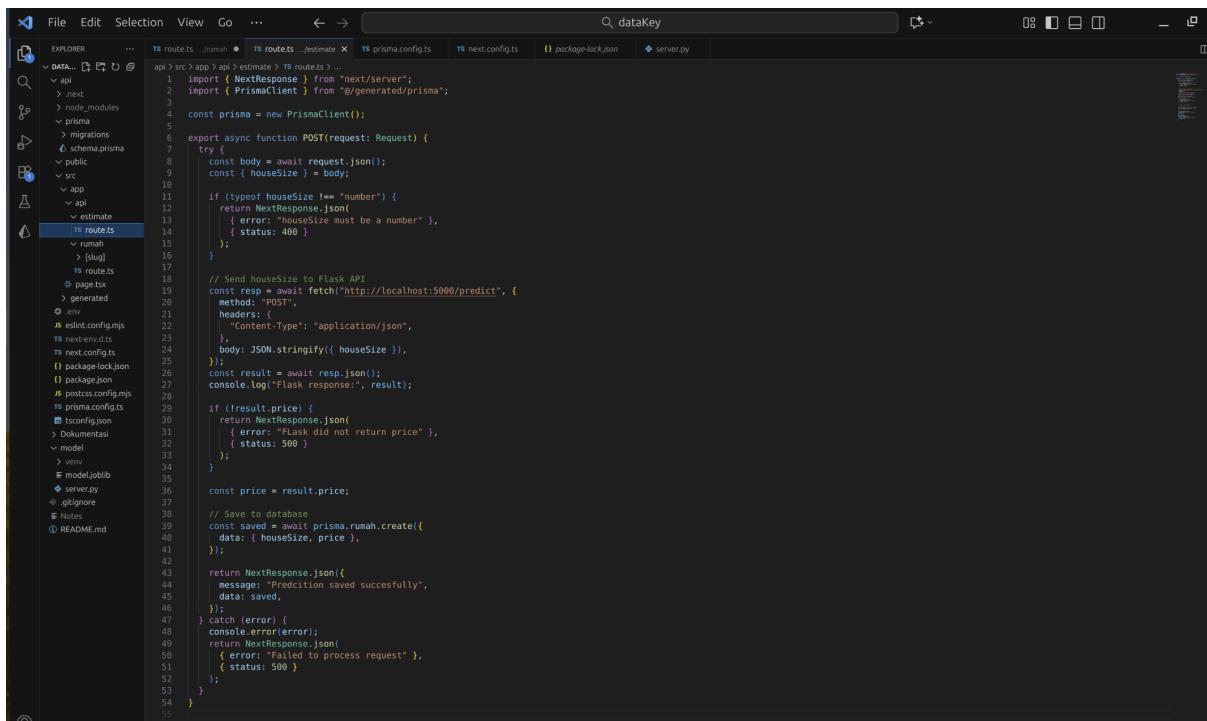


- Buat logic sebenarnya dalam di dalam `src/app/api/estimate`, untuk mengirim `houseSize` sebelumnya melanjutkannya ke python (dalam hal ini Flask), untuk kemudian diproses dan dikembalikan value berupa `price`.

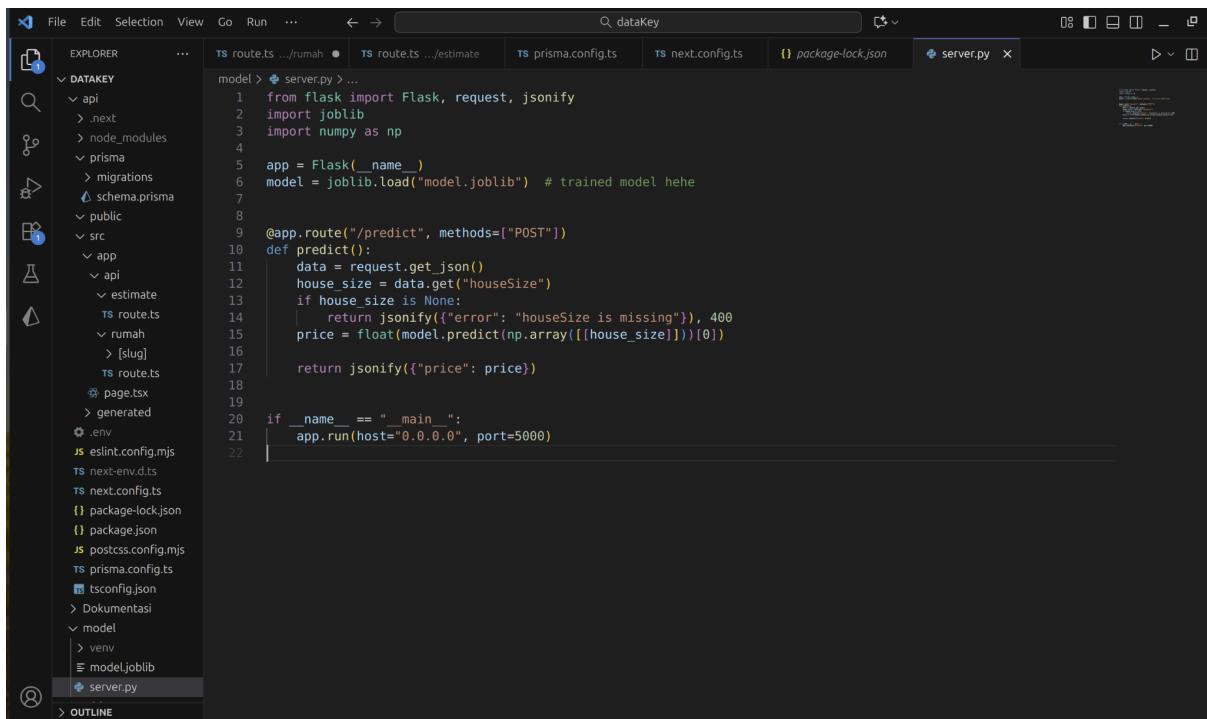


```

File Edit Selection View Go ... ⏪ ⏩ 🔍 dataKey
EXPLORER ... TS route.ts .../rumah ● TS route.ts .../estimate X TS prisma.config.ts TS next.config.ts 🔍 package-lock.json 🔍 server.py
api > src > app > api > estimate > TS route.ts > ...
1 import { NextResponse } from "next/server";
2 import { PrismaClient } from "@generated/prisma";
3
4 const prisma = new PrismaClient();
5
6 export async function POST(request: Request) {
7   try {
8     const body = await request.json();
9     const { houseSize } = body;
10
11     if (typeof houseSize !== "number") {
12       return NextResponse.json(
13         { error: "housesize must be a number" },
14         { status: 400 }
15       );
16     }
17
18     // Send houseSize to Flask API
19     const resp = await fetch("http://localhost:5000/predict", {
20       method: "POST",
21       headers: {
22         "Content-Type": "application/json",
23       },
24       body: JSON.stringify({ houseSize }),
25     });
26     const result = await resp.json();
27     console.log("Flask response:", result);
28
29     if (!result.price) {
30       return NextResponse.json(
31         { error: "Flask did not return price" },
32         { status: 500 }
33       );
34     }
35
36     const price = result.price;
37
38     // Save to database
39     const saved = await prisma.rumah.create({
40       data: { houseSize, price },
41     });
42
43     return NextResponse.json(
44       { message: "Prediction saved successfully" },
45       { data: saved },
46     );
47   } catch (error) {
48     console.error(error);
49     return NextResponse.json(
50       { error: "Failed to process request" },
51       { status: 500 }
52     );
53   }
54 }

```

- Gotcha!, inside `model/server.py`, you will find a logic code where's Flask serve value `houseSize`, to process it with the model we have trained, and then return it into `next.js`.



```

File Edit Selection View Go Run ... ⏪ ⏩ 🔍 dataKey
EXPLORER ... TS route.ts .../rumah ● TS route.ts .../estimate X TS prisma.config.ts TS next.config.ts 🔍 package-lock.json 🔍 server.py
model > 🔍 server.py > ...
1 from flask import Flask, request, jsonify
2 import joblib
3 import numpy as np
4
5 app = Flask(__name__)
6 model = joblib.load("model.joblib") # trained model hehe
7
8 @app.route("/predict", methods=["POST"])
9 def predict():
10   data = request.get_json()
11   house_size = data.get("houseSize")
12   if house_size is None:
13     return jsonify({"error": "houseSize is missing"}), 400
14   price = float(model.predict(np.array([[house_size]]))[0])
15
16   return jsonify({"price": price})
17
18
19
20 if __name__ == "__main__":
21   app.run(host="0.0.0.0", port=5000)
22

```

test it...!!!

The screenshot shows the Apidog API testing interface. On the left sidebar, there are sections for APIs, Tests, Docs, History, and Settings. The main area displays a POST request to `http://localhost:3000/api/estimate`. The request body is set to `JSON` with the value:

```
1 {
2     "houseSize": 125
3 }
```

The response details show a status of `200`, `890 ms`, and `88 B`. The response body is:

```
1 {
2     "message": "Prediction saved successfully",
3     "data": {
4         "id": 17,
5         "houseSize": 125,
6         "price": 1312
7     }
8 }
```

At the bottom, there are tabs for Design-first Mode and Request-first Mode, along with navigation links for Online, Request Proxy, Cookies, Trash, and Help & support.