

WEB. Работа с NodeJS.

Урок 19. Теория. Шаблонизатор Handlebars.

Server-Side Rendering (SSR).

Server-Side Rendering (SSR) — это процесс рендеринга веб-страниц на сервере, а не на клиенте. В этом подходе HTML-код генерируется сервером и отправляется в браузер пользователя.

Основные шаги SSR включают:

1. Запрос клиента

Браузер отправляет HTTP-запрос на сервер.

2. Генерация HTML

Сервер обрабатывает запрос, генерирует HTML-код и отправляет его обратно клиенту.

3. Отображение в браузере

Браузер получает готовую HTML-страницу и отображает ее пользователю.

Преимущества SSR.

- Быстрое время загрузки: Первичная загрузка страницы быстрее, так как браузер получает готовый HTML.
- SEO: Поисковые системы лучше индексируют серверные рендеринг-страницы.
- Совместимость: Работает даже в браузерах с отключенным JavaScript.
- Недостатки SSR
- Нагрузка на сервер: Сервер должен обрабатывать каждый запрос и генерировать HTML для каждого пользователя.
- Сложность реализации: В некоторых случаях сложнее настроить кэширование и другие оптимизации.

Шаблонизаторы.

Шаблонизаторы — это инструменты, которые позволяют разработчикам генерировать HTML-код, используя шаблоны. Шаблоны содержат статический HTML и динамические маркеры, которые заменяются данными во время рендеринга.

Примеры шаблонизаторов.

- Handlebars
- EJS (Embedded JavaScript)
- Pug
- Mustache

Handlebars.

Handlebars — это мощный и простой в использовании шаблонизатор для JavaScript. Он позволяет создавать семантически чистые шаблоны с минимальной логикой. Handlebars является расширением Mustache и добавляет дополнительные возможности, такие как хелперы и частичные шаблоны.

Шаблоны Handlebars выглядят как обычный HTML с вкраплениями маркеров, которые автоматически экранируют данные для предотвращения XSS-атак.

Поддержка хелперов и частичных шаблонов позволяет легко расширять функциональность.

Пример использования Handlebars.

Установка.

Для использования Handlebars в проекте на Node.js, необходимо установить библиотеку через npm:

```
npm install handlebars
```

Создание шаблона.

Создайте файл template.hbs с содержимым:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>{{title}}</title>
</head>

<body>
  <h1>{{title}}</h1>
  <p>{{body}}</p>
</body>
</html>
```

Рендеринг шаблона.

Используйте Handlebars для рендеринга шаблона с данными:

```
import * as Handlebars from 'handlebars';
import * as fs from 'fs';

// Загрузка шаблона из файла
const templateFile = fs.readFileSync('template.hbs', 'utf8');

// Компиляция шаблона
const template = Handlebars.compile(templateFile);
```

```
// Данные для рендеринга
const data = {
  title: 'Знакомство с Handlebars!',
  body: 'Это очень простой пример использования шаблонизатора.'
};

// Рендеринг HTML
const result = template(data);

console.log(result);
```

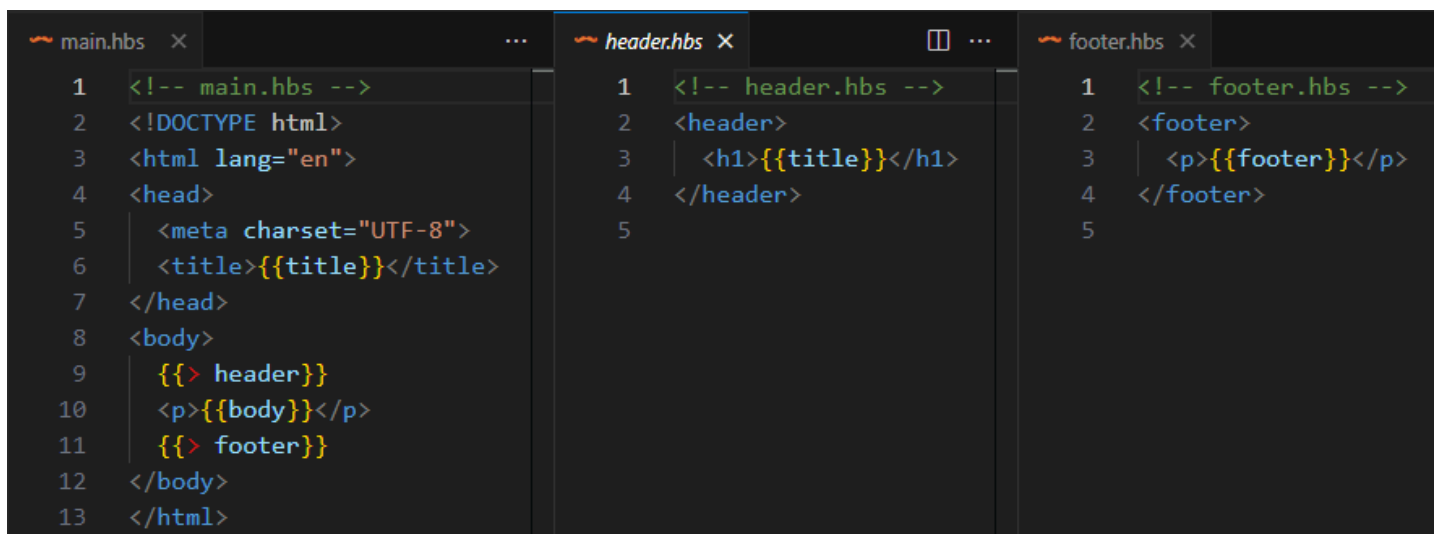
Не забудьте выполнить команду `npm install @types/node`

После сборки и запуска проекта в консоли мы увидим динамически измененную страницу:

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Знакомство с Handlebars!</title>
</head>
<body>
  <h1>Знакомство с Handlebars!</h1>
  <p>Это очень простой пример использования шаблонизатора.</p>
</body>
</html>
```

Частичные шаблоны.

Частичные шаблоны позволяют разбивать большие шаблоны на более мелкие и переиспользуемые части:



```
main.hbs
1 <!-- main.hbs -->
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5   <meta charset="UTF-8">
6   <title>{{title}}</title>
7 </head>
8 <body>
9   {{> header}}
10  <p>{{body}}</p>
11  {{> footer}}
12 </body>
13 </html>

header.hbs
1 <!-- header.hbs -->
2 <header>
3   <h1>{{title}}</h1>
4 </header>
5

footer.hbs
1 <!-- footer.hbs -->
2 <footer>
3   <p>{{footer}}</p>
4 </footer>
5
```

Все эти шаблоны размещены в папке `templates` в папке сборки.

Для рендеринга частичных шаблонов:

```
import * as Handlebars from 'handlebars';
import * as fs from 'fs';

// Регистрация частичных шаблонов
Handlebars.registerPartial('header', fs.readFileSync('Templates/header.hbs',
'utf8'));
Handlebars.registerPartial('footer', fs.readFileSync('Templates/footer.hbs',
'utf8'));

const mainTemplate = fs.readFileSync('Templates/main.hbs', 'utf8');
const template = Handlebars.compile(mainTemplate);

const data = {
  title: 'Шаблонизаторы это круто!',
  body: 'Основной контент страницы.',
  footer: 'Контент подвала.',
};

const result = template(data);

console.log(result);
```

В результате получаем вот такую страницу:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Шаблонизаторы это круто!</title>
</head>
<body>
  <!-- header.hbs -->
  <header>
    <h1>Шаблонизаторы это круто!</h1>
  </header>
  <p>Основной контент страницы.</p>
  <!-- footer.hbs -->
  <footer>
    <p>Контент подвала.</p>
  </footer>
</body>
</html>
```

Рендеринг страниц по запросу.

Начнем с создания привычной для нас файловой структуры:

```
▼ LESSON_15
  > dist
  > src
```

Теперь нужно установить необходимые зависимости:

- Модули express и express-handlebars
- А также типы для работы с модулями ноды и сторонними

Вот команда:

```
npm install express express-handlebars
```

Индикатором успешного выполнения станут создавшиеся 2 файла .json и папка с модулями:

```
▼ LESSON_15
  > dist
  > node_modules
  > src
  {} package-lock.json
  {} package.json
```

Теперь типы:

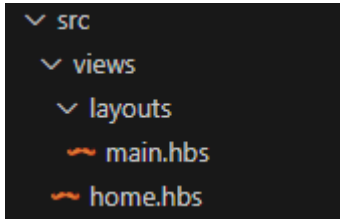
```
npm install @types/node @types/express @types/express-handlebars
```

Создадим файл tsconfig.json и передадим в него следующие настройки:

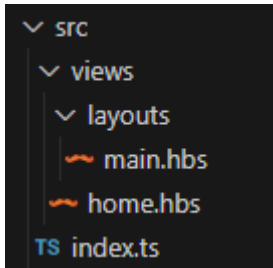
```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true
  },
  "include": ["src"],
  "exclude": ["node_modules"]
}
```

Отлично, все приготовления завершены, можно приступать к разработке.

Создадим папку с моделями представления (шаблонными страницами):



И в самой папке src индексный файл проекта:



В файле main.hbs опишем html шаблон главной страницы. Он будет очень простой:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{{title}}</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 20px;
      background-color: #f4f4f4;
    }
    h1 {
      color: #333;
    }
  </style>
</head>
<body>
  {{{body}}}
</body>
</html>
```

Как видите, тело (body) нашей страницы будет динамическим.

В файле home.hbs определим динамическое тело:

```
<h1>{{title}}</h1>
<p>{{message}}</p>
```

Теперь перейдем к файлу index.ts.

Сначала импортируем необходимые модули:

```
import express from 'express';
import { engine } from 'express-handlebars';
import path from 'path';
```

Основное - это импортировать функцию engine.

Ниже создадим express приложение и опишем порт:

```
const app = express();
const port = 3000;
```

Далее опишем настройку движка handlebars:

```
// Настройка Handlebars
app.engine('hbs', engine({
  extname: 'hbs',
  defaultLayout: 'main',
  layoutsDir: path.join(__dirname, '../src/views/layouts')
}));
app.set('view engine', 'hbs');
app.set('views', path.join(__dirname, '../src/views'));
```

Разберем эту конструкцию, так как она типовая.

1. `app.engine('hbs', engine({...}))`

Эта строка настраивает Handlebars как движок шаблонов для Express.

- `'hbs'`: Это название, под которым движок будет зарегистрирован. Вы можете использовать это имя позже, чтобы сослаться на движок (например, при настройке типа шаблонов).
- `engine({...})`: Функция engine из пакета express-handlebars принимает объект конфигурации.

Объектом конфигурации являются настройки, где ключ - конкретная настройка, а в качестве значения вы передаете настройки для шаблонизации.

Параметры конфигурации:

- `extname: 'hbs'`: Расширение файлов шаблонов, которое будет использоваться. Здесь указано, что файлы шаблонов будут иметь расширение `.hbs`.
- `defaultLayout: 'main'`: Указывает, какой файл макета будет использоваться по умолчанию. В данном случае это файл `main.hbs`, который будет использоваться как основной макет для всех представлений.
- `layoutsDir: path.join(__dirname, '../src/views/layouts')`: Путь к директории, где хранятся файлы макетов. В данном случае используется `path.join(__dirname, '../src/views/layouts')`, что указывает на директорию `layouts` в папке `src/views`.

2. `app.set('view engine', 'hbs')`

Эта строка устанавливает `'hbs'` в качестве движка представлений по умолчанию. Это значит, что когда вы вызываете `res.render('template')` (о нем ниже), Express будет использовать Handlebars для рендеринга файлов с расширением `.hbs`.

3. `app.set('views', path.join(__dirname, '../src/views'))`

Эта строка указывает Express, где искать файлы представлений.

Здесь используется `path.join` для создания абсолютного пути к директории `views`, которая находится в папке `src`. `__dirname` представляет собой путь к текущей директории, где находится файл `app.ts`.

Таким образом, `path.join(__dirname, '../src/views')` создает путь к директории `views`, которая находится на уровне выше по сравнению с текущей директорией и внутри папки `src`.

Идём далее. Дальнейшие конструкции будут вам более знакомы:

```
// Маршрут корневой страницы
app.get('/', (req, res) => {
  res.render('home', {
    title: 'Домашняя страница',
    message: 'Добро пожаловать на наш вебсайт!'
  });
});

app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```

Из нового для вас, это функция ответа `res.render('шаблон рендеринга', {...});`
Функция `res.render` используется для рендеринга представления (шаблона) и отправки результирующего HTML-кода в ответ на HTTP-запрос.

Первый параметр: `'home'`

Это имя шаблона (без расширения), который будет рендериться. Express будет искать файл с именем `home.hbs` (или с любым другим расширением, настроенным в движке представлений) в директории, указанной в `app.set('views', ...)`.

Второй параметр: `{ title: 'Домашняя страница', message: 'Добро пожаловать на наш вебсайт!' }`

Это объект, содержащий данные, которые будут переданы в шаблон. Эти данные будут доступны внутри шаблона для рендеринга динамического контента.

Вот и все, осталось собрать.