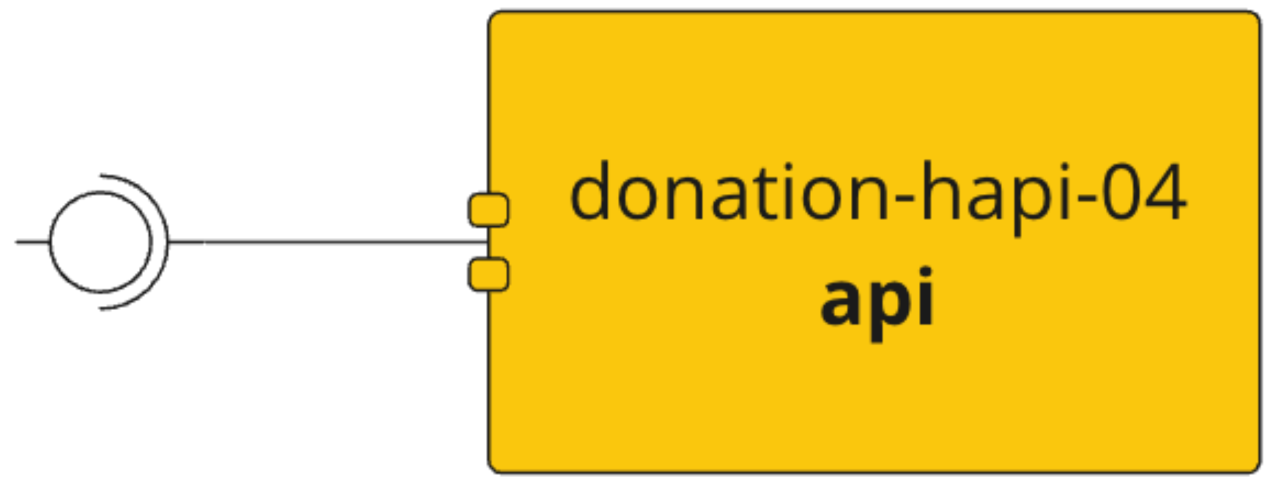
Hapi/JWT API

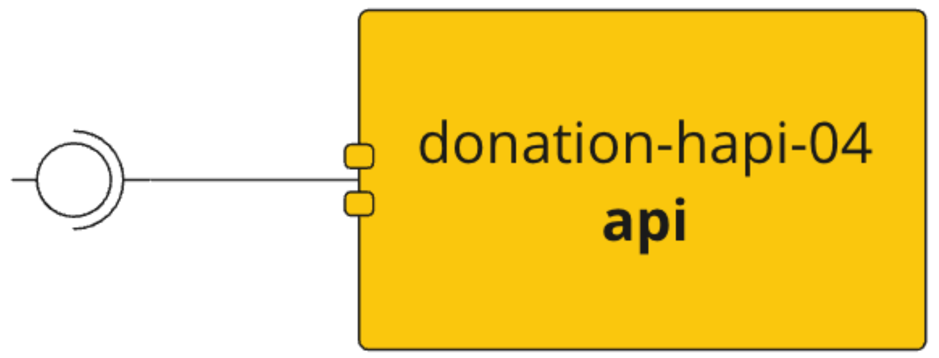
HAPI Authentication API using JWT

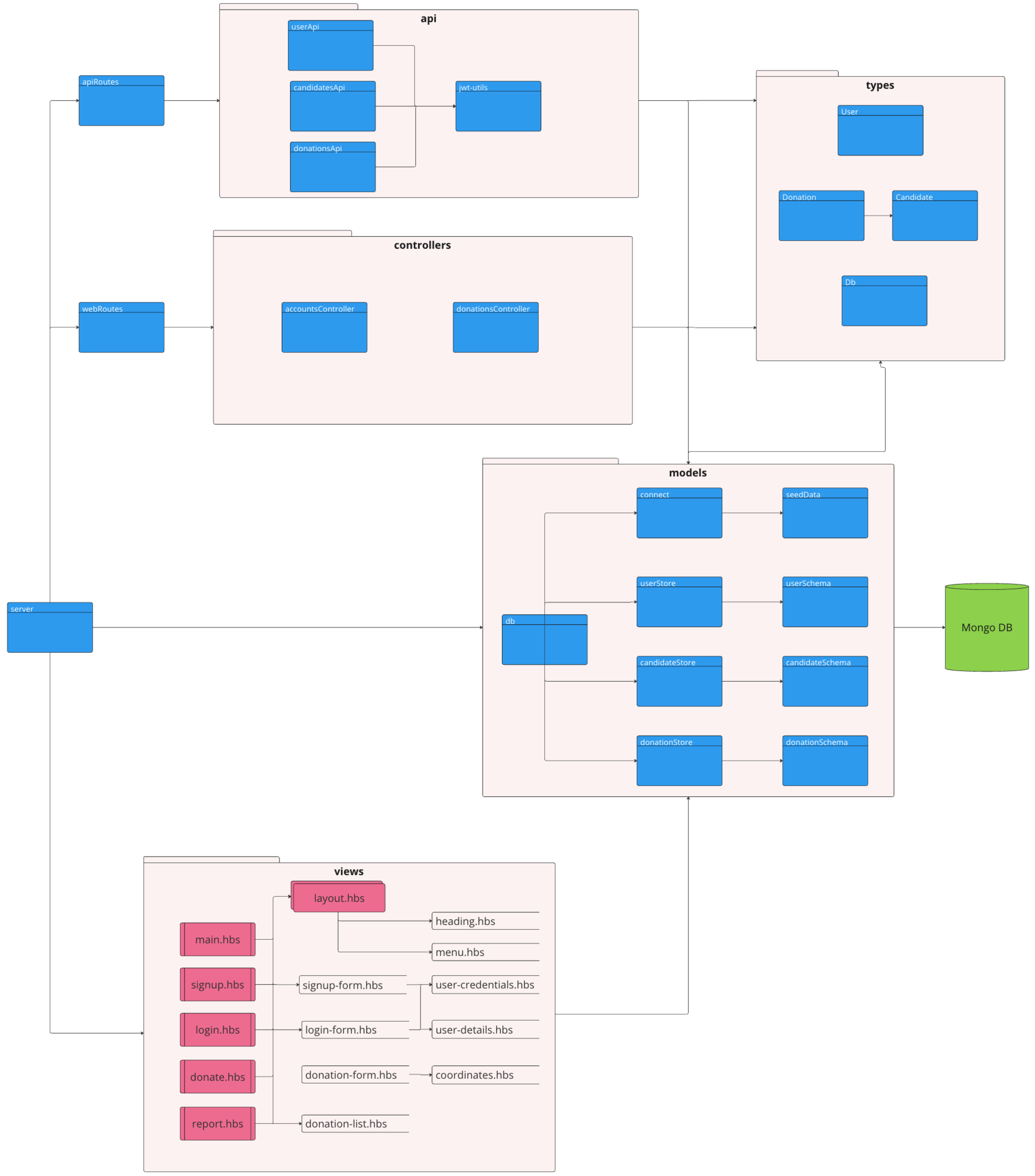
donation-hapi-0.4.0

- Extend donation-hapi to include
  - Users API
  - JWT Security Strategy
  - Donations API
  - Candidates API

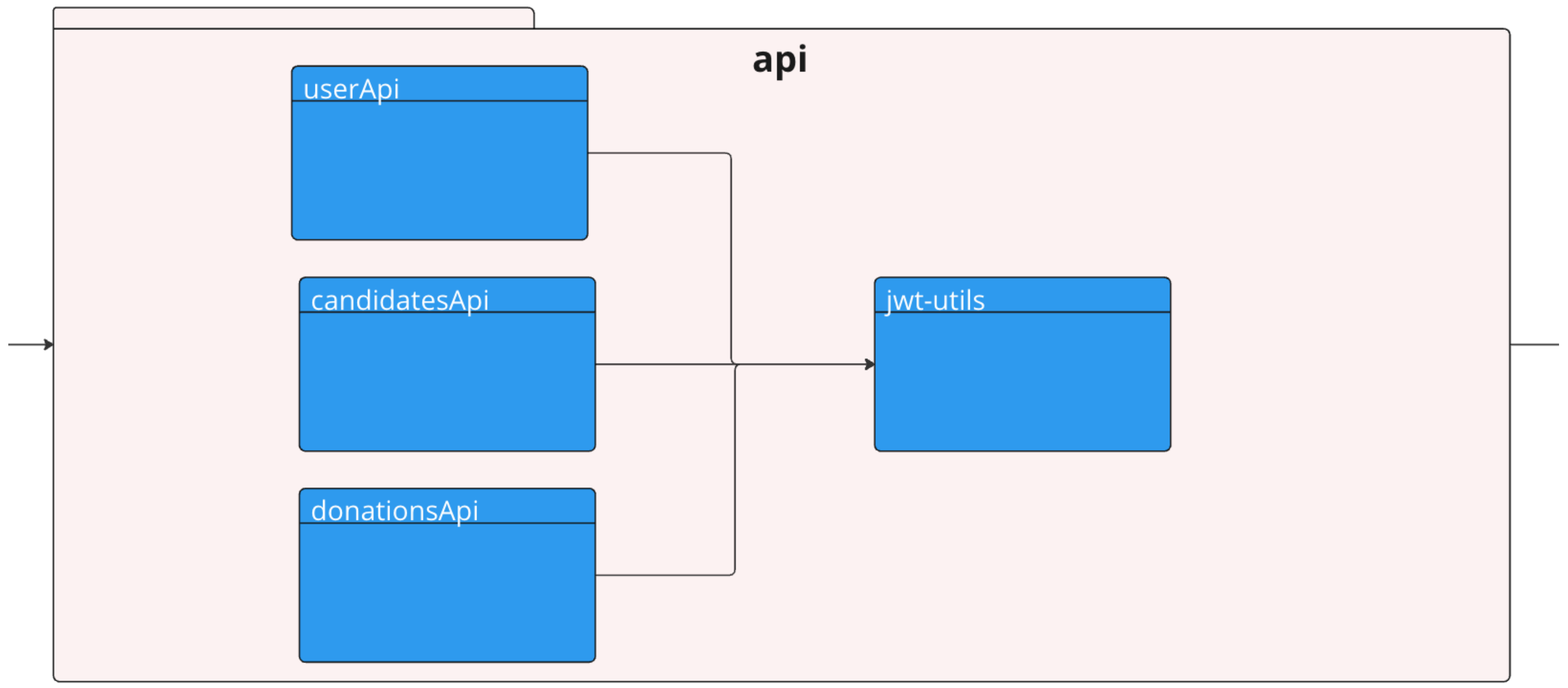


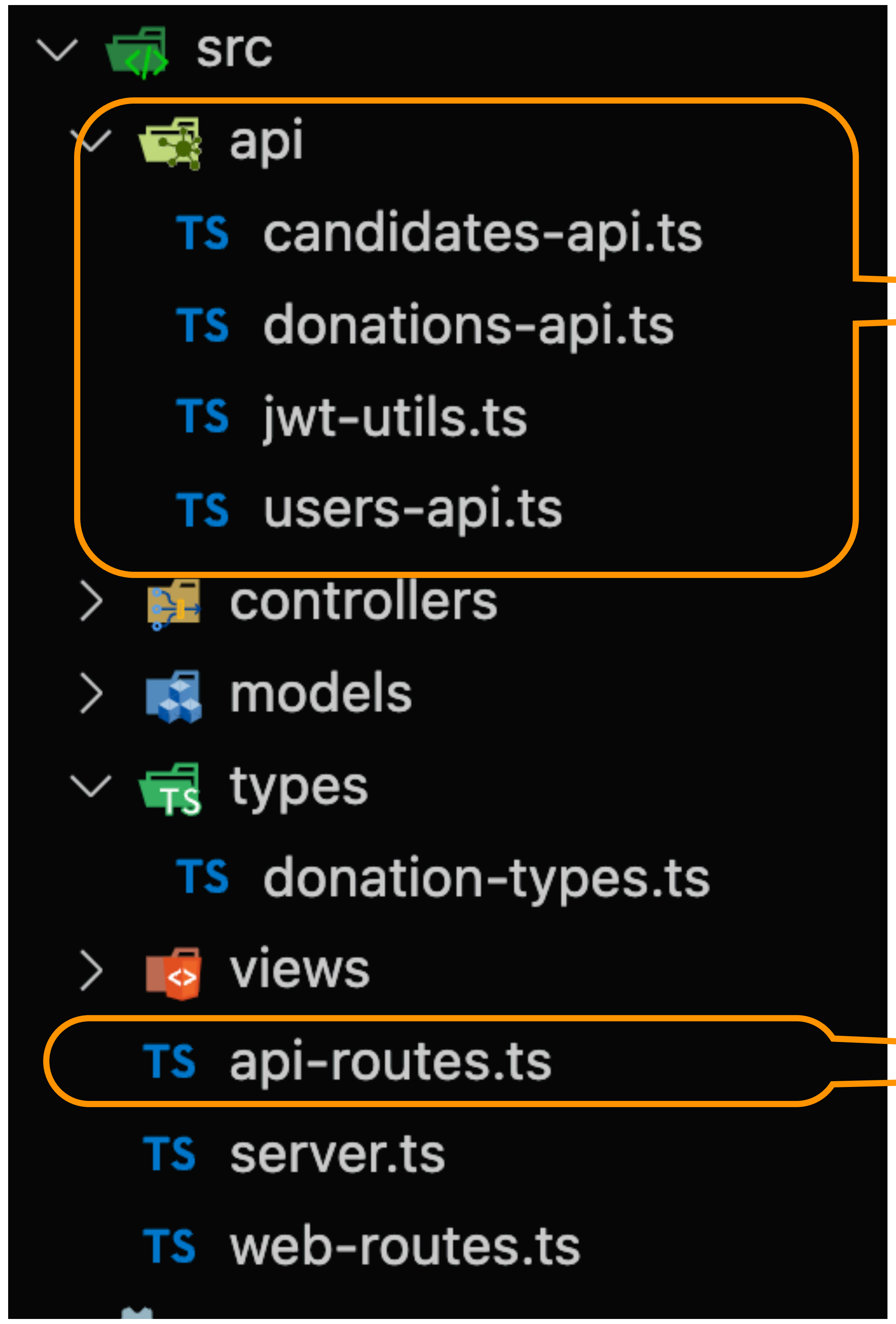
Architecture





API Subsystem



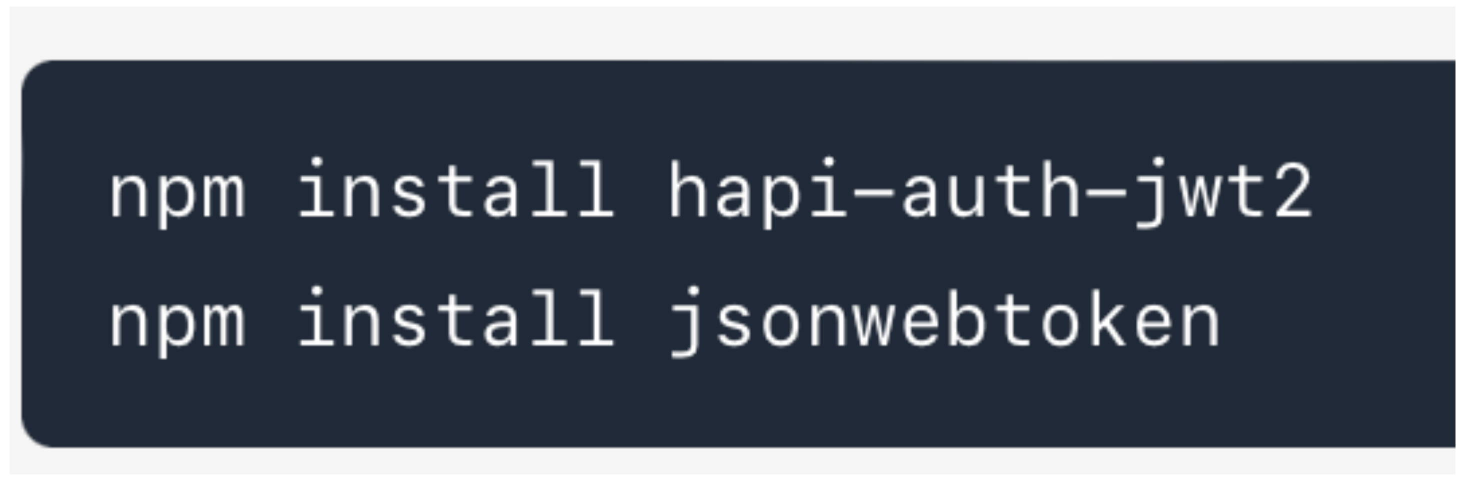


```
src
  api
    candidates-api.ts
    donations-api.ts
    jwt-utils.ts
    users-api.ts
  controllers
  models
  types
    donation-types.ts
  views
  api-routes.ts
  server.ts
  web-routes.ts
```

- JWT Security Strategy
- Users API
- Donations API
- Candidates API

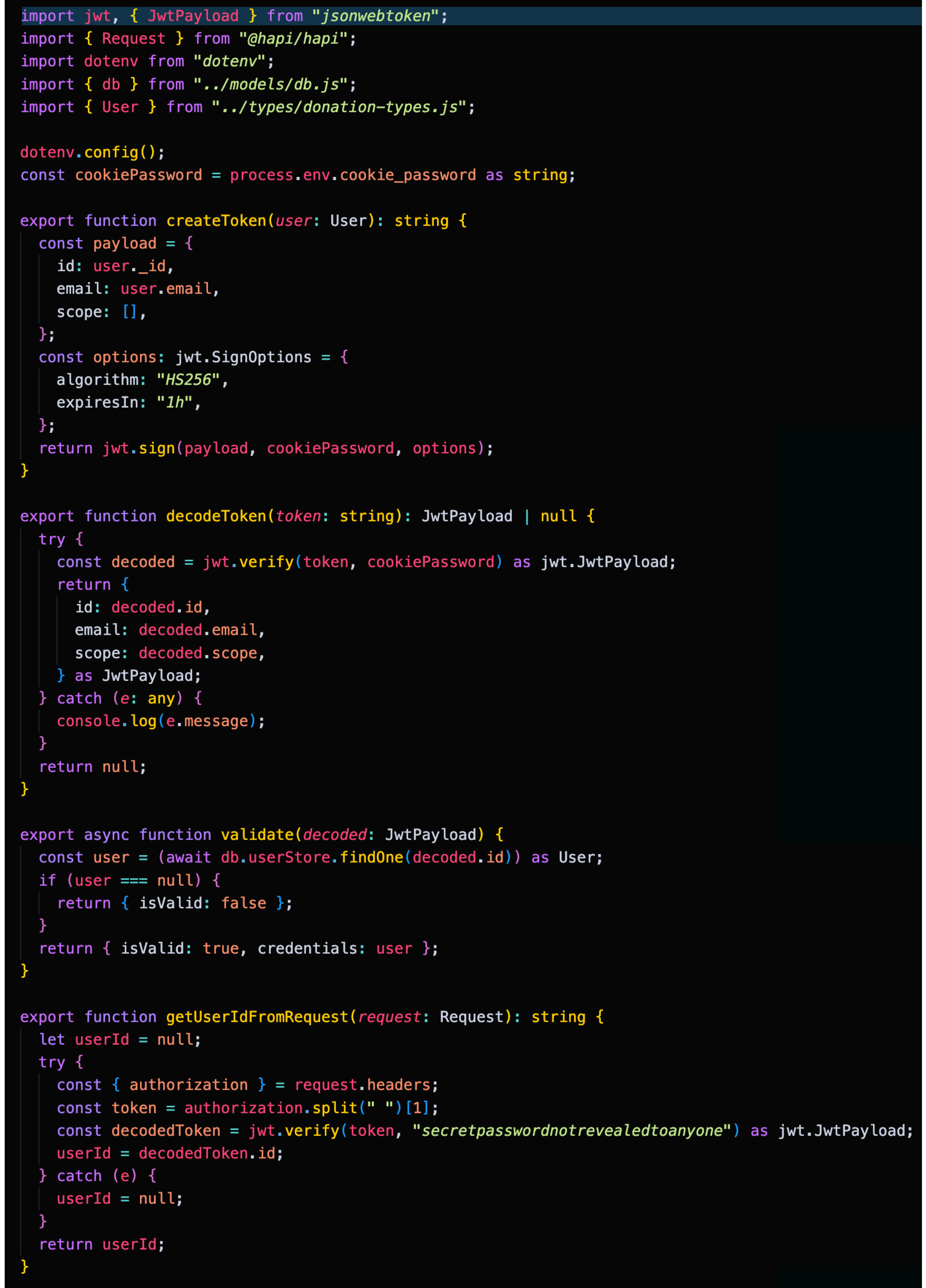
- API Routes

JWT Utils



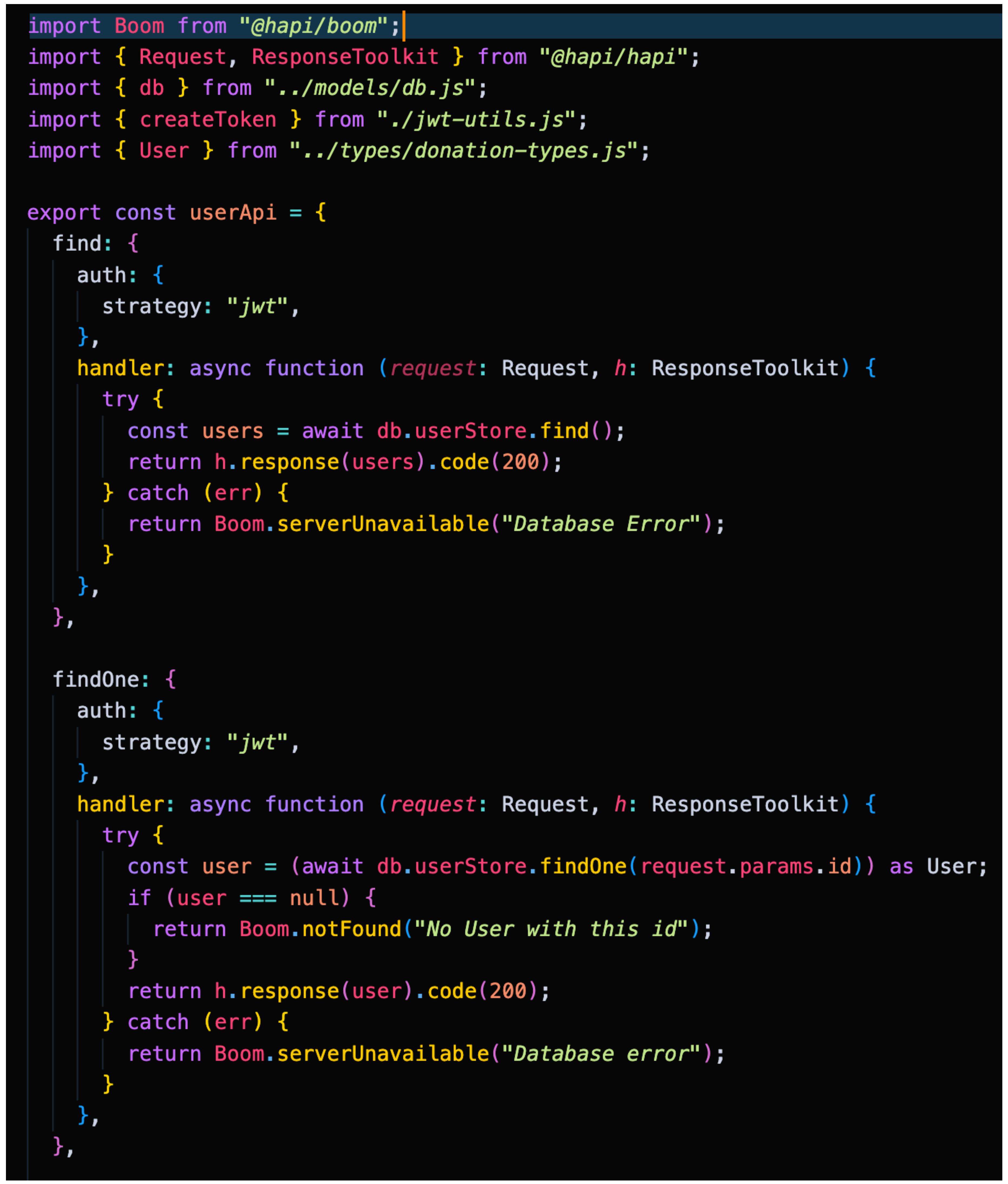
```
npm install hapi-auth-jwt2
npm install jsonwebtoken
```

- JWT Helper functions
  - createToken()
  - decodeToken()
  - validate()
  - getUserIdFromRequest()

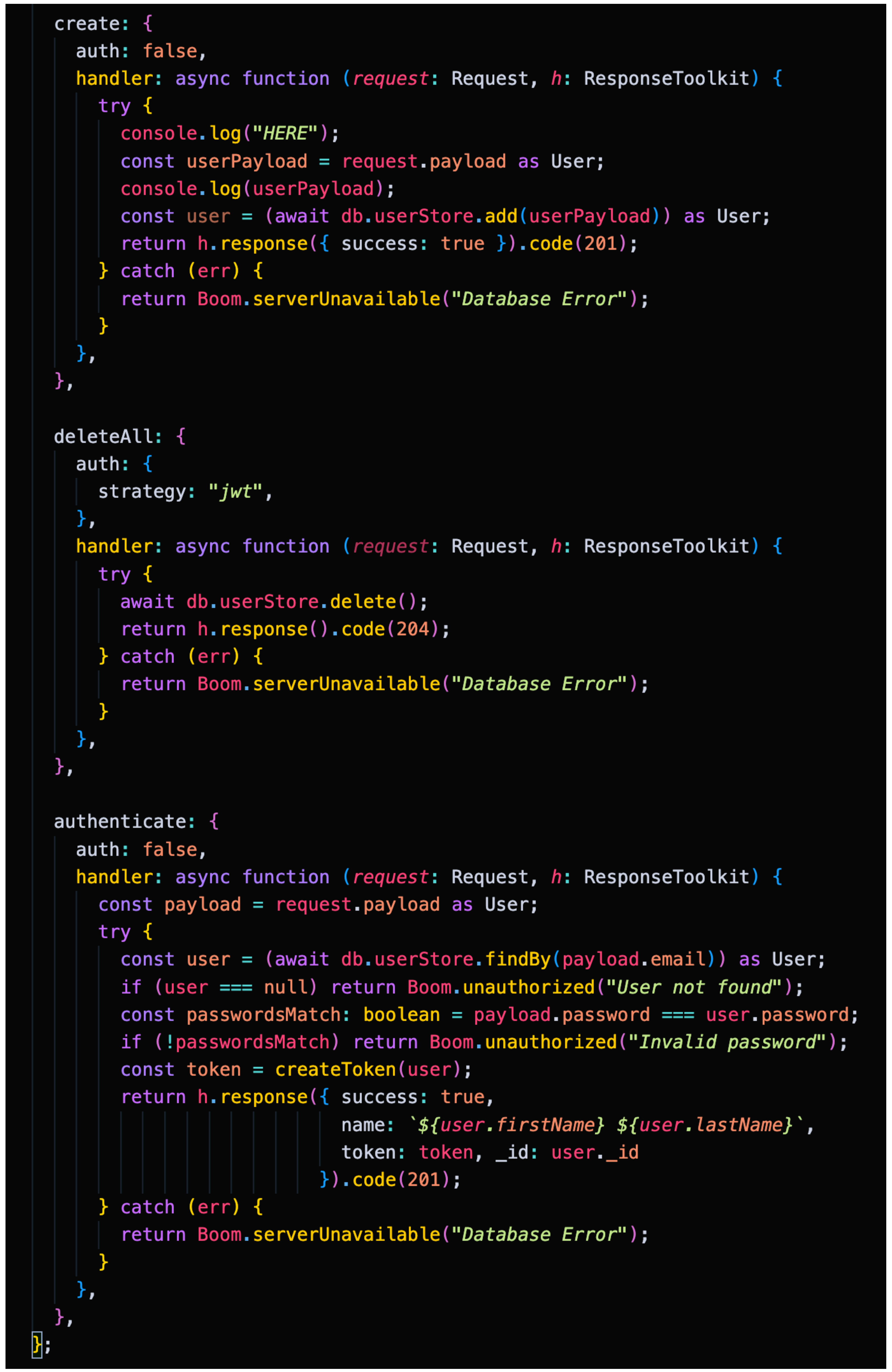


```
import jwt, { JwtPayload } from "jsonwebtoken";
import { Request } from "@hapi/hapi";
import dotenv from "dotenv";
import { db } from "../models/db.js";
import { User } from "../types/donation-types.js";

dotenv.config();
const cookiePassword = process.env.cookie_password as string;

export function createToken(user: User): string {
  const payload = {
    id: user._id,
    email: user.email,
    scope: [],
  };
  const options: jwt.SignOptions = {
    algorithm: "HS256",
    expiresIn: "1h",
  };
  return jwt.sign(payload, cookiePassword, options);
}

export function decodeToken(token: string): JwtPayload | null {
  try {
    const decoded = jwt.verify(token, cookiePassword) as jwt.JwtPayload;
    return {
      id: decoded.id,
      email: decoded.email,
      scope: decoded.scope,
    } as JwtPayload;
  } catch (e: any) {
    console.log(e.message);
  }
  return null;
}

export async function validate(decoded: JwtPayload) {
  const user = (await db.userStore.findOne(decoded.id)) as User;
  if (user === null) {
    return { isValid: false };
  }
  return { isValid: true, credentials: user };
}

export function getUserIdFromRequest(request: Request): string {
  let userId = null;
  try {
    const { authorization } = request.headers;
    const token = authorization.split(" ")[1];
    const decodedToken = jwt.verify(token, "secretpasswordnotrevealedtoanyone") as jwt.JwtPayload;
    userId = decodedToken.id;
  } catch (e) {
    userId = null;
  }
  return userId;
}
```

User API Implementation



```
import Boom from "@hapi/boom";
import { Request, ResponseToolkit } from "@hapi/hapi";
import { db } from "../models/db.js";
import { createToken } from "./jwt-utils.js";
import { User } from "../types/donation-types.js";

export const userApi = {
  find: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request: Request, h: ResponseToolkit) {
      try {
        const users = await db.userStore.find();
        return h.response(users).code(200);
      } catch (err) {
        return Boom.serverUnavailable("Database Error");
      }
    },
  },

  findOne: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request: Request, h: ResponseToolkit) {
      try {
        const user = (await db.userStore.findOne(request.params.id)) as User;
        if (user === null) {
          return Boom.notFound("No User with this id");
        }
        return h.response(user).code(200);
      } catch (err) {
        return Boom.serverUnavailable("Database error");
      }
    },
  },
```



```
  create: {
    auth: false,
    handler: async function (request: Request, h: ResponseToolkit) {
      try {
        console.log("HERE");
        const userPayload = request.payload as User;
        console.log(userPayload);
        const user = (await db.userStore.add(userPayload)) as User;
        return h.response({ success: true }).code(201);
      } catch (err) {
        return Boom.serverUnavailable("Database Error");
      }
    },
  },

  deleteAll: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request: Request, h: ResponseToolkit) {
      try {
        await db.userStore.delete();
        return h.response().code(204);
      } catch (err) {
        return Boom.serverUnavailable("Database Error");
      }
    },
  },

  authenticate: {
    auth: false,
    handler: async function (request: Request, h: ResponseToolkit) {
      const payload = request.payload as User;
      try {
        const user = (await db.userStore.findBy(payload.email)) as User;
        if (user === null) return Boom.unauthorized("User not found");
        const passwordsMatch: boolean = payload.password === user.password;
        if (!passwordsMatch) return Boom.unauthorized("Invalid password");
        const token = createToken(user);
        return h.response({ success: true,
                            name: `${user.firstName} ${user.lastName}`,
                            token: token, _id: user._id
                          }).code(201);
      } catch (err) {
        return Boom.serverUnavailable("Database Error");
      }
    },
  },
};
```

User API Routes

- Define API routes

```
import { userApi } from "./api/users-api.js";

export const apiRoutes = [
  { method: "GET", path: "/api/users", config: userApi.find },
  { method: "POST", path: "/api/users", config: userApi.create },
  { method: "DELETE", path: "/api/users", config: userApi.deleteAll },
  { method: "GET", path: "/api/users/{id}", config: userApi.findOne },
  { method: "POST", path: "/api/users/authenticate", config: userApi.authenticate },
];
```

server.ts

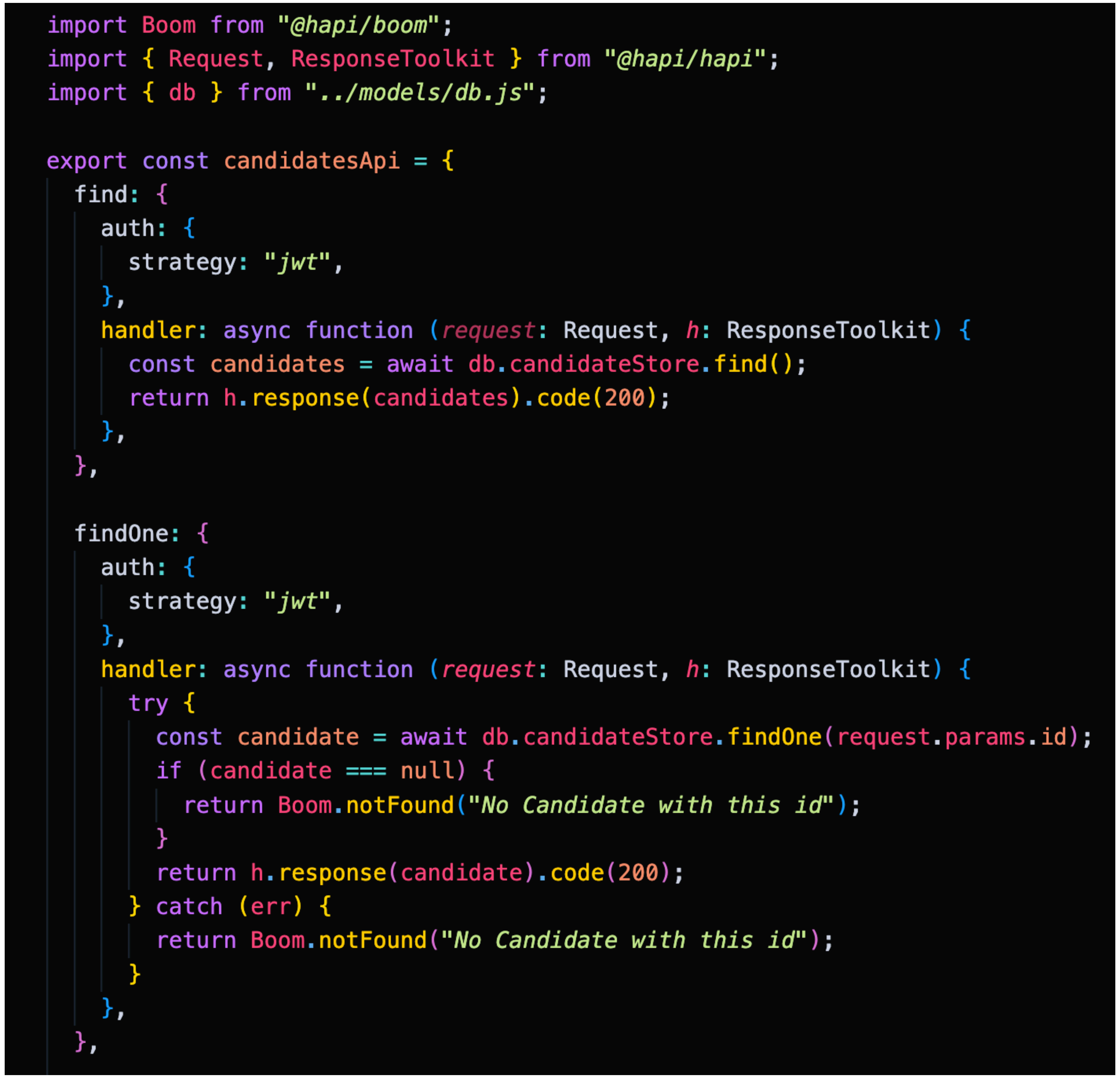
- Configure JWT

```
  server.auth.strategy("jwt", "jwt", {
    key: process.env.cookie_password,
    validate: validate,
    verifyOptions: { algorithms: ["HS256"] },
  });
```

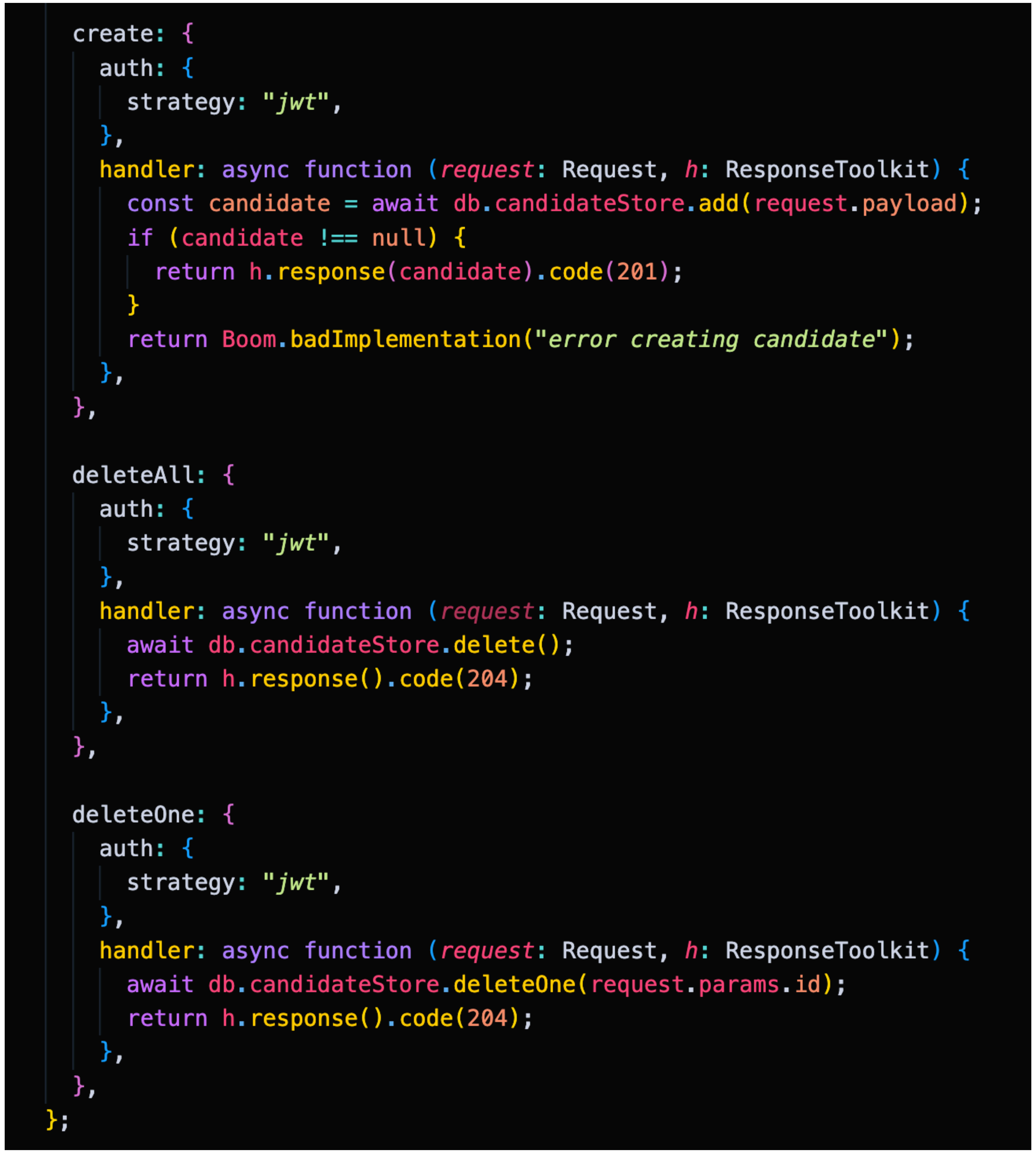
- Install API routes

```
  server.route(apiRoutes);
```

Candidates API

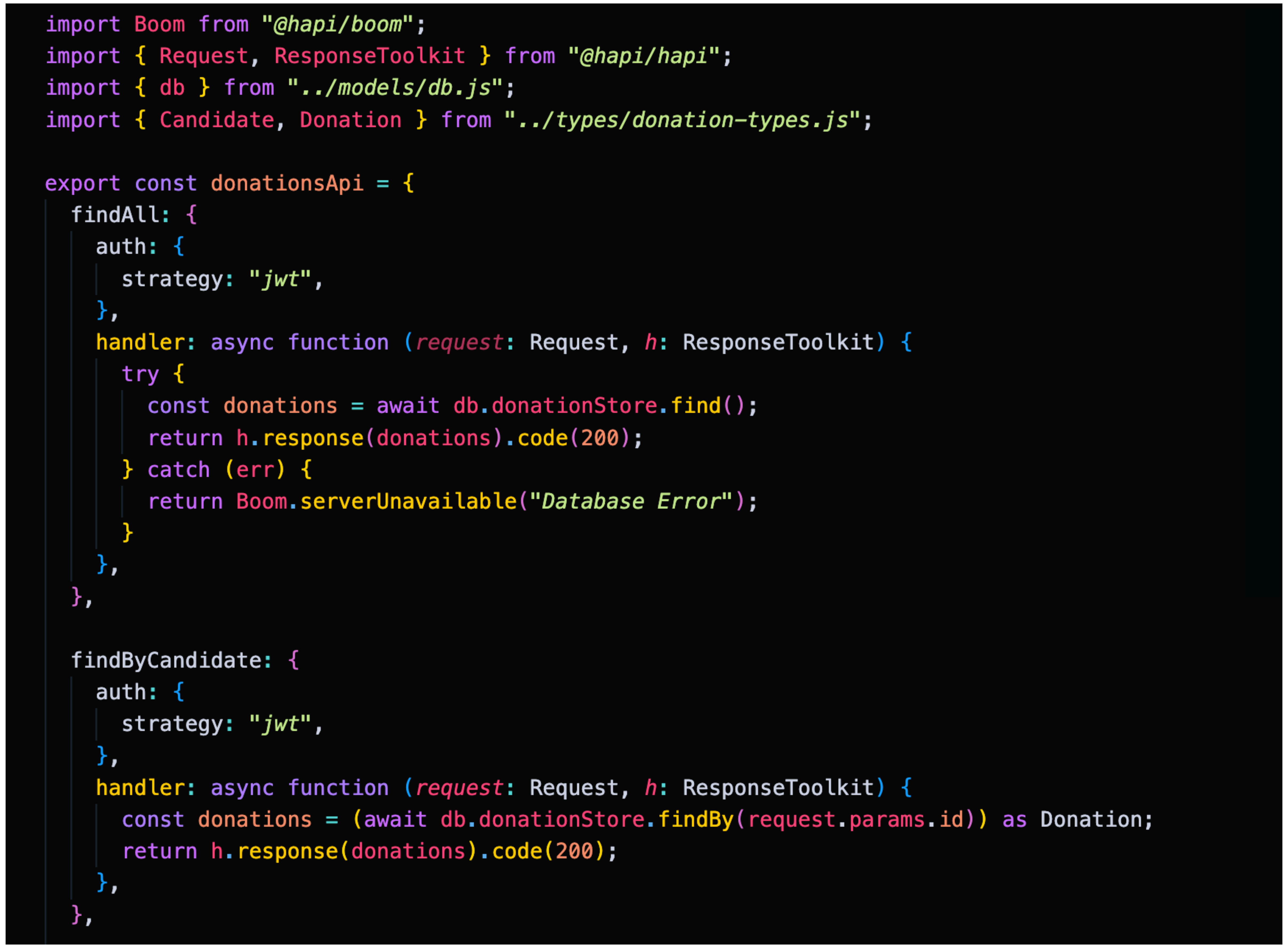


```
import Boom from "@hapi/boom";
import { Request, ResponseToolkit } from "@hapi/hapi";
import { db } from "../models/db.js";

export const candidatesApi = {
  find: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request: Request, h: ResponseToolkit) {
      const candidates = await db.candidateStore.find();
      return h.response(candidates).code(200);
    },
  },

  findOne: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request: Request, h: ResponseToolkit) {
      try {
        const candidate = await db.candidateStore.findOne(request.params.id);
        if (candidate === null) {
          return Boom.notFound("No Candidate with this id");
        }
        return h.response(candidate).code(200);
      } catch (err) {
        return Boom.notFound("No Candidate with this id");
      }
    },
  },
```

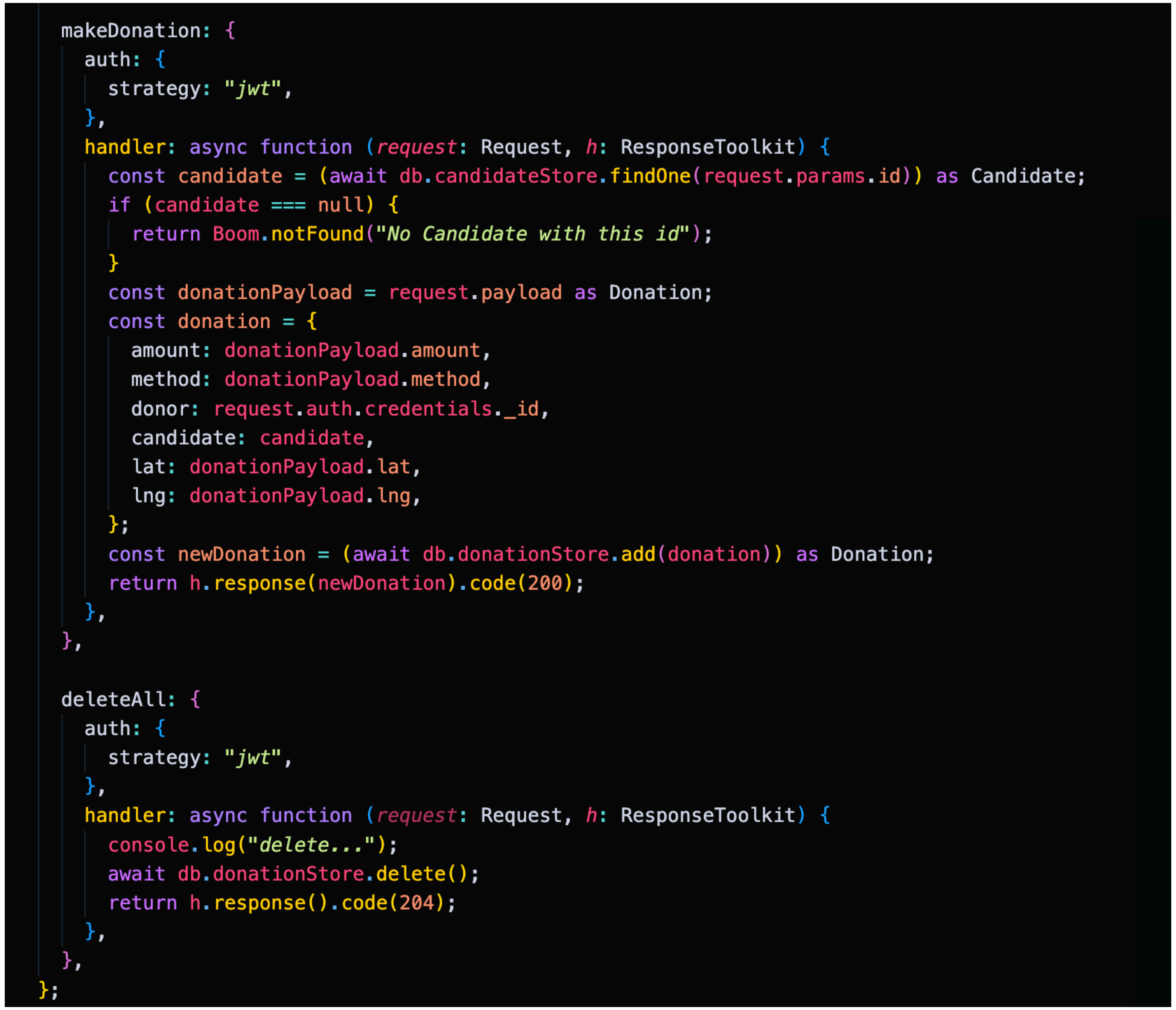


```
  create: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request: Request, h: ResponseToolkit) {
      const candidate = await db.candidateStore.add(request.payload);
      if (candidate !== null) {
        return h.response(candidate).code(201);
      }
      return Boom.badImplementation("error creating candidate");
    },
  },

  deleteAll: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request: Request, h: ResponseToolkit) {
      await db.candidateStore.delete();
      return h.response().code(204);
    },
  },

  deleteOne: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request: Request, h: ResponseToolkit) {
      await db.candidateStore.deleteOne(request.params.id);
      return h.response().code(204);
    },
  },
};
```

Donate API

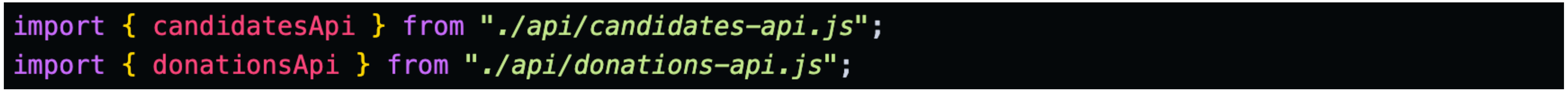


```
import Boom from "@hapi/boom";
import { Request, ResponseToolkit } from "@hapi/hapi";
import { db } from "../models/db.js";
import { Candidate, Donation } from "../types/donation-types.js";

export const donationsApi = {
  findAll: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request: Request, h: ResponseToolkit) {
      try {
        const donations = await db.donationStore.find();
        return h.response(donations).code(200);
      } catch (err) {
        return Boom.serverUnavailable("Database Error");
      }
    },
  },

  findByCandidate: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request: Request, h: ResponseToolkit) {
      const donations = (await db.donationStore.findBy(request.params.id)) as Donation;
      return h.response(donations).code(200);
    },
  },
```

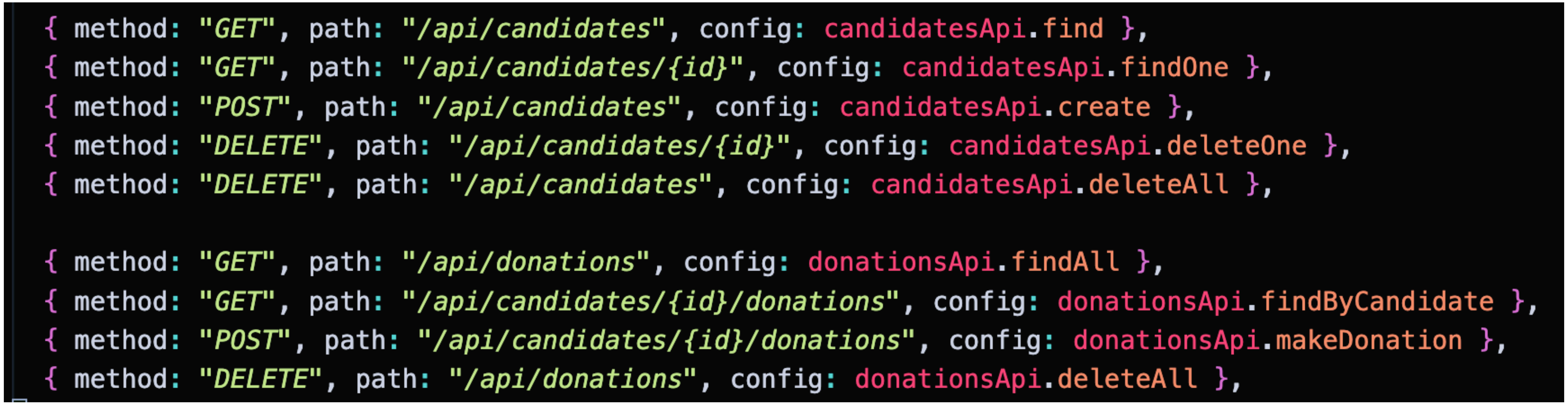


```
  makeDonation: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request: Request, h: ResponseToolkit) {
      const candidate = (await db.candidateStore.findOne(request.params.id)) as Candidate;
      if (candidate === null) {
        return Boom.notFound("No Candidate with this id");
      }
      const donationPayload = request.payload as Donation;
      const donation = {
        amount: donationPayload.amount,
        method: donationPayload.method,
        donor: request.auth.credentials._id,
        candidate: candidate,
        lat: donationPayload.lat,
        lng: donationPayload.lng,
      };
      const newDonation = (await db.donationStore.add(donation)) as Donation;
      return h.response(newDonation).code(200);
    },
  },

  deleteAll: {
    auth: {
      strategy: "jwt",
    },
    handler: async function (request: Request, h: ResponseToolkit) {
      console.log("delete...");
      await db.donationStore.delete();
      return h.response().code(204);
    },
  },
};
```

Candidates & Donate API Routes



```
import { candidatesApi } from "./api/candidates-api.js";
import { donationsApi } from "./api/donations-api.js";
```



```
  { method: "GET", path: "/api/candidates", config: candidatesApi.find },
  { method: "GET", path: "/api/candidates/{id}", config: candidatesApi.findOne },
  { method: "POST", path: "/api/candidates", config: candidatesApi.create },
  { method: "DELETE", path: "/api/candidates/{id}", config: candidatesApi.deleteOne },
  { method: "DELETE", path: "/api/candidates", config: candidatesApi.deleteAll },

  { method: "GET", path: "/api/donations", config: donationsApi.findAll },
  { method: "GET", path: "/api/candidates/{id}/donations", config: donationsApi.findByCandidate },
  { method: "POST", path: "/api/candidates/{id}/donations", config: donationsApi.makeDonation },
  { method: "DELETE", path: "/api/donations", config: donationsApi.deleteAll },
```

Hapi/JWT API

HAPI Authentication API using JWT