9. REST

→ REST : Representational State Transfer

→ REST is an architectural style that defines a set of constraints to be used for creating web services.

→ CRUD operations

GET	retrieves resources
POST	submits new data to the server
PUT	updates existing data
PATCH	updates existing data partially
DELETE	removes data

e.g.→ We will make a page (simple Quora Posts) which has some posts and we will use CRUD operations on this page.

* Creating RESTful APIs

GET	/posts	index	to get data for all posts
POST	/posts	create	to add new post
GET	/posts/:id	view	to get one post (using id)
PATCH	/posts/:id	update	to update specific post
DELETE	/posts/:id	destroy	to delete specific post

In demo/ REST_CLASS Terminal

```
- HP@DP MINGW64 ~/Desktop/ demo / REST_CLASS
  $ npm init -y
  :

- $ npm install express
  :

- $ npm install ejs
  :

- $ touch index.js
- $ code index.js
```

In demo/ REST_CLASS / index.js

```
- const express = require("express");
  const app = express();

  const port = 8080;

app. listen (port, () => {

    console.log(`App is listening on port ${port}`);
});
```

In Terminal

- $ nodemon index.js :

```
App is listening on port 8080
```

* Implement: GET /posts :

GET /posts to get data for all posts
(index route)

In demo/ REST_CLASS/ index.js:

```
- let posts = [
        {
            username: "parth",
            content: "I love coding":
        }, { ... }, { ... }
];

app.use(express.urlencoded( { extended: true } ) );
app.use(express.static("public"));

app.set("view engine", "ejs");

app.get("/posts", (req, res) ⇒ {

    res.render( "index.ejs", { posts });
});
```

In demo/REST_CLASS/public/ style.css

```
- body {    background-color: aqua;    }

h1 {    color: maroon;    }

h3, h4 {
    margin: 0;
    padding: 0.5rem 1rem;
}

.user {
    font-style: italic;
    color: blue;
}

.post {
    background-color: wheat;
    margin: 1rem 0;
}
```

In demo/ REST_CLASS/ views/ index.ejs

```
- <!DOCTYPE html>
  :
    <link rel="stylesheet" href="/style.css".>
</head>
```

```
<body>

  <h1>  Quora  Posts  </h1>

  <% for ( const post of posts ) {     %>

       <div  class="post" >
            <h3  class = "user">
                 @ <%=  post.username   %>
            </h3>
            <h4 class = "content">
                 <%=  post.content  %>
            </h4>
       </div>

  <%  }   %>

</body>
</html>
```

* Implement: POST /posts

POST /posts to add new post
(create (new) route)

2 routes:
→ Serve the form GET /posts/new
→ Add new post POST /posts

In views/ index.ejs

```
- <a href="http:// localhost: 8080/posts/new">
    Create a new post
  </a>
```

In views/ new.ejs

```
- <! DOCTYPE html>
  :
     <title> Create a New Post </title>
  </head>
  <body>

    <form action="/posts" method="post">

       <Input type="text"
              name="username"
              placeholder="enter username">
```

```
        <br>
        <br>

        <textarea  name="content"
                   placeholder="enter your content">
        </textarea>

        <br>
        <br>

        <button type="submit"> Submit Post </button>

    </form>

</body>
</html>
```

In REST_CLASS/ index.js

```
- app.get("/posts/ new", (req ,res) => {

    res.render("new.ejs");
});

app.post("/posts", (req, res) => {

    let { username , content } = req.body;
    posts.push( {username , content } );

    res.redirect("/posts");
});          ↑ built-in get request of express
```

* Implement: GET /posts/:id

GET /posts/:id to get one post (using id)
(show route)

In views/ index.ejs

```
- <div class="post">
      ⋮
      <a href="http://localhost:8080/posts/<%= post.id %>">
      show Details
      </a>
  </div>
```

In views/ show.ejs

```
- <!DOCTYPE html>
  ⋮
      <title> Post in Detail </title>
      <link rel="stylesheet" href="/style.css">
  </head>
  <body>

    <h2> Here is your post in Details </h2>
    <p> Post id: <%= post.id %> </p>
  ⋮
    <br>
```

```
<div class="post">
    <h3 class="user">
      @ <%= post.username %>
    </h3>
    <h4 class="content">
        <%= post.content %>
    </h4>
  </div>

  <br>

  <a href="http://localhost:8080/posts"> All posts</a>
</body>
</html>
```

In <u>REST_CLASS/ index.js</u>

- we will add id: "1a" in first object of array, for second object id: "2b", ... so on.

```
app.get("/posts/:id", (req, res) => {

    let { id } = req.params;
    let post = posts.find((element) => id === element.id);

    res.render("show.ejs", {post});
});
```

* Create id for post

→ When we create new post, we are not setting-up new id, so for that we will write(do) as below:

→ UUID Package : Universally Unique Identifier

In <u>Terminal</u>

```
- $ npm install uuid
```

In <u>index.js</u>

```
- const { v4: uuidv4 } = require("uuid");
```

- Replace value of **id** of array items with uuidv4() :

↖ creates random alphanumeric string with characte
(ab4ch - jml4 - abc - xyz ...)

```
- app.post("/posts", (req, res) => {

    let { username, content } = req.body;

    posts.push({ id: uuidv4(), username, content });

    res.redirect("/posts");

});
```

* Implement: PATCH /posts/:id

PATCH /posts/:id to update specific post
(update route)
(edit)

2 routes:
→ serve update form GET /posts/:id/edit
→ Update post PATCH /posts/:id

In views/ index.ejs

```
- <div class=" post ">
  :
  <a href="http://localhost:8080/post/<%= post.id %>/edit">
    Edit
  </a>
</div>
```

In index.js

```
- app.get("/posts/:id/edit", (req, res) =>{
      let { id } = req.params;
      let post = posts.find((item) => id === item.id );
      res.render("edit.js", {post});
  });
```

* method - override package

→ In HTML form element has only two method built-in get & post, so if we want to use PATCH, PUT, DELETE, ... ; we have to install a package.

In Terminal

```
- $ npm install method-override
  :
```

In views/ edit.ejs

```
- <!DOCTYPE html>

      <title> Edit Post </title>
  </head>
  </body>

      <h2> Edit Your Post </h2>
      <p> Post id : <%= post.id %> </p>
      <p> Post Username: <%= post. id %> </p>

      <form method="post"
            action="/posts/<%=post.id %>?_method=PATCH">
          <textarea name="content">
            <%= post.content %>
          </textarea>
          <button type="submit"> Submit</button>
      </form>
  </body>
  </html>
```

In index.js

```
- const methodOverride = require("method-override");

  app.use(methodOverride("_method"));

  app.patch("/posts/:id", (req, res) => {

      let { id } = req.params;
      let newContent = req.body.content;
      let post = posts.find((item) => id===item.id);
      post.content = newContent;

      res.redirect("/posts");
  });
```

reference → to the original value(object), NOT a new copy

* Implement: /posts/:id DELETE

DELETE /posts/:id to delete specific post
(destroy route)

In views/ index.ejs

```
- <div class="post">
    :
  . <form method="post"
        action="/posts/<%= post.id %>?_method=DELETE">

        <button> Delete Post </button>
    </form>
</div>
```

In index.js

```
- app.delete("/posts/:id", (req, res) => {

      let {id} = req.params;

      posts = posts.filter((item) => id !== item.id);

      res.redirect("/posts");
  });
```