Далее нам предлагают выбрать версию. Выбираем Routify 2 (синий текст) и жмём Enter.

Важно: Если вторая версия уже не актуальная, и вам автоматически ставится 3, то в некоторых моментах надо будет самому в доках искать актуальную информацию. Сейчас на Beta версии изменено поведение `_layout.svelte`, которые стали модулями, работа с `$url()`, возможно что-то ещё будет изменено.



```
npm init routify
~/Coding/Personal/svelte-installation-guide-ru main* > npm init routify
Need to install the following packages:
  create-routify@1.3.2
Ok to proceed? (y) y
  v1.0.0
  Routify CLI

✓ Routify Version: > Routify 2
? Directory is not empty, continue? > (y/N)
```

Если ваша директория не пустая (как у меня), то вас попросят утвердить, что вы хотите установить проект именно в эту директорию.

Вполне возможно, что на этом этапе может вылезти ошибка, установите проект в пустую директорию, а потом переместите куда вам нужно.

После установки у вас должны появится папки `public` и `src`, и много всяких файлов.

```
alex@macbook-pro:~/Coding/Personal/svelte-installation-guide-ru
```

```
~/Coding/Personal/svelte-installation-guide-ru main* > ls          01:15:02 PM
GUIDE_RU.md
LICENSE
README.md
guide_images
index.html
netlify.toml
package-lock.json
package.json
postcss.config.js
public
src
svelte-installation-guide-ru.code-workspace
vercel.json
vite.config.js
~/Coding/Personal/svelte-installation-guide-ru main* > █          01:15:03 PM
```

Опционально: Tailwind CSS

Можно не устанавливать его и писать вёрстку самому, тут уже как хотите. Но данный фреймворк является практически обязательным, если планируете становиться Frontend разработчиком.

Для установки Tailwind воспользуемся двумя источниками, [обычной установкой](#) и [установкой для Vite](#). Второй источник нужен для добавления Tailwind к Vite (серверный фреймворк, который использует Routify).

```
npm install -D tailwindcss postcss autoprefixer # Установим зависимости.
npx tailwindcss init # Создаём стандартный конфигурационный файл.
```

Результат:

```
● ● ●  ~%1 alex@macbook-pro:~/Coding/Personal/svelte-installation-guide-ru
~/Coding/Personal/svelte-installation-guide-ru main* > npm install -D tailwindcs
s

added 45 packages, changed 1 package, and audited 278 packages in 2s

53 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
~/Coding/Personal/svelte-installation-guide-ru main* > npx tailwindcss init

Created Tailwind CSS config file: tailwind.config.js
~/Coding/Personal/svelte-installation-guide-ru main* > 01:24:53 PM
```

Из [гайда для SvelteKit](#) укажем, в каких файлах будет работать Tailwind, и запишем это в наш только-что созданный `tailwind.config.js`, который теперь должен выглядеть примерно так:

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ['./src/**/*.{html,js,svelte,ts}'],
  theme: {
    content: [],
    extend: {},
  },
  plugins: [],
}
```

Далее переходим в `/src/global.css`, удаляем всё оттуда и добавляем директивы импорта Tailwind.

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Далее мы должны импортировать Tailwind в корень Svelte проекта. Заходим в соседний файл `App.svelte` и добавляем в импорт наш `global.css`.

```
<script>
  import { Router } from '@roxi/routify'
  import { routes } from '../routify/routes'
```

```
import './global.css'  
</script>  
  
<Router routes="{routes}" />
```

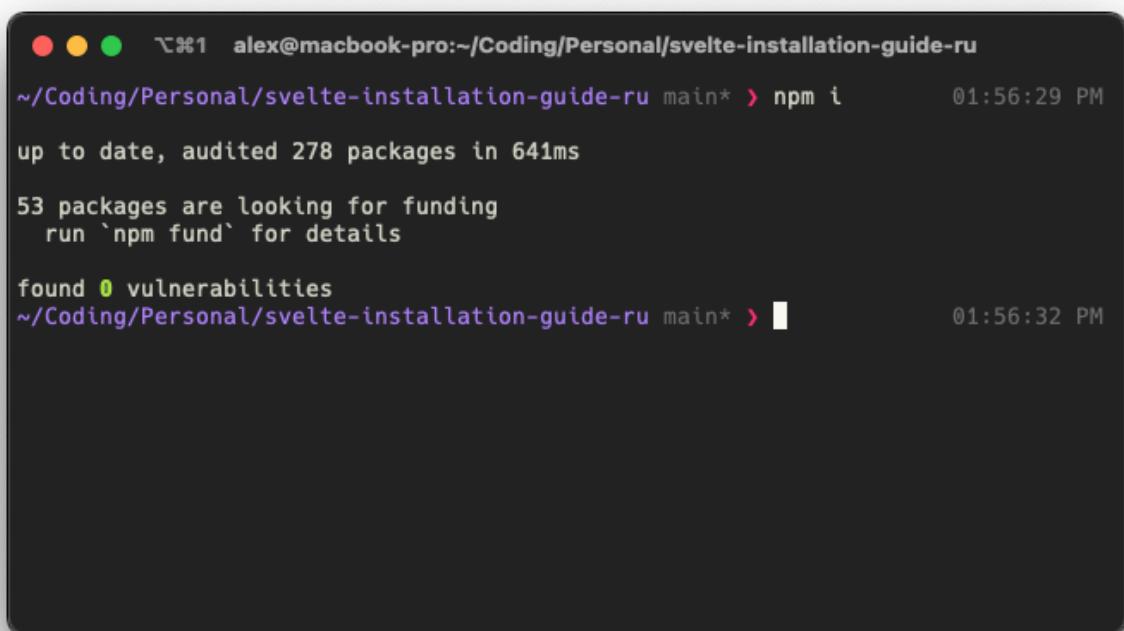
Для корректной работы Tailwind директорий текст в файле `postcss.config.js` необходимо заменить на следующий:

```
module.exports = {  
  plugins: [require('tailwindcss'), require('autoprefixer')],  
}
```

Тестовый запуск

Перед запуском установить все (возможно отсутствующие) зависимости:

```
npm i
```



A screenshot of a terminal window on a Mac. The title bar says "alex@macbook-pro:~/Coding/Personal/svelte-installation-guide-ru". The command "npm i" was run, and the output shows:

```
alex@macbook-pro:~/Coding/Personal/svelte-installation-guide-ru ~ % npm i  
~/Coding/Personal/svelte-installation-guide-ru main* > npm i          01:56:29 PM  
up to date, audited 278 packages in 641ms  
53 packages are looking for funding  
  run `npm fund` for details  
found 0 vulnerabilities  
~/Coding/Personal/svelte-installation-guide-ru main* > █          01:56:32 PM
```

И запустим сервер командой:

```
npm run dev
```

Если возникает ошибка: `sh: run-p: command not found`, то попробуйте установить зависимость и снова запустить:

```
npm i npm-run-all
```

Если ошибка другая - тут только гугл.

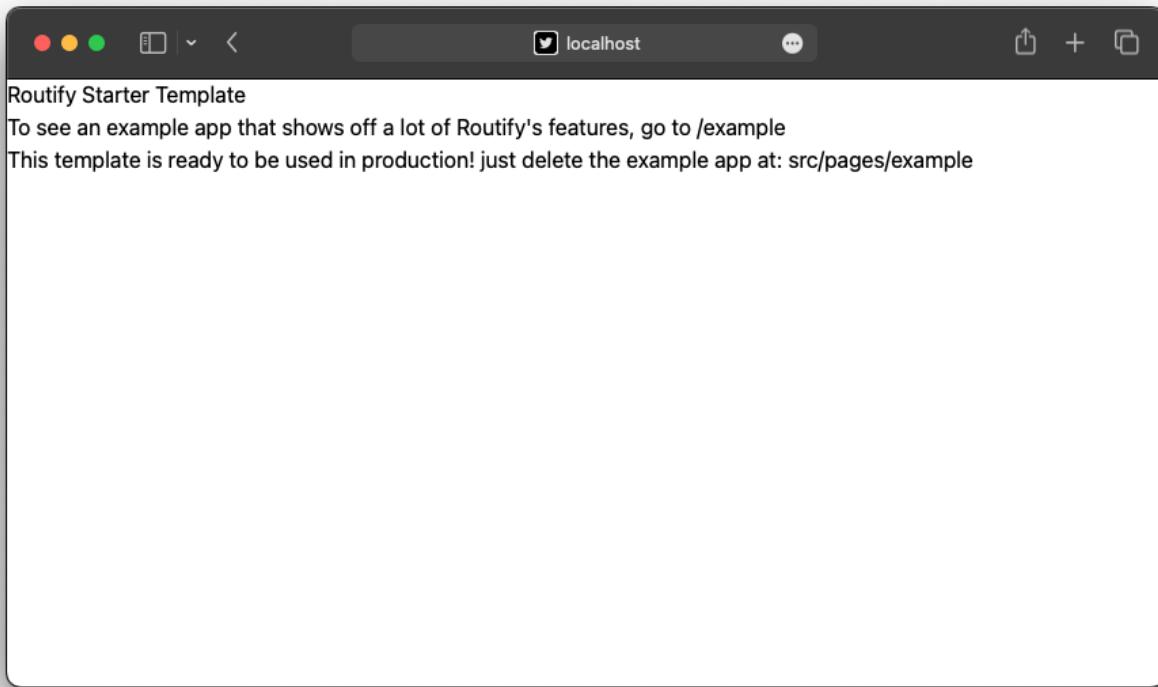
После запуска сервера должен появится следующий текст.



A screenshot of a dark-themed terminal window. At the top, there are three colored window control buttons (red, yellow, green) followed by the text "npm run dev". Below this, the terminal displays the following text:

```
VITE v3.1.0 ready in 197 ms
+ Local: http://localhost:5000/
+ Network: use --host to expose
```

И при заходе на <http://localhost:5000/> вы должны увидеть стартовую страницу. (Если не устанавливали Tailwind, возможно визуально она будет другая).



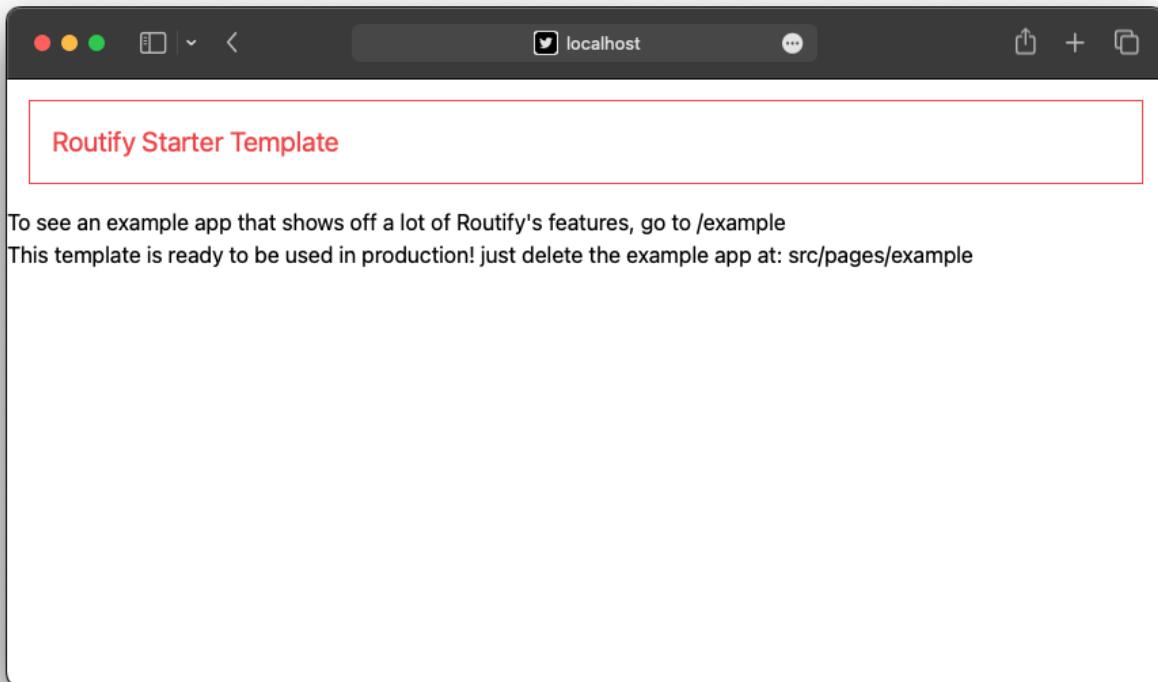
Tailwind CSS: проверка импорта

Проверить его достаточно просто. Во первых, визуально ваша страница должна выглядеть идентично скриншоту выше, хотя это лишь означает, что все стили которые мы удаляли - удалились.

Добавим в файл `/src/pages/index.svelte` немного 👍 классов👍 и посмотрим на результат. Например, поменяем заголовок на:

```
<h1 class="p-4 m-4 text-xl text-red-500 border border-red-500">  
  Routify Starter Template  
</h1>
```

Перезапускать сервер не нужно. Если вы запускали его с помощью `npm run dev`, то все изменения будут обновляться автоматически.



Если у вас всё идентично – поздравляю.

Разработка

В этой главе мы напишем максимально простое приложение, которое показывает как принимать и отправлять запросы, добавлять страницы, переходы между ними, работу с компонентами.

Для пользователей VSCode:

Установите эти плагины и ваша жизнь сильно облегчится:

- [Svelte for VS Code](#)
- [Tailwind CSS IntelliSense](#)
- [Headwind](#) - Сортировщик классов для TW

Как общаться с другими сервисами (Spring в нашем случае)

Общение будет происходить через REST-API, которое сервер нам предоставляет. Важный момент, что всё общение происходит через API-запросы с помощью JSON, и фронтэнд не имеет никакого доступа к данным, кроме тех, которые вы посыпаете с сервера. Учитывайте это при разработке.

Структура проекта

Больше всего нас интересует папка `/src`, так как в ней находится весь исходный код.

В изначальном варианте она выглядит так.

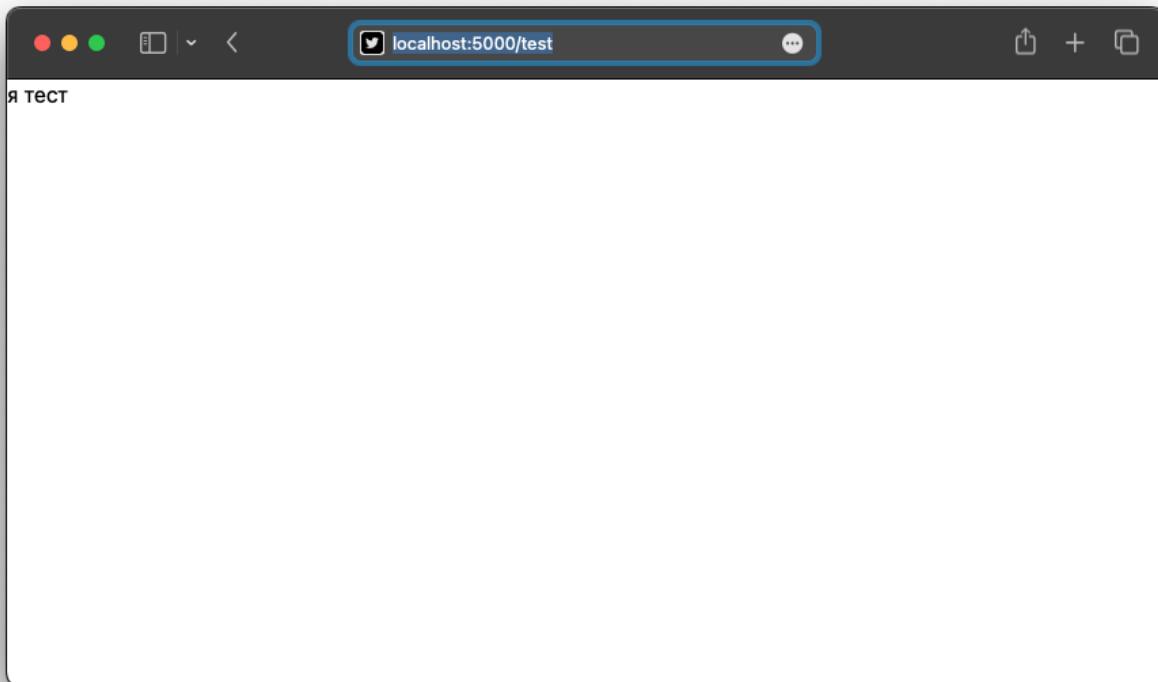


```
alex@macbook-pro:~/Coding/Personal/svelte-installation-guide-ru
~/Coding/Personal/svelte-installation-guide-ru main* > find ./src      02:20:21 PM
./src
./src/global.css
./src/App.svelte
./src/main.js
./src/pages
./src/pages/index.svelte
./src/pages/_layout.svelte
~/Coding/Personal/svelte-installation-guide-ru main* >                                02:20:22 PM
```

Файлы `_fallback.svelte` и вся папка `example` были удалены за ненадобностью, изначально они у вас будут. Удалять не обязательно.

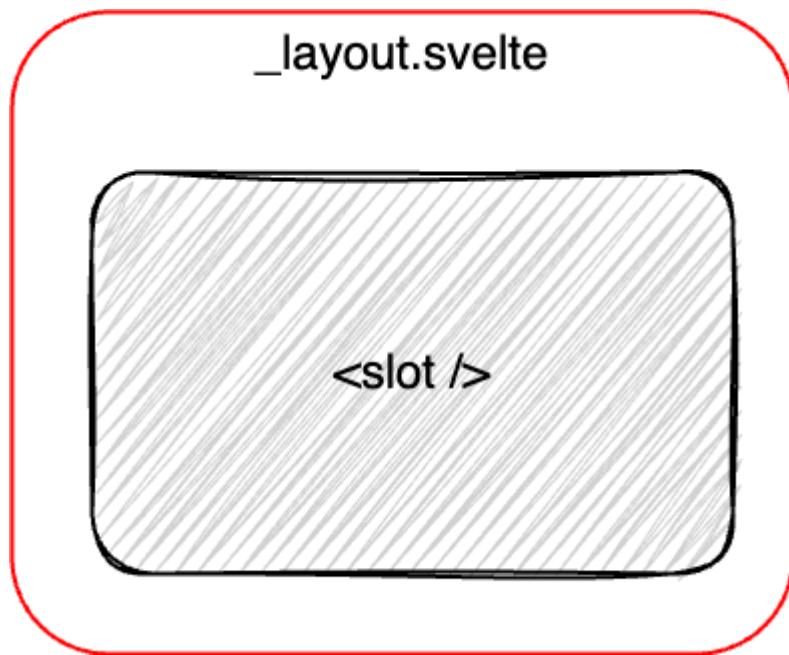
`global.css`, `App.svelte` и `main.js` нам не интересны и их лучше не трогать.

В папке `pages` лежат файлы страниц. Файлы, начинающиеся с `_` - особенные, и их показывать в браузере не будет. `index.svelte` доступен как изначальный файл по данному пути и доступен просто по `http://localhost:5000/`. Если создать файл `/src/pages/test.svelte` - то он будет доступен по адресу: `http://localhost:5000/test`.



Файлы `_layout.svelte`

Это особый файл, который используется как обёртка для всех файлов в текущий папке + рекурсивно для всех подпапок, который позволяет применить к ним общую разметку. Наглядно это можно объяснить так.



Всё в файле layout - будет применено ко всем файлам и дочерним папкам рекурсивно, а вместо тега `<slot>` будет подставлен контент вызываемого файла.

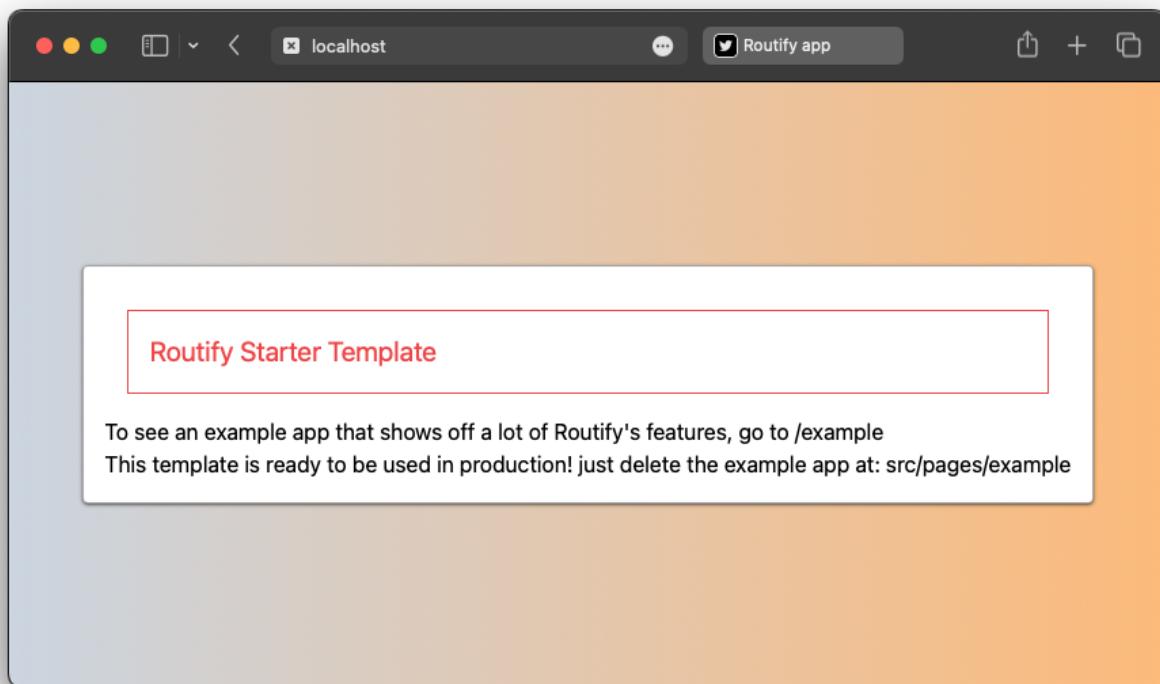
Это очень удобно, когда какой-то элемент необходимо сделать глобальным, например навигацию, по сайту.

Используем layout чтобы придать сайту красивый вид и убедимся этому на страницах / и /test .

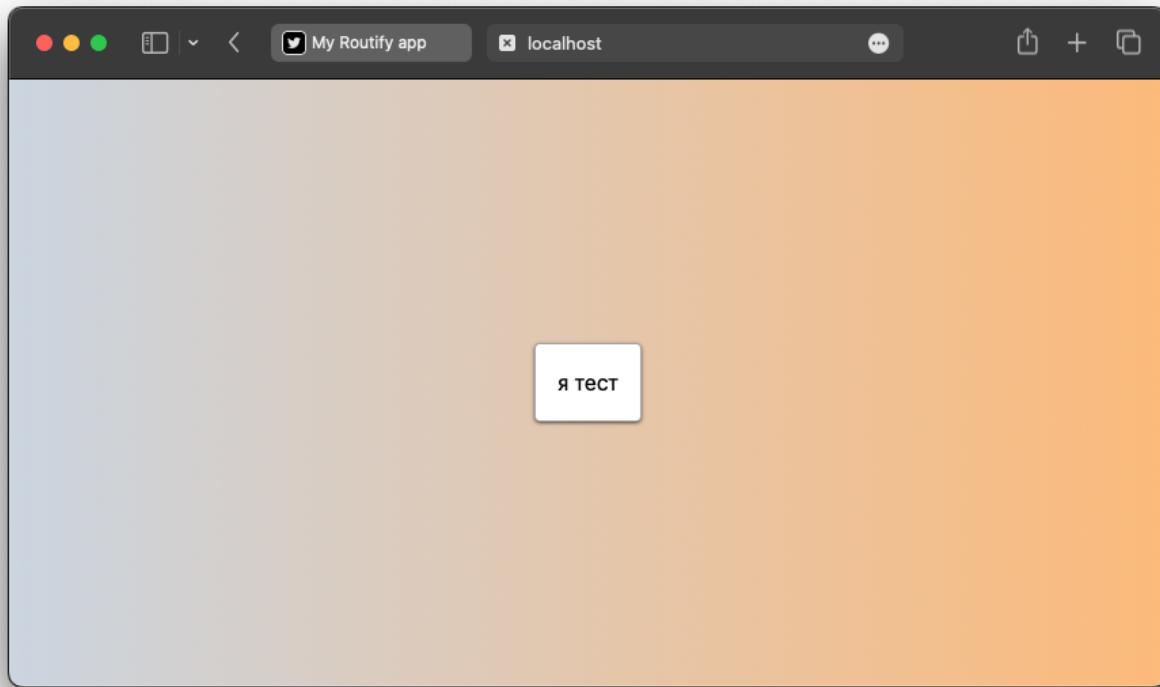
/src/pages/_layout.svelte

```
<div class="..." >
  <div class="..." >
    <slot />
  </div>
</div>
```

Путь /



Путь /test



Удобно, не так ли?

Компоненты

Компоненты позволяют выносить отдельные (обычно повторяющиеся) элементы в файлы, и потом импортировать их при необходимости.

Давайте добавим небольшой бар для навигации по сайту.

Создадим папку `/src/components`, чтобы наши компоненты нельзя было отдельно просмотреть с браузера.

Создадим 2 файла `NavBar.svelte` для самого бара и `NavTab.svelte` для одной вкладки(ссылки).

Сразу импортируем в `_layout.svelte` наш бар.

```
<script>
  import NavBar from '../components/NavBar.svelte'
</script>

<div ... >
  <div ... >
    <NavBar />
  ...
</div>
```

Тут мы вызвали наш компонент без аргументов (в Svelte аргументы называются `props`).

Разберём код `NavBar.svelte` .

```
<script>
  import NavTab from './NavTab.svelte'

  const tabs = [
    {
      name: 'Главная',
      path: '/',
    },
    {
      name: 'Тест',
      path: '/test',
    },
  ]
</script>

<div class="...">
  {#each tabs as tab}
    <NavTab name="{tab.name}" path="{tab.path}" />
  {/each}
</div>
```

В теге `<script>` , так же как и обычном HTML, можно использовать любой клиентский Javascript. В нашем случае - мы объявили константу для хранения элементов меню.

`{#each ...}` - тоже неизвестная до этой поры конструкция. По сути это обычный цикл `foreach` , который говорит: пройдись по всем `tabs` , и положи каждую из ячеек массива поочерёдно в `tab` .

Svelte позволяет использовать `if` , `foreach` и другие подобные конструкции внутри HTML, позволяя сокращать код и добавлять туда логику.

Подробнее о том какие конструкции есть - в официальной [инструкции](#) в главе Logic.

Далее видно, что в компонент `NavTab` было передано 2 параметра - наименование и путь ссылки, но как он их принимает?

`NavTab.svelte`

```
<script>
  import { url } from '@roxi/routify'

  export let name, path
</script>
```

```
<a href="{{$url(path)}}" ...>
  {name}
</a>
```

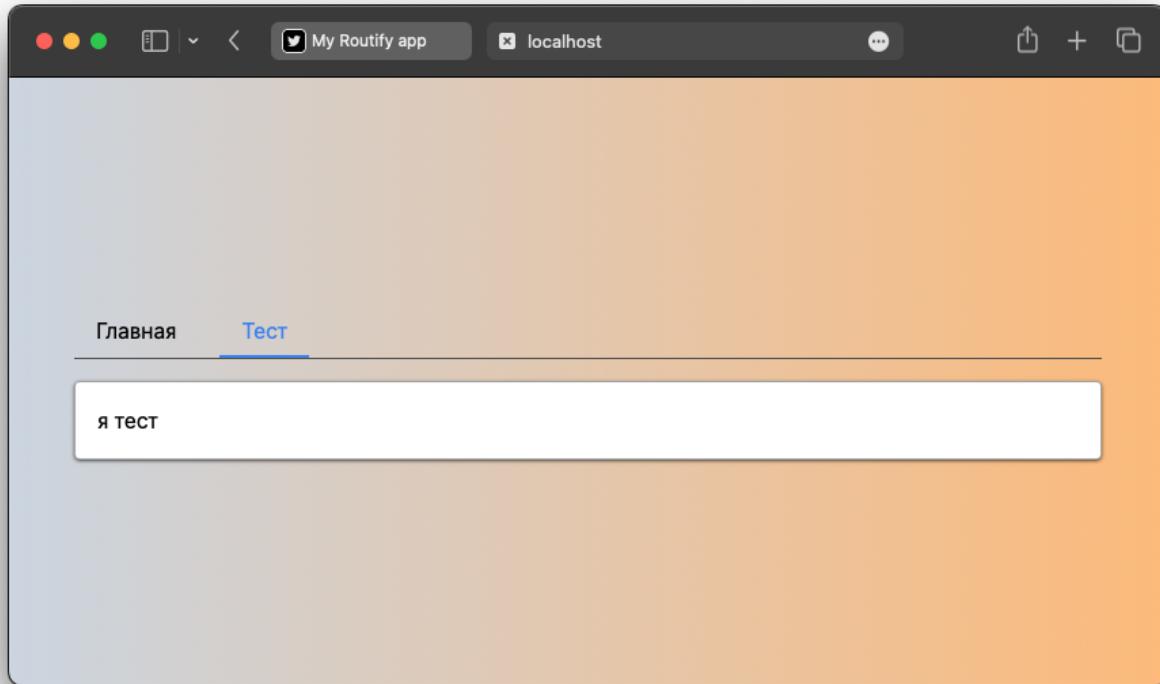
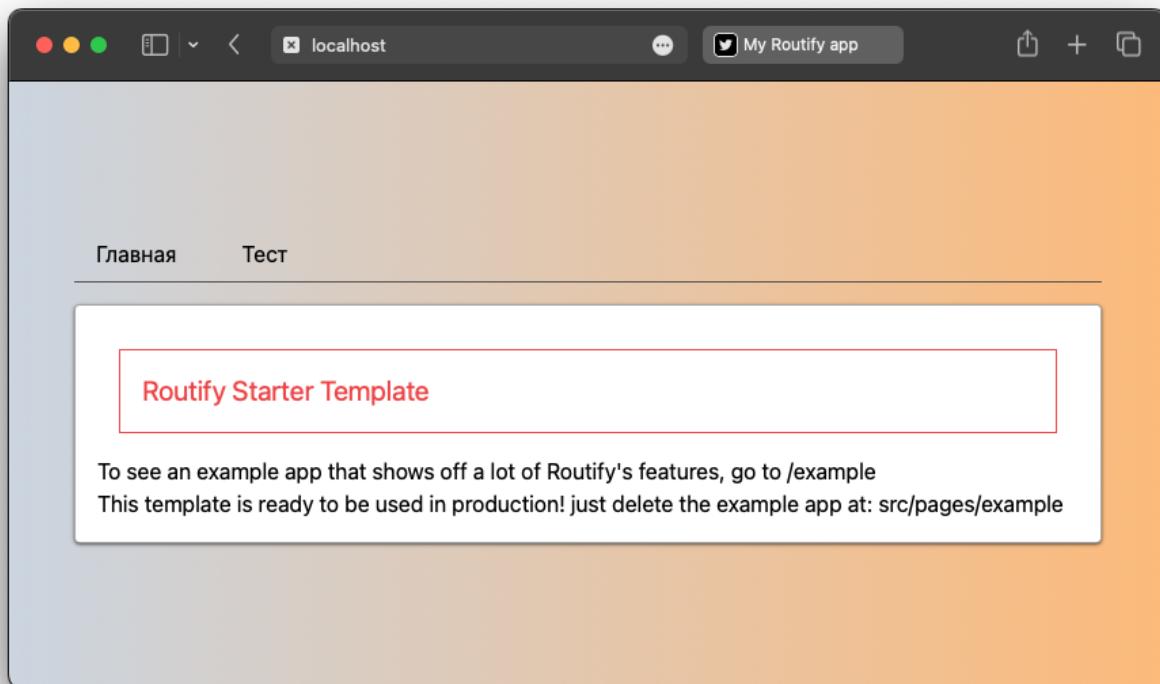
Для начала про `$url`. При использовании Routify ссылки лучше оборачивать в `$url(...)`, так как он позволяет без перезагрузки менять страницы и использовать относительные пути. `$url("../")` - вернёт вас на директорию выше. Стоит отметить, что переходы по ссылкам через `$url` работают как переходы по папкам внутри папки `pages`.

Для вставки JS кода в HTML используйте `{}` скобки. `{name}` при загрузке страницы превратится в название элемента меню и т.п.

Для того чтобы принимать аргументы – необходимо указать, что мы экспортируем переменные, в нашем случае:

```
<script>
  export let name, path
  // ^      ^      ^      ^
  // |      |      |      |
  // |      |  Имена переменных
  // |      |  Объявление переменной(-ых)
  // Ключевое слово для экспортации переменных
</script>
```

В результате получился такой простой бар с ссылками, на основе которого мы познакомились с компонентами. Стоит заметить, что переход между этими страницами не вызывает обновление страницы, как если бы мы сделали `...`.



Отправка данных от ребёнка к родителю

В примерах выше мы рассмотрели как пересыпать данные от родителя к ребёнку через Props дочернего компонента. Вполне возможно, что вам понадобится пересыпать данные от дочернего компонента к родителю.

В таком случае вам стоит обратиться к событиям. [Подробнее тут](#)

Полноценный сценарий есть в примерах, если кратко, то:

```
<!-- child.svelte -->
<script>
  import { createEventDispatcher } from 'svelte';

  const dispatch = createEventDispatcher();

  function sayHello() {
    dispatch('eventName', 'hello');
  }

  sayHello();
</script>

<!-- parent.svelte -->
<script>
  import child from './child.svelte';

  function handleMessage(event) {
    console.log(event.detail)
  }
</script>

<Inner on:eventName={handleMessage} />
```

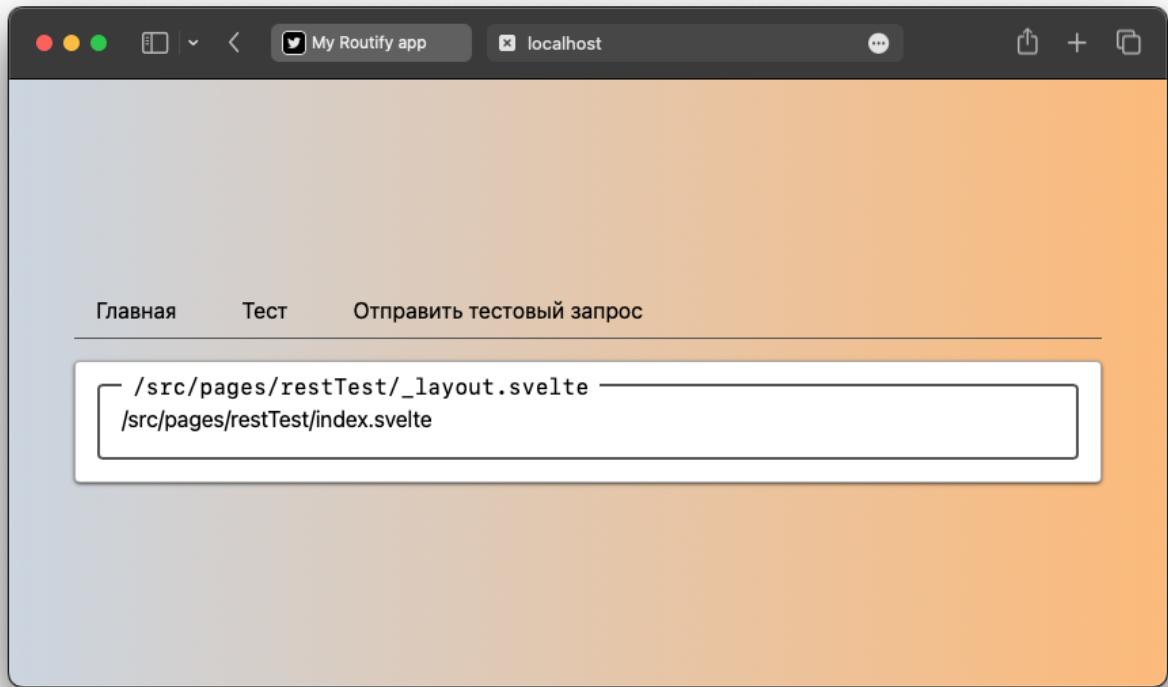
В коде ребёнка необходимо создать `dispatch()`, отправлять данные с помощью данной функции, а на родителе перехватывать эти события через `on:eventName`, где `eventName` - это первый аргумент функции `dispatch`.

Вложенные layout

Создадим новую папку, в которой будет содержаться ещё один `_layout.svelte`, и `index.svelte`, а папку назовём `restTest`. Так же добавим данную папку в нашу навигацию в `NavBar.svelte`.

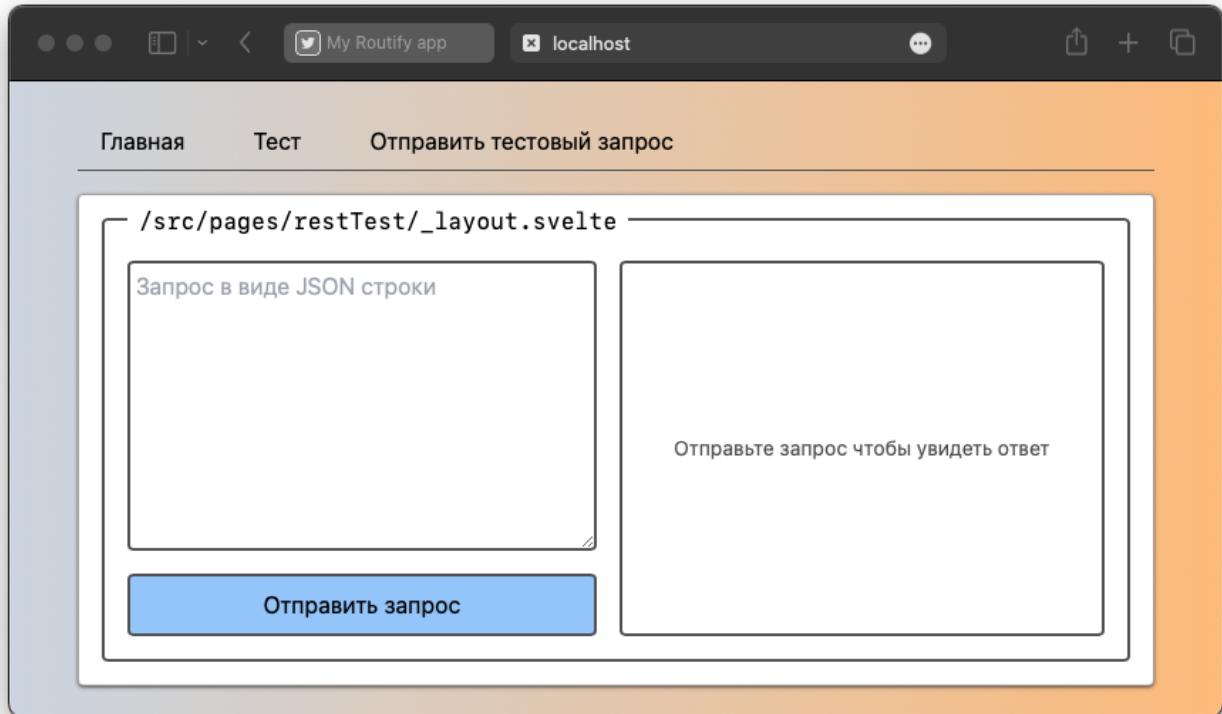
Не забудьте добавить `<slot />` в `/src/pages/restTest/_layout.svelte`. Либо можно просто не создавать его.

```
const tabs = [
  ...,
  {
    name: 'Отправить тестовый запрос',
    path: '/restTest',
  },
]
```



Как видим, `/restTest/index.svelte` обёрнут уже в 2 файла `_layout`, основной, и тот что находится в папке `/src/pages/restTest/`.

Сверстаем небольшой макет. Слева мы можем ввести свой запрос, а справа мы будем видеть ответ на наш запрос.



Директивы `bind`, `on`

Прежде чем отправить запрос, нам необходимо как-то получить информацию, которую ввёл пользователь, а так же совершать действие отправки этой информации по клику кнопки.

Привязка данных к переменным

Для того чтобы привязать вводимую информацию к какой-то переменной, которая будет обновляться каждый раз, как изменяется поле ввода, нам необходимо использовать директиву `bind`.

```
.../restTest/index.svelte

<script>
//...
let value
//...
</script>

<div ... >
<div ... >
<textarea bind:value="{value}" ... ></textarea>
...

```

Теперь, когда мы обращаемся к переменной `value` - в ней всегда будет лежать то, что написано в блоке ввода.

Это работает и в обратную сторону. `value = "123"` установит значение в блоке ввода `123`.

Обработчики событий `on`

Следующей нашей задачей является привязать событие клика к какому-то действию. В Svelte это делается очень просто.

```
.../restTest/index.svelte

<script>
// ...
function handleSendRequest() {
  alert('Я родился')
}
// ...
</script>

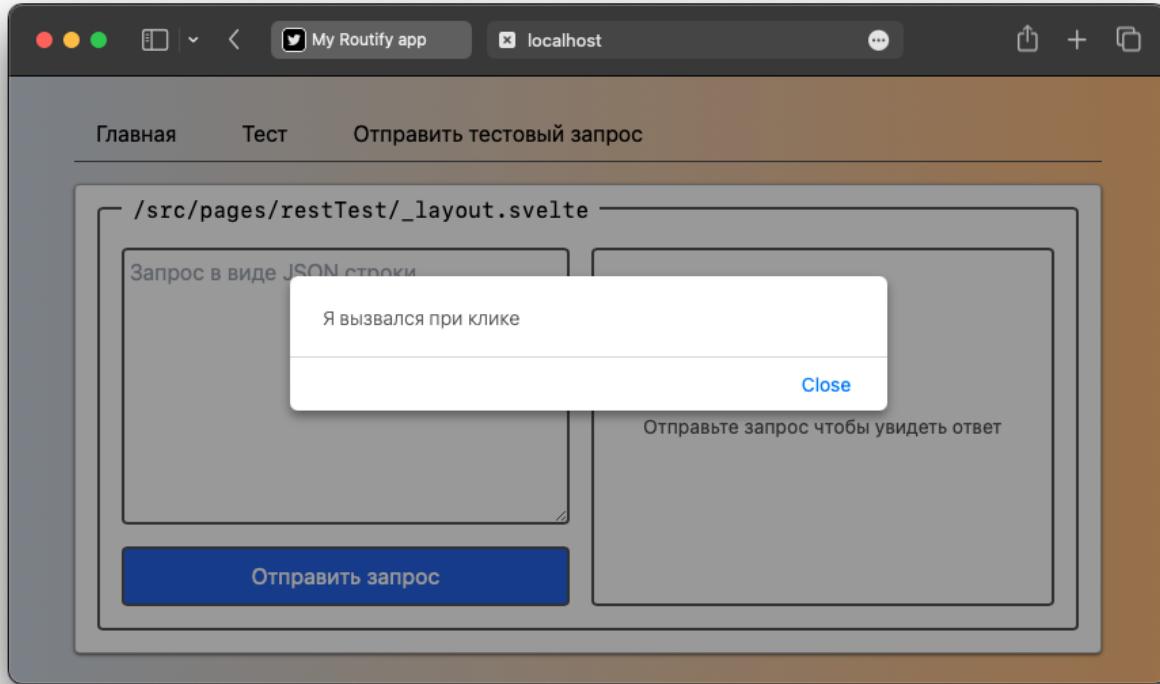
<div ... >
<div ... >
...
<button on:click="{handleSendRequest}" ... >
```

```
    Отправить запрос  
</button>  
...
```

Теперь каждый раз при клике на кнопку будет вызываться функция `handleSendRequest()`.

ОЧЕНЬ ВАЖНО: В `on:click` необходимо писать название функции **БЕЗ** скобок, иначе произойдёт вызов сразу после загрузки страницы.

Теперь при клике у нас срабатывает обработчик.



Отправляем запрос

Изменим функцию при клике, чтобы отправлять запрос.

```
/src/pages/restTest/index.svelte
```

```
let responseJson  
  
async function handleSendRequest() {  
  const res = await fetch('https://echo.zuplo.io/', {  
    method: 'POST', // Метод  
    body: JSON.stringify(value), // Тело запроса. Обязательно в виде строки  
  })  
  const resJson = await res.json()  
  
  // Дополнительные проверки resJson, какая-то логика
```

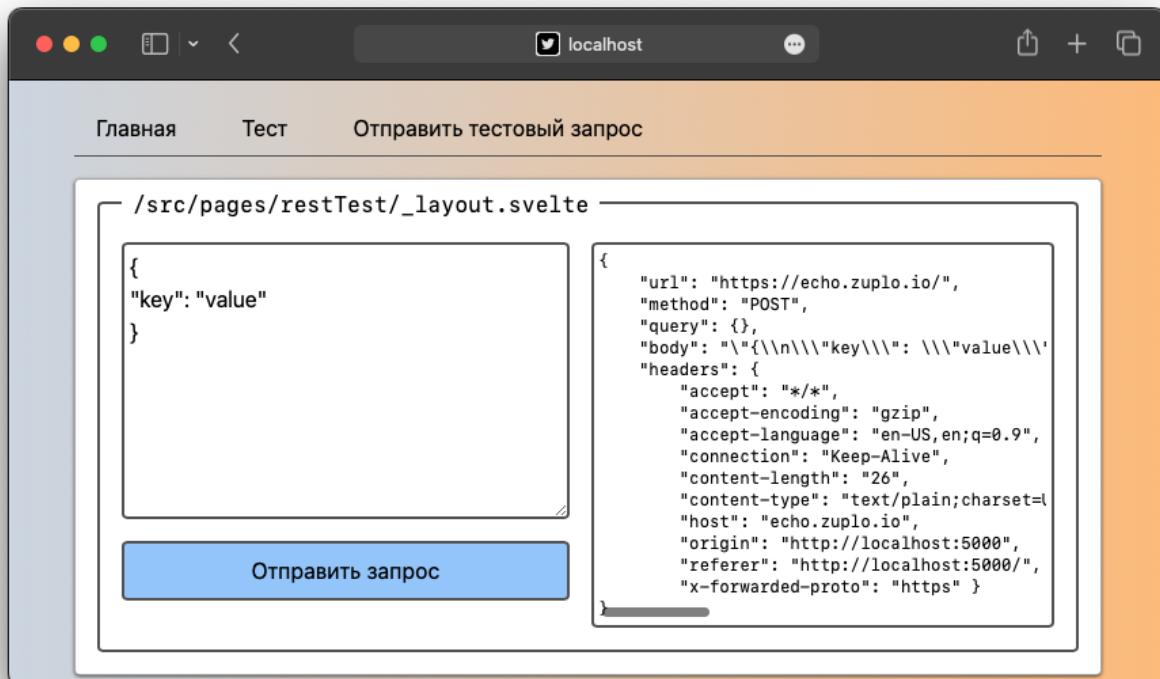
```
    responseJson = resJson
}
```

Добавим вывод ответа в виде отформатированной JSON строки:

```
/src/pages/restTest/index.svelte
```

```
<pre ... >{JSON.stringify(responseJson, null, '\t')}</pre>
```

И при клике получим ответ.



Данный JSON приходит в виде объекта, поэтому вы можете обращаться к его полям. Например, `responseJson.body` вернёт вам ваш запрос, который вы отправили.

CORS

Часто при отправке вы можете столкнуться с ошибками, связанными CORS. Они возникают, если вы пытаетесь отправить запрос с, например, `localhost:5000` на `localhost:8080`. Это исправляется на стороне сервера, со стороны фронтэнда вы вряд ли сможете с этим что-то сделать. (Костыли не в счёт)

Надеюсь, данное руководство было вам полезно.

Полезные ссылки

В основном ссылки на офф. доки, потому что по большинству вопросов там есть ответы, стоит лишь поискать.

- <https://www.google.com/>
- <https://svelte.dev/tutorial/basics> - Официальный гайд по всем возможностям. Есть версия на русском [тут](#), однако обычно там ошибка, если заходить без ВПН.
- <https://tailwindcss.com/> - Все классы Tailwind
- <https://daisyui.com/> - Обёртка поверх Tailwind, в которой есть готовые компоненты. Несколько упрощает жизнь.