3. REST API

Goal

The goal of this lesson is to use your freshly gained knowled
implement some routes of a REST API using Express.

Setup

You can continue working in the same `prisma-workshop` p
However, the starter for this lesson is located in the `rest-a`
cloned.

Before you switch to that branch, you need to commit the c
simplicity, you can use the `stash` command to do this:

```
git stash
```

After you ran this command, you can switch to the `rest-ap`
`migrations` directory and `dev.db` file:

```
git checkout rest-api rm -rf prisma/migrations
```

Next, wipe your npm dependencies and re-install them to a
`package.json`:

```
rm -rf node_modules npm install
```

The data model you will use here is very similar to the one tl
`Post` model got extended with a few additional fields:

```
model User { id Int @id @default(autoincrement()
String? posts Post[] } model Post { id Int @id @
createdAt DateTime @default(now()) updatedAt Dat
content String? published Boolean @default(false
author User? @relation(fields: [authorId], refer
```

Since you are starting over with your Prisma setup, you have
tables. Run the following command to do this:

```
npx prisma migrate dev --name init
```

Finally, you can seed the database with some sample data t
`prisma/seed.ts` file. You can execute this seed script with

```
npx prisma db seed --preview-feature
```

That's it, you're ready for your tasks now!

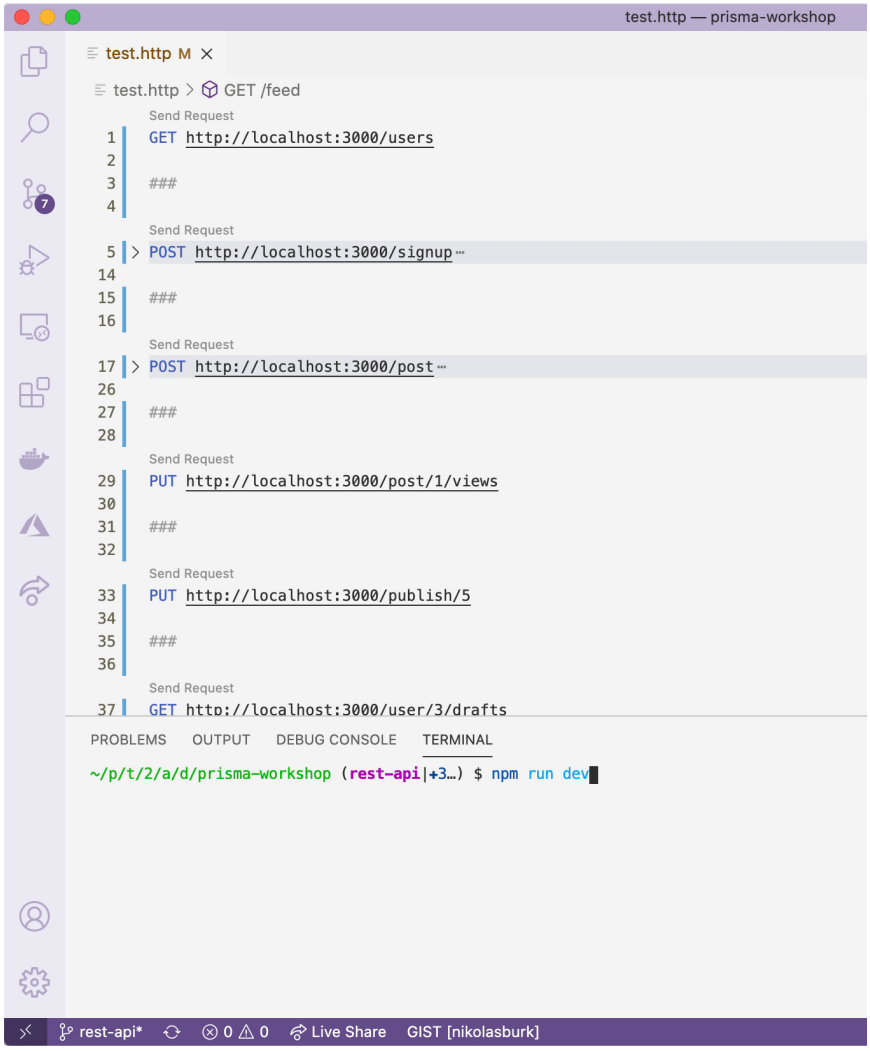
Tasks

You can find the tasks for this lesson inside the `src/index.`
is to insert the right Prisma Client queries for each REST AP

Note that this is in *not* a lesson in API design and you should
your API operations in a real-world application.

If you're using VS Code, you can install the REST Client exte
using the provided HTTP calls in `test.http`.

▼ View a quick demo of the VS Code REST Client extensio



`GET /users`

Fetches all users.

▼ Solution

```
app.get("/users", async (req, res) => { const
prisma.user.findMany() res.json(result) });
```

`POST /signup`

Creates a new user.

▼ Solution

```
app.post(`/signup`, async (req, res) => { co
req.body; const result = await prisma.user.c
}) res.json(result) });
```

`POST /post`

Creates a new post.

▼ Solution

```
app.post(`/post`, async (req, res) => { cons
authorEmail } = req.body; const result = awa
{ title, content, author: { connect: { email
res.json(result) });
```

`PUT /post/:id/views`

Increments the views of a post by 1.

▼ Solution

```
app.put("/post/:id/views", async (req, res)
req.params; const result = await prisma.post
Number(id), }, data: { viewCount: { incremen
res.json(result); });
```

`PUT /publish/:id`

Publishes a post.

▼ Solution

```
app.put("/publish/:id", async (req, res) =>
const result = await prisma.post.update({ wh
{ published: true, }, }); res.json(result);
```

GET /user/:id/drafts

Fetches the unpublished posts of a specific user.

▼ Solution

```
app.get("/user/:id/drafts", async (req, res)
req.params; const result = await prisma.user
Number(id) }, }).posts({ where: { published:
});
```

GET /post/:id

Fetches a post by its ID.

▼ Solution

```
app.get(`/post/:id`, async (req, res) => { c
const result = await prisma.post.findUnique(
}); res.json(result); });
```

GET /feed?searchString=<searchString>&ski

Fetches all published posts and optionally paginates and/or search string appears in either title or content.

▼ Solution

```
app.get("/feed", async (req, res) => { const
= req.query; const or = searchString ? { OR:
searchString as string } }, { content: { con
} }, ], } : {} const result = await prisma.p
published: true, ...or }, skip: Number(skip)
Number(take) || undefined, }); res.json(resu
```