Cross Skill

Target

Creating Crud API’s with PostGreSql using Prisma Orm and NestJS

Content

- Connection setup
- Environment
- User Model
- Create
- Retrieve
- Update
- Delete

Connection Setup With Prisma

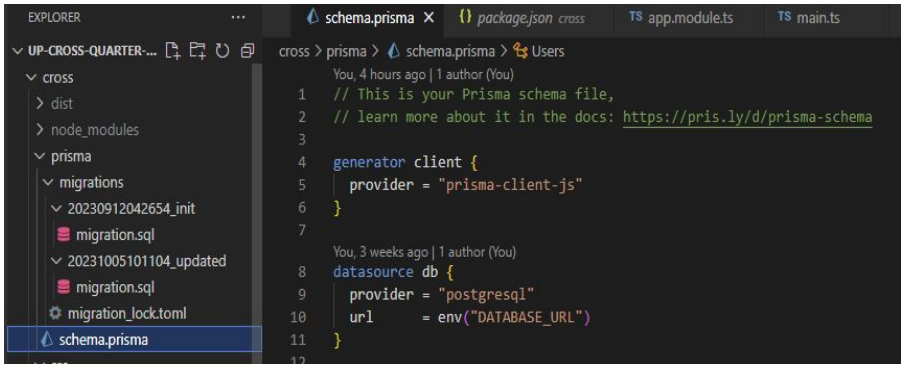
Add prisma and prisma client with the following command::

```
PS D:\up-cross-quarter-1-24\cross> npm install prisma @prisma/client
```

Then initiate prisma with the following:

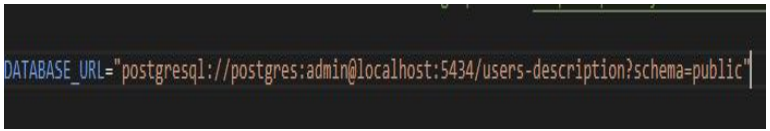
```
PS D:\up-cross-quarter-1-24\cross> npx prisma init
```

After the command a folder is created named “prisma” and inside it there’s a file, schema.prisma



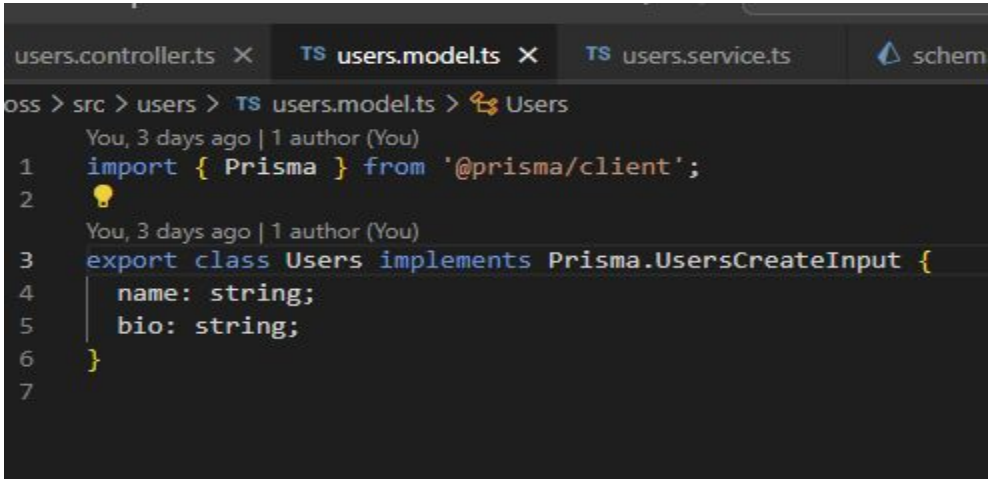
Environment

Add database url in the .env file:



Add the model in the schema.prisma file:

Model


users.controller.ts
users.model.ts
users.service.ts
schem
oss > src > users > users.model.ts > Users
You, 3 days ago | 1 author (You)
1 import { Prisma } from '@prisma/client';
2
You, 3 days ago | 1 author (You)
3 export class Users implements Prisma.UsersCreateInput {
4 name: string;
5 bio: string;
6 }
7

Prisma service and User module

Prisma service:

```
import { INestApplication, Injectable, OnModuleInit } from '@nestjs/common';
import { PrismaClient, Prisma } from '@prisma/client';

You, 3 weeks ago | 1 author (You)
@Injectable()
export class PrismaService
  extends PrismaClient<Prisma.PrismaClientOptions, 'beforeExit'>
  implements OnModuleInit
{
  async onModuleInit() {
    await this.$connect();
  }

  async enableShutdownHooks(app: INestApplication) {
    this.$on('beforeExit', async () => {
      await app.close();
    });
  }
}
```

User module:

```
You, 3 weeks ago | 1 author (You)
import { Module } from '@nestjs/common';
import { UsersController } from './users.controller';
import { UsersService } from './users.service';
import { PrismaService } from 'src/prisma.service';

You, 3 weeks ago | 1 author (You)
@Module({
  controllers: [UsersController],
  providers: [UsersService, PrismaService],
})
export class UsersModule {}
```

Create

	id [PK] integer	name character varying (100)	bio character varying (255)
1	1	sss	asaa
2	3	sss	ssaseee
3	4	safa safa	Test test

Create (Contd.)

```
async createUser(data: Users): Promise<Users> {
  return this.prisma.users.create({
    data,
  });
}
```

```
@Post()
async postUser(@Body() postData: Users): Promise<Users> {
  return this.usersService.createUser(postData);
}
```

Retrieve (All)

User.service:

```
async getAllUser(): Promise<Users[]> {
  return this.prisma.users.findMany();
}
```

Users.controller:

```
@Get()
async getAllUsers(        You, now • Uncommitted changes
  @Req() request: Request,
  @Res() response: Response,
): Promise<any> {
  const result = await this.usersService.getAllUser();
  return response.status(200).json({
    status: 'Ok!',
    message: 'Successfully fetch data!',
    result: result,
  });
}
```

Result:


```
{
    "status": "Ok!",
    "message": "Successfully fetch data!",
    "result": [
        {
            "id": 3,
            "name": "sss",
            "bio": "ssaseee"
        },
        {
            "id": 1,
            "name": "sss",
            "bio": "asaa"
        },
        {
            "id": 4,
            "name": "safa safa",
            "bio": "Test test"
        },
        {
            "id": 5,
            "name": "safa5",
            "bio": "Test test"
        }
    ]
}
```

Retrieve Single element

User.service:

```
async getUser(id: number): Promise<Users> {
  return this.prisma.users.findUnique({ where: { id: Number(id) } });
}       You, 3 weeks ago • Cross done 70%
```

User.controller:

```
@Get(':id')
async getUsers(@Param('id') id: number): Promise<Users | null> {
  return this.usersService.getUser(id);
}
```

URL:

Result:


```
{
    "id": 1,
    "name": "sss",
    "bio": "asaa"
}
```

Update

User.service:

```
async updateUser(id: number, data: Users): Promise<Users> {
  return this.prisma.users.update({
    where: { id: Number(id) },
    data,
  });
}       You, 3 weeks ago • Cross done 70%
```

User.controller:

```
@Put(':id')
async updateUsers(
  @Param('id') id: number,
  @Body() data: Users,
): Promise<Users> {
  return this.usersService.updateUser(id, data);
}
```

URL:

Body:

```
{
    "name":"sss",
    "bio":"admin"
}
```

Result:

```
{
    "id": 1,
    "name": "sss",
    "bio": "admin"
}
```

Delete

User.service:

```
async deleteUser(id: number): Promise<Users> {
  return this.prisma.users.delete({
    where: { id: Number(id) },
  });
}
```

User.controller:

```
@Delete(':id')
async deleteUsers(@Param('id') id: number): Promise<Users> {
  return this.usersService.deleteUser(id);
}
```

Url:

Result:

	id [PK] integer	name character varying (100)	bio character varying (255)
1	1	sss	admin
2	3	sss	ssaseee
3	4	safa safa	Test test

Up Skill

Session: July- September, 2023

Md. Rashedul Hasan Safa

10482

Target

Basic of React-redux with redux/toolkit

What was done?

- Basic Page setup
- Store setup
- Create a postSlice file
- Create
- Retrieve
- Update
- Delete

Basic setup

First run the cross app

Downloaded skeleton code from:

https://github.com/trickjsprogram/toolkit-crud-api

The package.json:

Setup store

```
You, 1 second ago | 1 author (You)
import React from "react";
import ReactDOM from "react-dom";
import "./index.css";
import App from "./App";
import "antd/dist/antd.css";
import { BrowserRouter } from "react-router-dom";
import { Provider } from "react-redux";
import { configureStore } from "@reduxjs/toolkit";
// import PostReducer from "./feature/postSlice";

const store = configureStore({
  reducer: {
    // app: PostReducer,
  },
});

ReactDOM.render(
  <React.StrictMode>
    <Provider store={store}>
      <BrowserRouter>
        <App />
      </BrowserRouter>
    </Provider>
  </React.StrictMode>,
  document.getElementById("root")
);
```

Create a postSlice

```
const postSlice = createSlice({
  name: "post",
  initialState: {
    post: [],
    loading: false,
    error: null,
    edit: false,
    bio: "",
  },
```

Retrieve

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";

export const getPost = createAsyncThunk("post/getPost ", async ({ id }) => {
  return fetch(`http://localhost:3000/api/users/${id}`).then((res) =>
    res.json()
  );
});

extraReducers: {
    [getPost.pending]: (state, action) => {
      state.loading = true;
    },
    [getPost.fulfilled]: (state, action) => {
      state.loading = false;
      state.post = [action.payload];
    },
    [getPost.rejected]: (state, action) => {
      state.loading = false;
      state.error = action.payload;
    },
```

Retrieve (Contd.)

```
import { useSelector, useDispatch } from "react-redux";
import { getPost, deletePost, setEdit, updatePost } from "../feature/postSlice";

import { Button, Card, Input, Space } from "antd";

const UserPost = ({ history }) => {
  const dispatch = useDispatch();

  const [id, setId] = useState();
  const [bodyText, setBodyText] = useState("");
  const { loading, post, edit, bio } = useSelector((state) => ({
    ...state.app,
  }));
  const onChangeInput = (e) => {
    setId(e.target.value);
  };

  const fetchUserPost = () => {
    if (!id) {
      window.alert("Please enter id");
    } else {
      dispatch(getPost({ id }));
      setId("");
    }
  };
```

Create

In PostSlice:

```
export const createPost = createAsyncThunk(
  "post/createPost ",
  async ({ values }) => {
    return fetch(`http://localhost:3000/api/users`, {
      method: "POST",
      headers: {
        Accept: "application/json",
        "Content-type": "application/json",
      },
      body: JSON.stringify({
        name: values.name,
        bio: values.bio,
      }),
    }).then((res) => res.json());
  }
);
```

```
//  create post
[createPost.pending]: (state, action) => {
  state.loading = true;
},
[createPost.fulfilled]: (state, action) => {
  state.loading = false;       You, 5 days ago
  state.post = [action.payload];
},
[createPost.rejected]: (state, action) => {
  state.loading = false;
  state.error = action.payload;
},
```

```
const dispatch = useDispatch();
const { loading, post } = useSelector((state) => ({ ...state.app }));

const [showPost, setShowPost] = useState(false);

const handleSubmit = (e) => {
  e.preventDefault();
  dispatch(createPost({ values }));
```

Create (Contd.)

The UI:

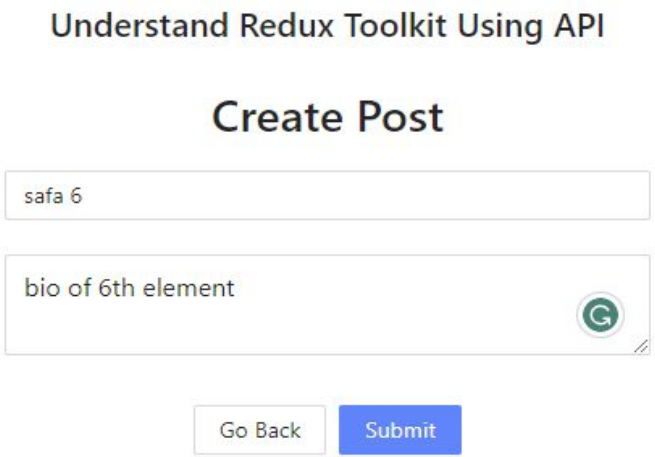


Table before submission:

	id [PK] integer	name character varying (100)	bio character varying (255)
1	1	sss	admin
2	3	sss	ssaseee
3	4	safa safa	Test test

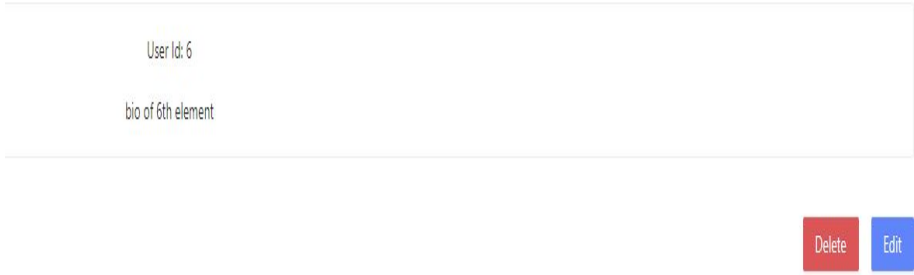
Table after submission:

	id [PK] integer	name character varying (100)	bio character varying (255)
1	1	sss	admin
2	3	sss	ssaseee
3	4	safa safa	Test test
4	6	safa 6	bio of 6th element

Update

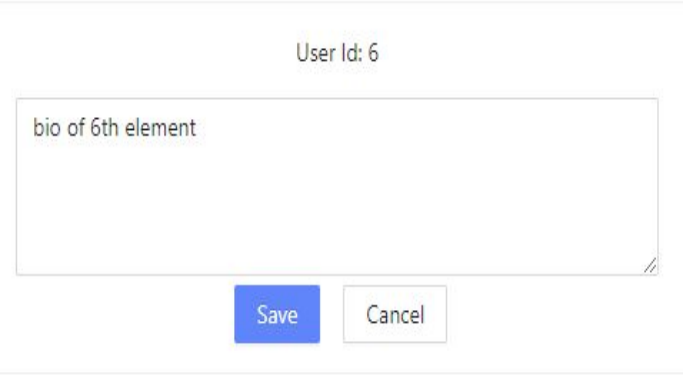
```
reducers: {
  setEdit: (state, action) => {
    state.edit = action.payload.edit;
    state.bio = action.payload.bio;
  },
},
```

```
      dispatch(setEdit({ edit: false, bio: "" }));
    }}
  >
    Save
  </Button>
  <Button
    onClick={() =>
      dispatch(setEdit({ edit: false, bio: "" }))
    }
  >
    Cancel
  </Button>
```



Update(Contd.)

After Clicking edit

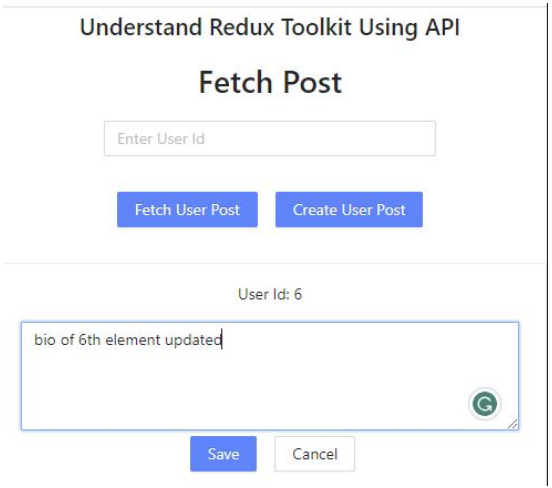



```
export const updatePost = createAsyncThunk(
  "post/updatePost ",
  async ({ id, bio, name }) => {
    return fetch(`http://localhost:3000/api/users/${id}`, {
      method: "PUT",
      headers: {
        Accept: "application/json",
        "Content-type": "application/json",
      },
      body: JSON.stringify({
        name,
        bio,
      }),
    }).then((res) => res.json());
  }
);
```

```
// update post
[updatePost.pending]: (state, action) => {
  state.loading = true;
},
[updatePost.fulfilled]: (state, action) => {
  state.loading = false;
  state.post = [action.payload];
},
[updatePost.rejected]: (state, action) => {
  state.loading = false;
  state.error = action.payload;
},
```

Update (Contd.)

```
onClick={() => {
  dispatch(
    updatePost({
      id: post[0].id,
      bio: bodyText,
      name: post[0].name,
    })
  );
```

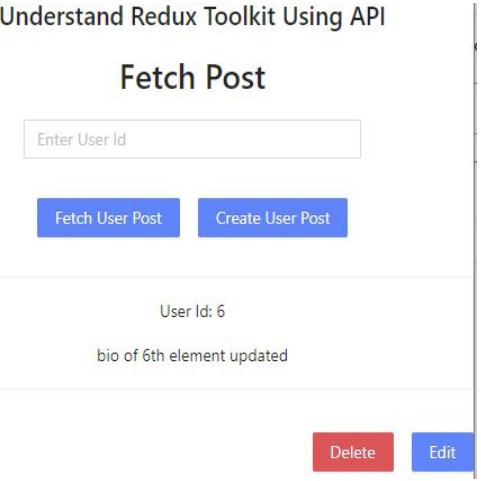


User Id: 6

bio of 6th element updated

	id [PK] integer	name character varying (100)	bio character varying (255)
1	1	sss	admin
2	3	sss	ssaseee
3	4	safa safa	Test test
4	6	safa 6	bio of 6th element updated

Delete




```
export const deletePost = createAsyncThunk(
  "post/deletePost ",
  async ({ id }) => {
    return fetch(`http://localhost:3000/api/users/${id}`, {
      method: "DELETE",
    }).then((res) => res.json());
  }
);
```

```
// for delete objects
[deletePost.pending]: (state, action) => {
  state.loading = true;
},
[deletePost.fulfilled]: (state, action) => {
  state.loading = false;
  state.post = {};
},
[deletePost.rejected]: (state, action) => {
  state.loading = true;
  state.error = action.payload;
},
```

	id [PK] integer	name character varying (100)	bio character varying (255)
1	1	sss	admin
2	3	sss	ssaseee
3	4	safa safa	Test test

THANK YOU