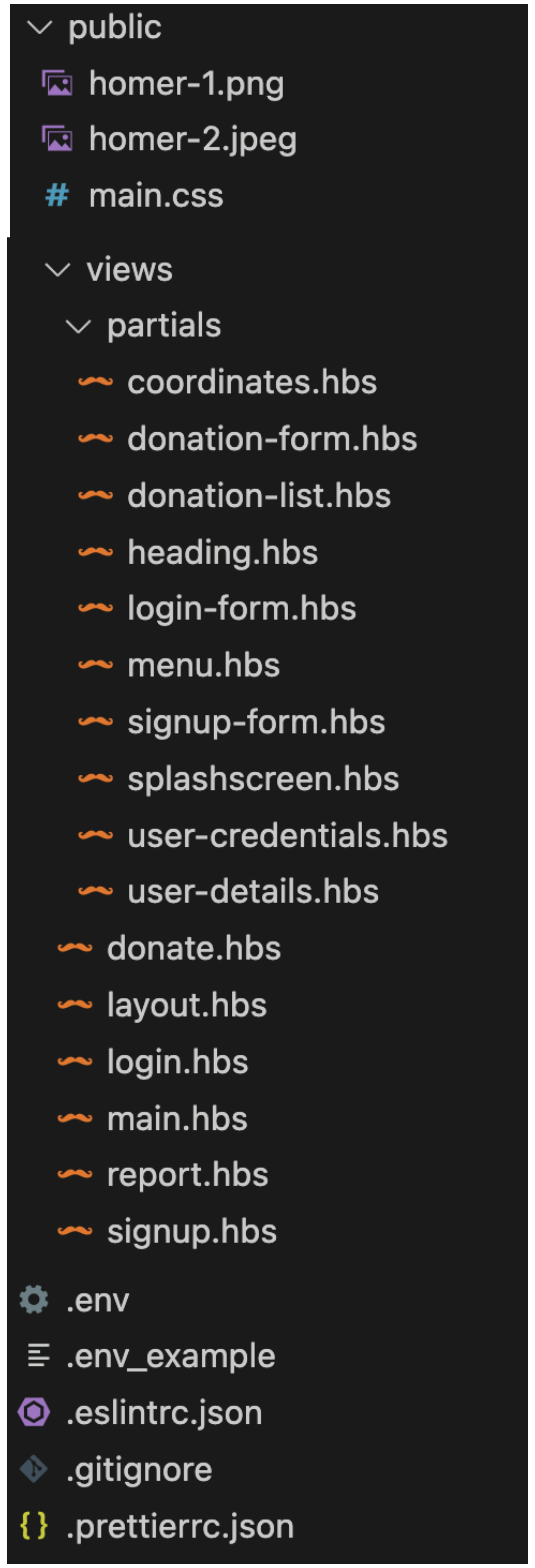
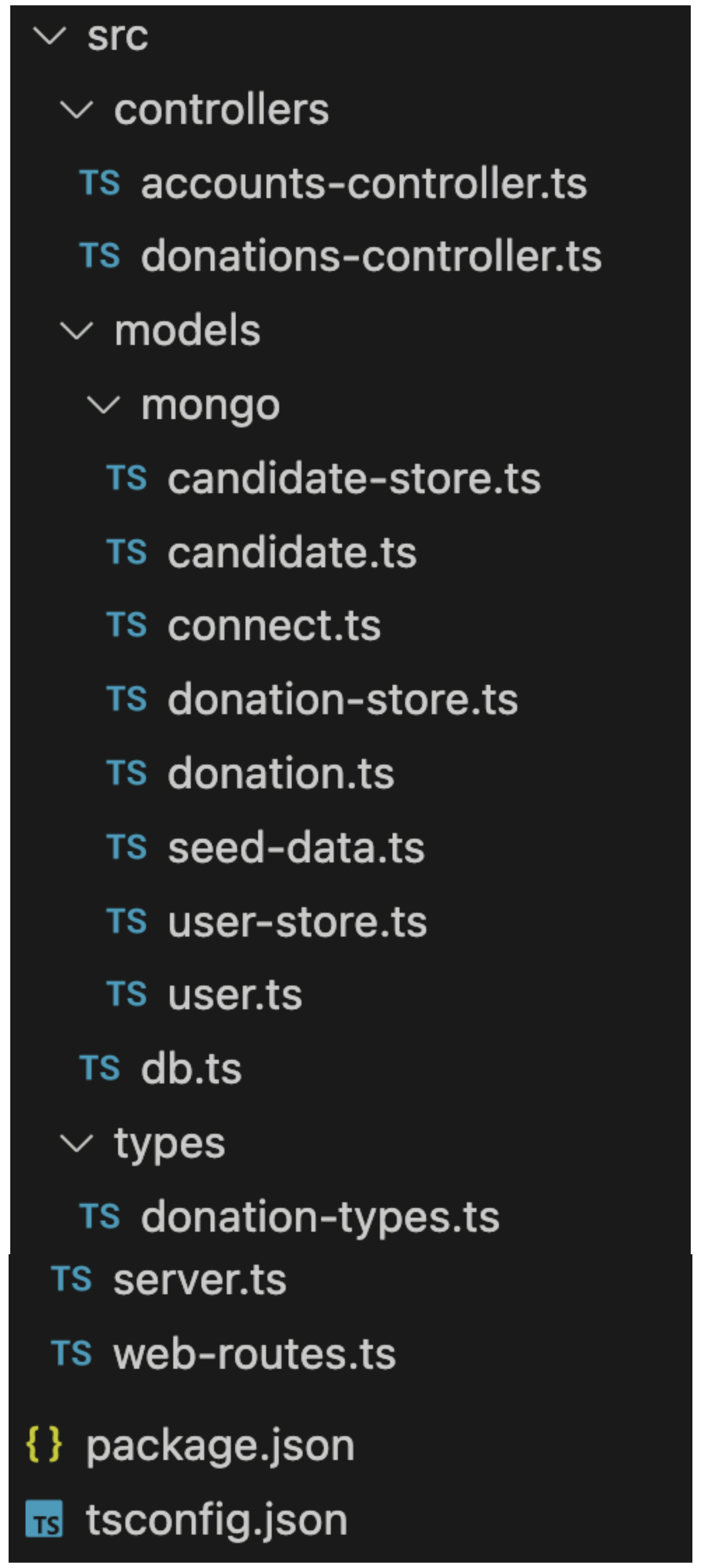
Types

Introduce Types into Donation



```
public
  homer-1.png
  homer-2.jpeg
  main.css
views
  partials
    coordinates.hbs
    donation-form.hbs
    donation-list.hbs
    heading.hbs
    login-form.hbs
    menu.hbs
    signup-form.hbs
    splashscreen.hbs
    user-credentials.hbs
    user-details.hbs
  donate.hbs
  layout.hbs
  login.hbs
  main.hbs
  report.hbs
  signup.hbs
.env
.env_example
.eslintrc.json
.gitignore
.prettierrc.json
```

- Unchanged

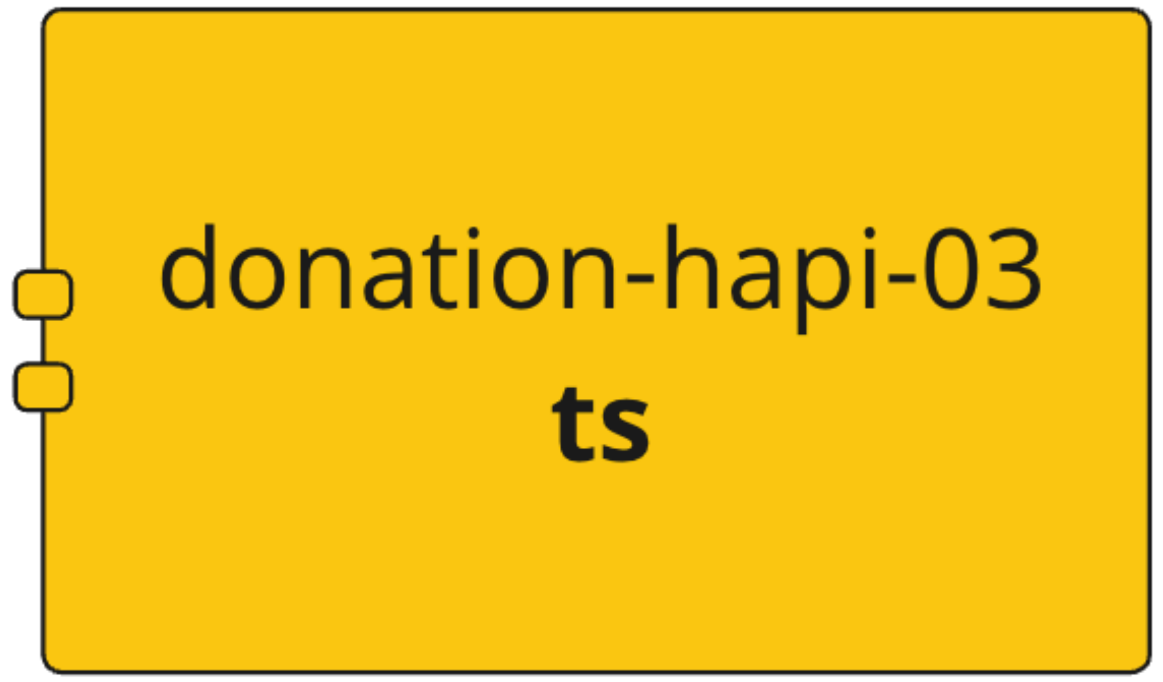


```
src
  controllers
    accounts-controller.ts
    donations-controller.ts
  models
    mongo
      candidate-store.ts
      candidate.ts
      connect.ts
      donation-store.ts
      donation.ts
      seed-data.ts
      user-store.ts
      user.ts
    db.ts
  types
    donation-types.ts
  server.ts
  web-routes.ts
package.json
tsconfig.json
```

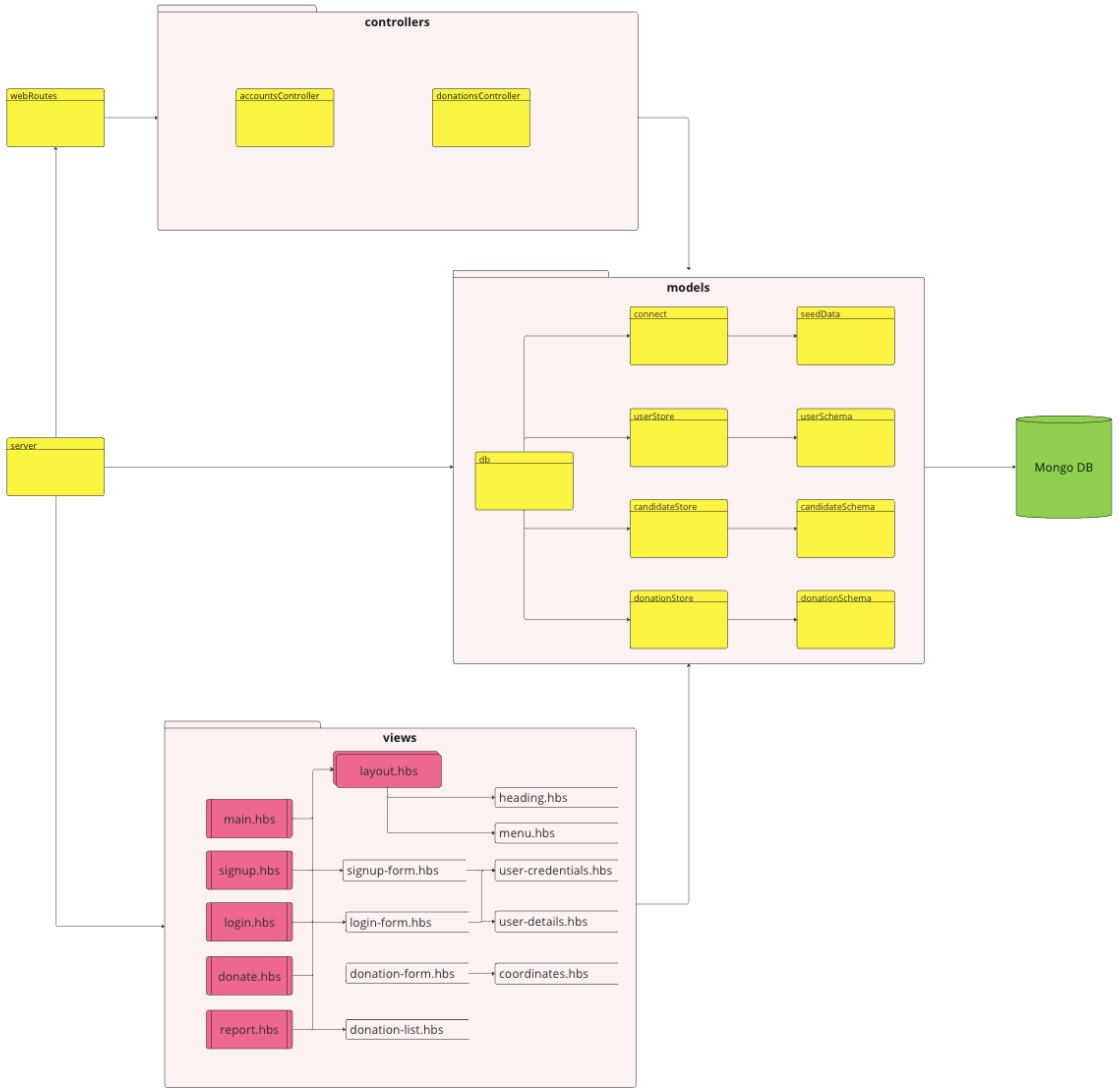
- Modified/new

donation-hapi-03-ts

- TypeScript version of all js modules

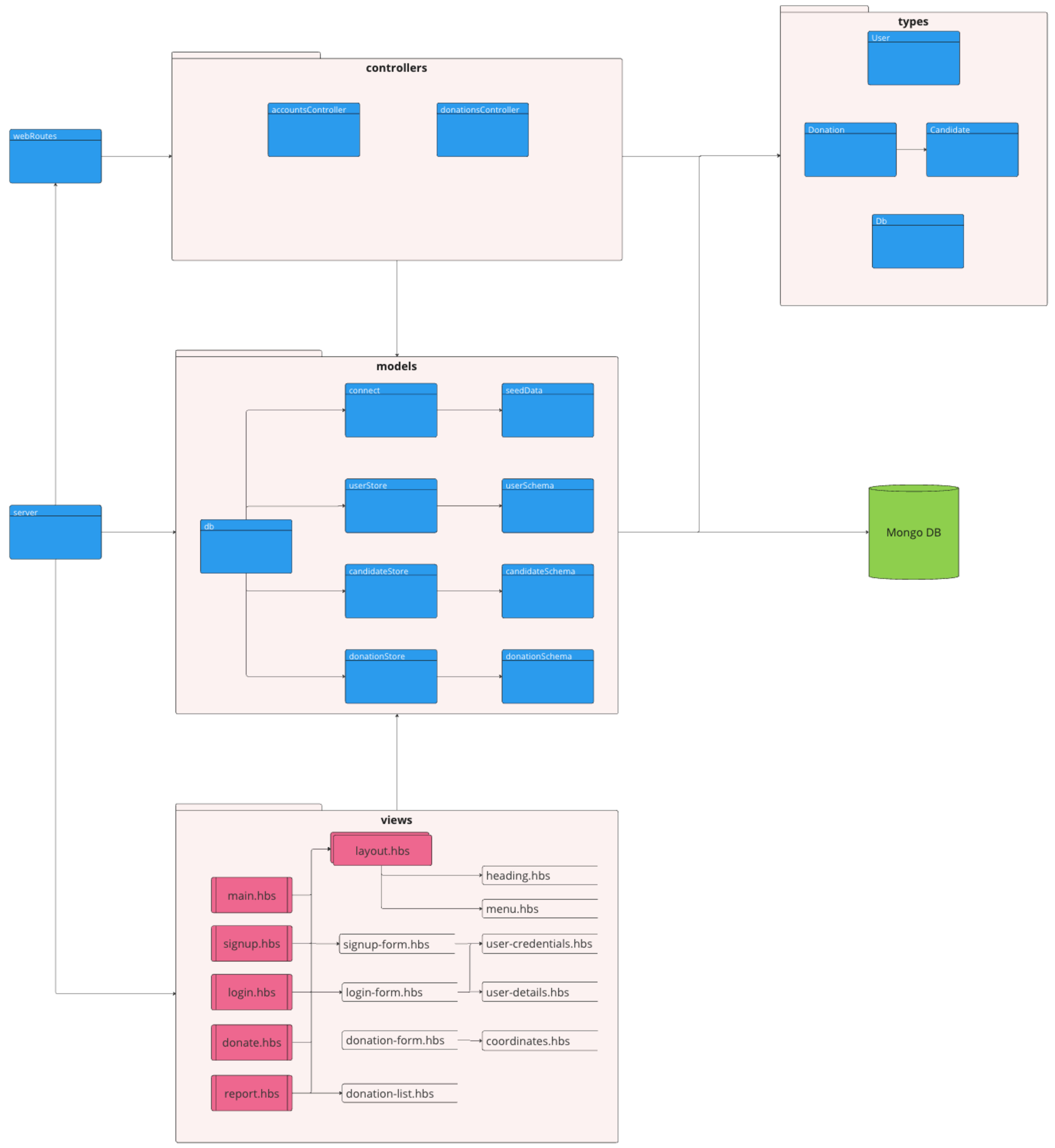


- donation-hapi-02-donate

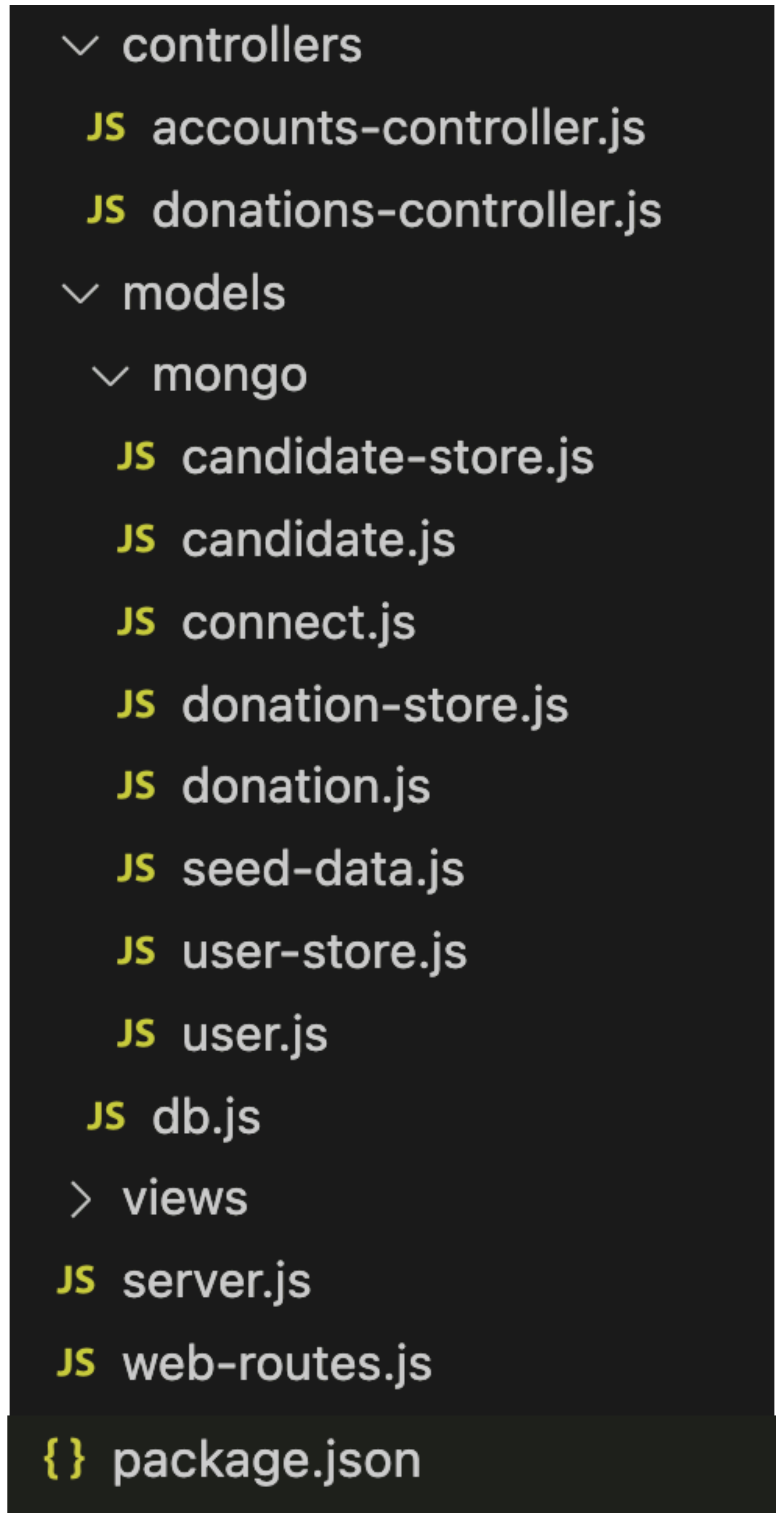


- Donation-hapi-03-ts

- No change in features or UX

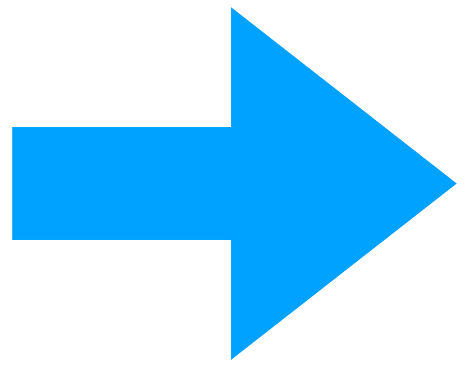


JavaScript -> TypeScript



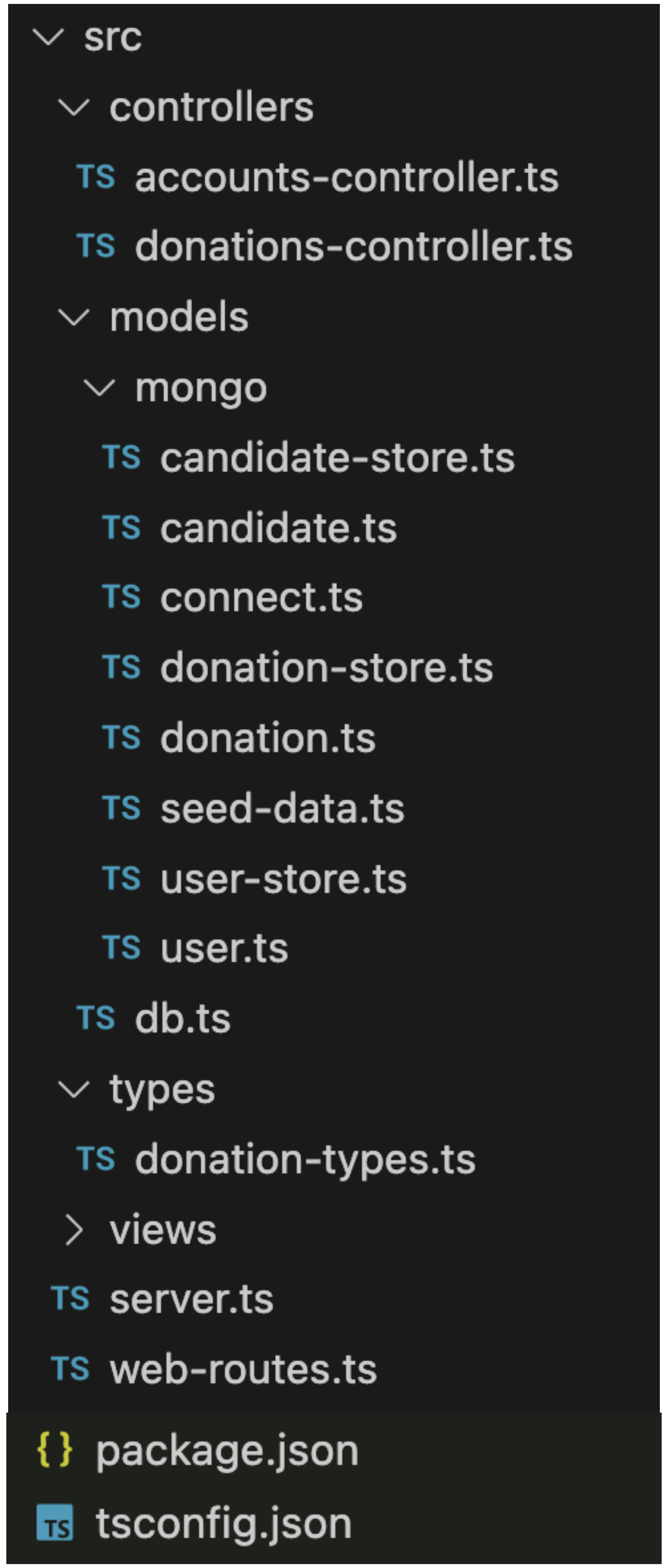
```
∨ controllers
  JS accounts-controller.js
  JS donations-controller.js
∨ models
  ∨ mongo
    JS candidate-store.js
    JS candidate.js
    JS connect.js
    JS donation-store.js
    JS donation.js
    JS seed-data.js
    JS user-store.js
    JS user.js
  JS db.js
> views
JS server.js
JS web-routes.js
{} package.json
```

- Each module renamed from .js to .ts



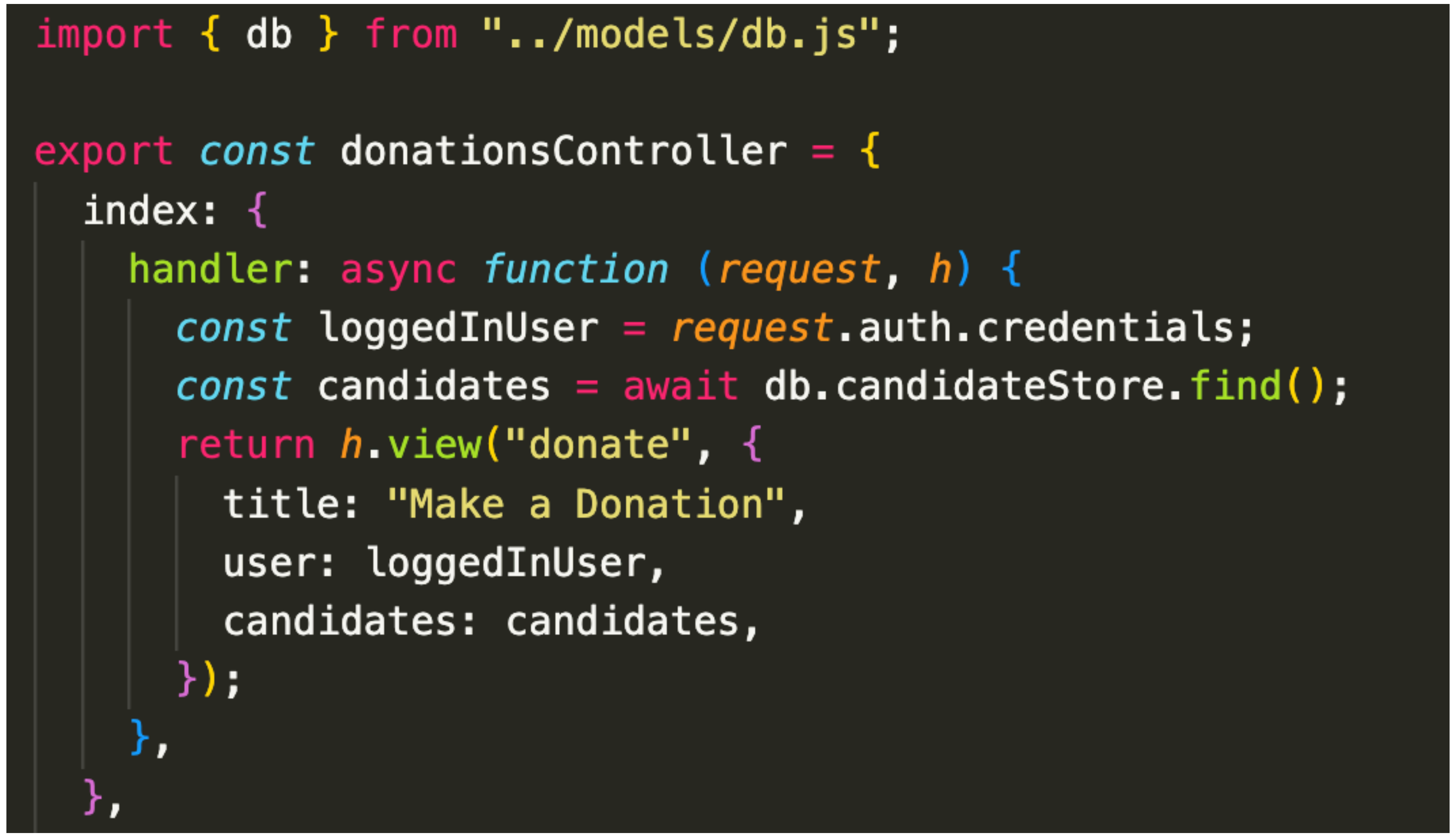
- Refactor to introduce typescript specific syntax

- donation-types introduced

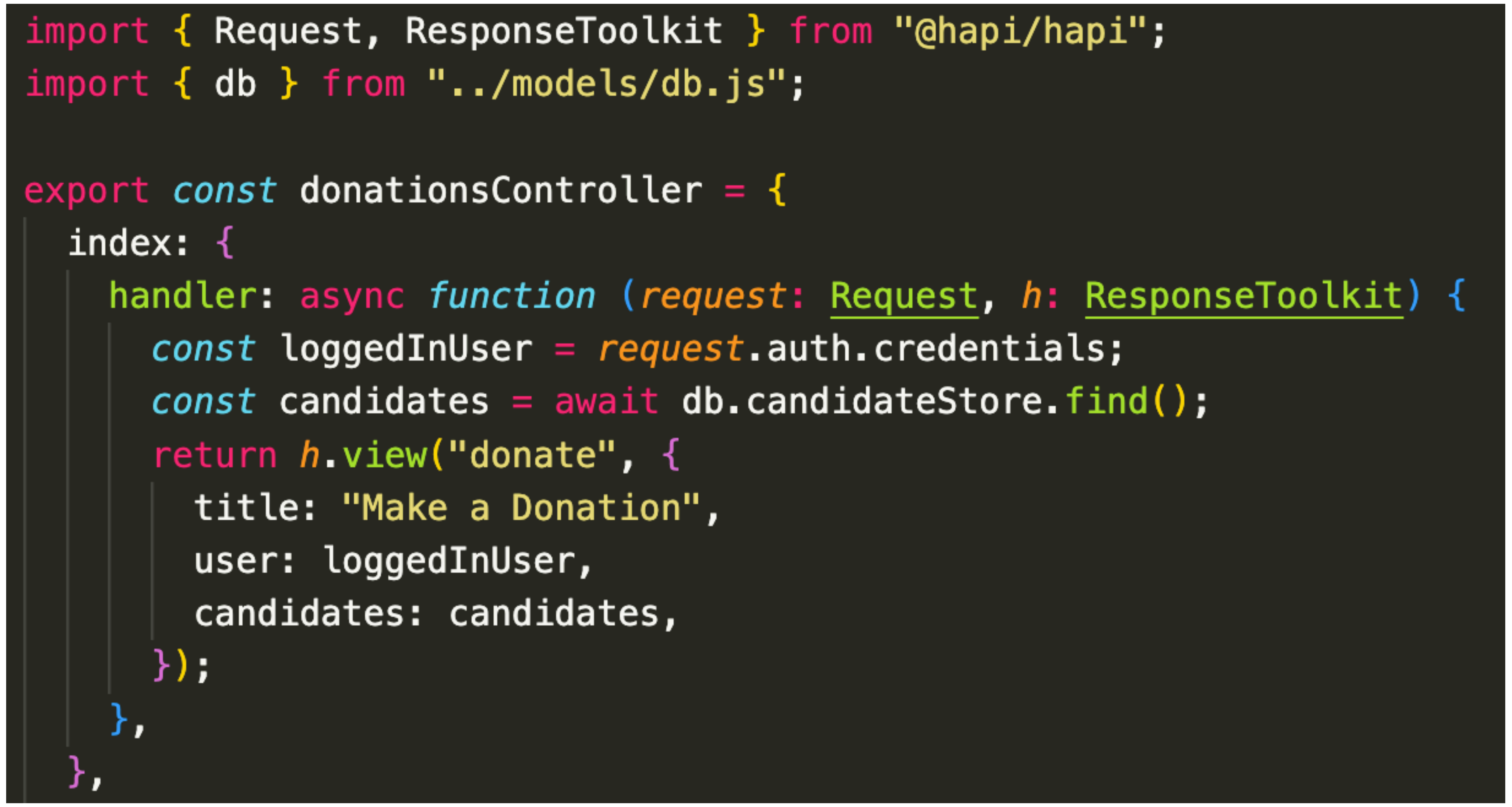


```
∨ src
  ∨ controllers
    TS accounts-controller.ts
    TS donations-controller.ts
  ∨ models
    ∨ mongo
      TS candidate-store.ts
      TS candidate.ts
      TS connect.ts
      TS donation-store.ts
      TS donation.ts
      TS seed-data.ts
      TS user-store.ts
      TS user.ts
    TS db.ts
  ∨ types
    TS donation-types.ts
  > views
  TS server.ts
  TS web-routes.ts
{} package.json
TS tsconfig.json
```

Controllers



```
import { db } from "../models/db.js";

export const donationsController = {
  index: {
    handler: async function (request, h) {
      const loggedInUser = request.auth.credentials;
      const candidates = await db.candidateStore.find();
      return h.view("donate", {
        title: "Make a Donation",
        user: loggedInUser,
        candidates: candidates,
      });
    },
  },
```



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

export const donationsController = {
  index: {
    handler: async function (request: Request, h: ResponseToolkit) {
      const loggedInUser = request.auth.credentials;
      const candidates = await db.candidateStore.find();
      return h.view("donate", {
        title: "Make a Donation",
        user: loggedInUser,
        candidates: candidates,
      });
    },
  },
```

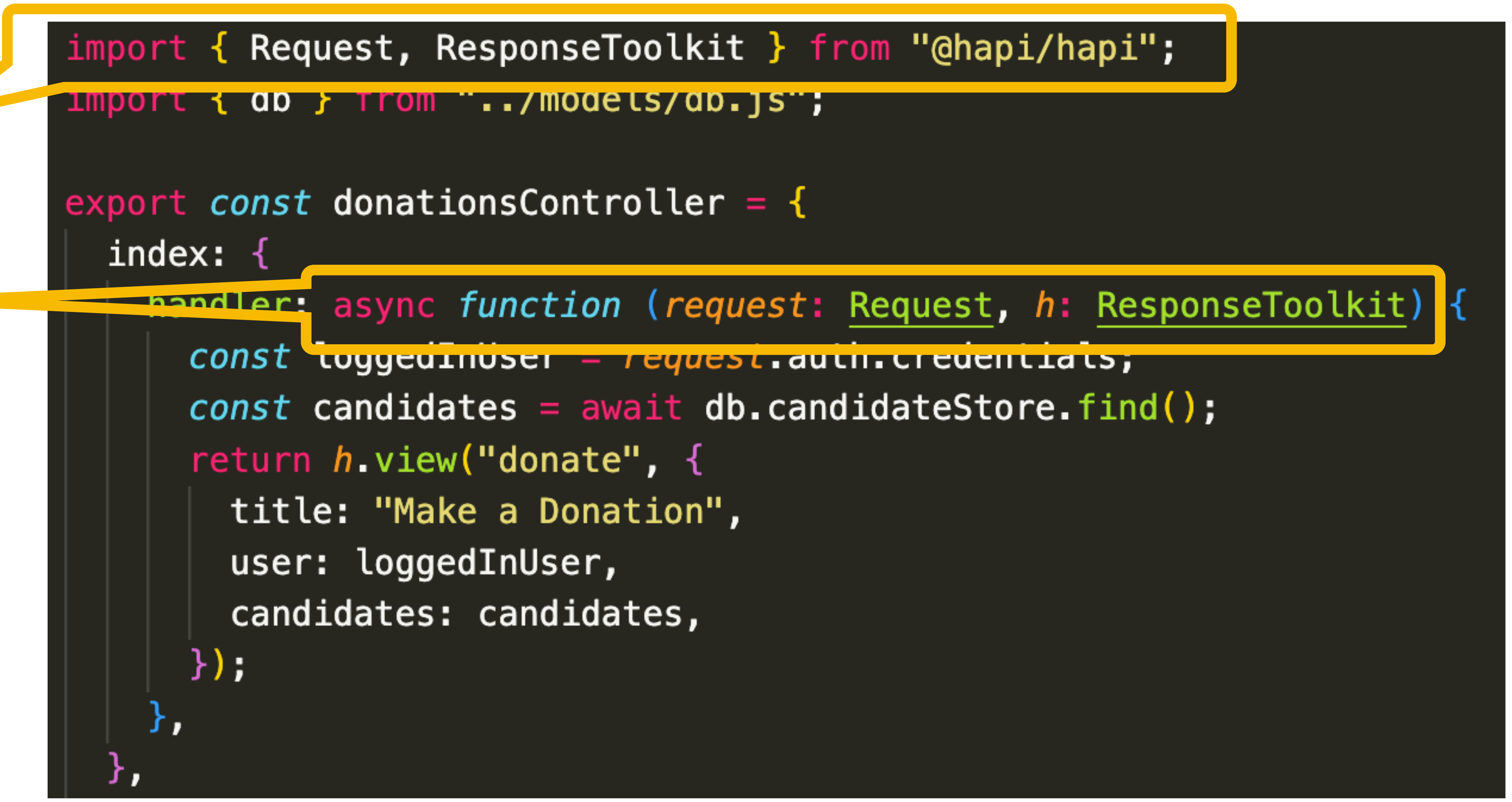
donations-controller.js

donations-controller.ts

Controllers

Import controller paramaters

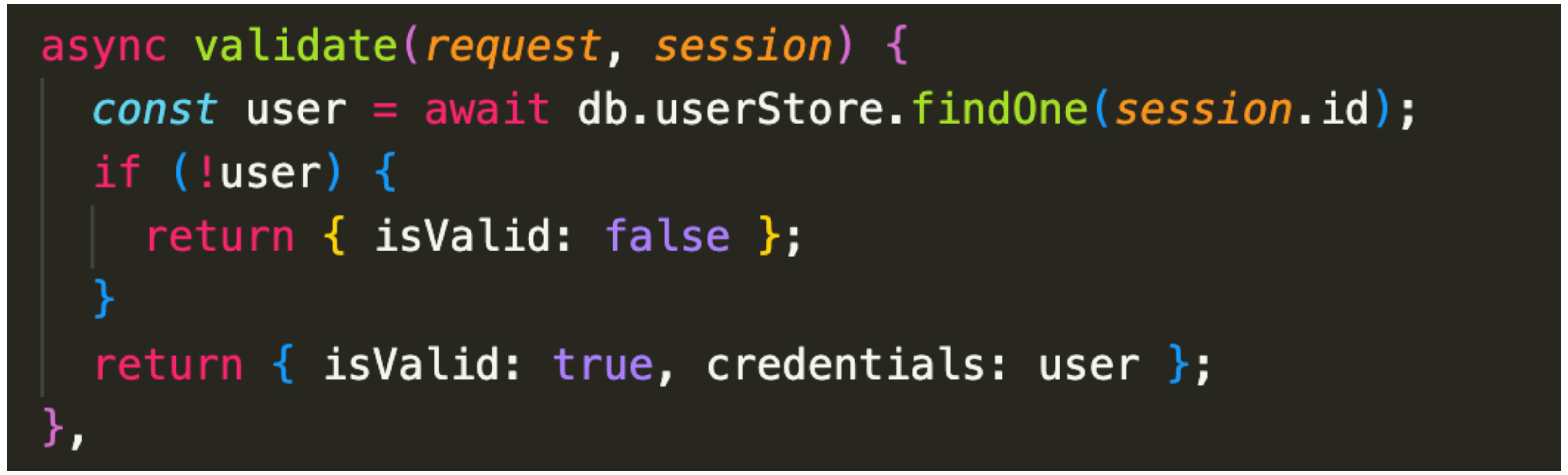
Specifiy parameters to handlers



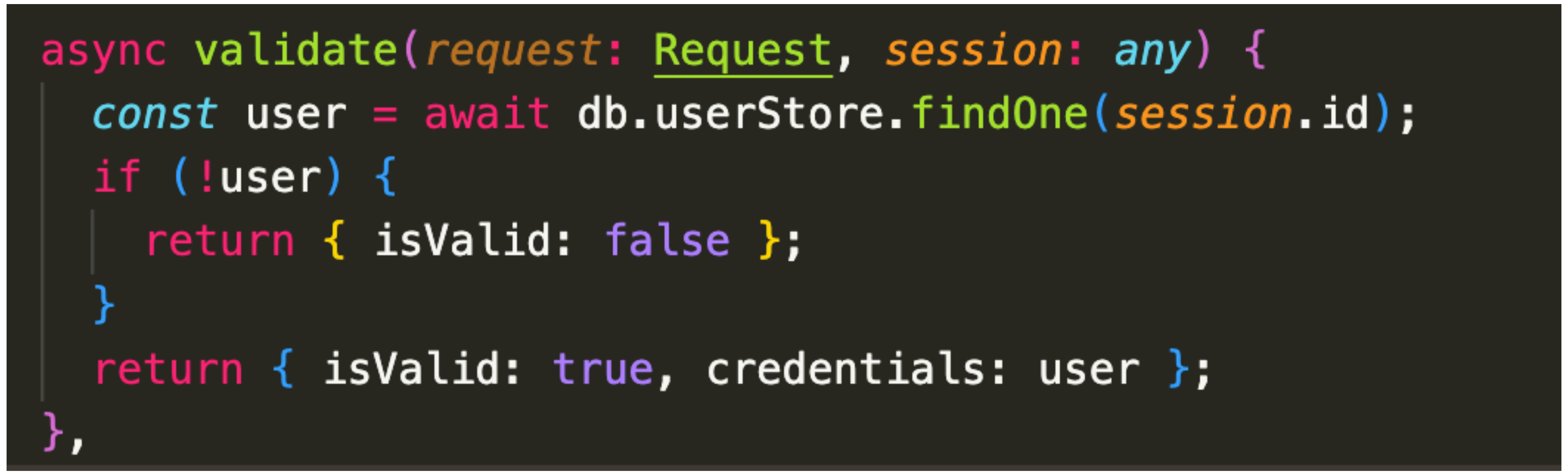
```
import { Request, ResponseToolkit } from "@hapi/hapi";
import { db } from "../models/db.js";

export const donationsController = {
  index: {
    handler: async function (request: Request, h: ResponseToolkit) {
      const loggedInUser = request.auth.credentials;
      const candidates = await db.candidateStore.find();
      return h.view("donate", {
        title: "Make a Donation",
        user: loggedInUser,
        candidates: candidates,
      });
    },
  },
```

donations-controller.ts

Validate



```
async validate(request, session) {
  const user = await db.userStore.findOne(session.id);
  if (!user) {
    return { isValid: false };
  }
  return { isValid: true, credentials: user };
},
```



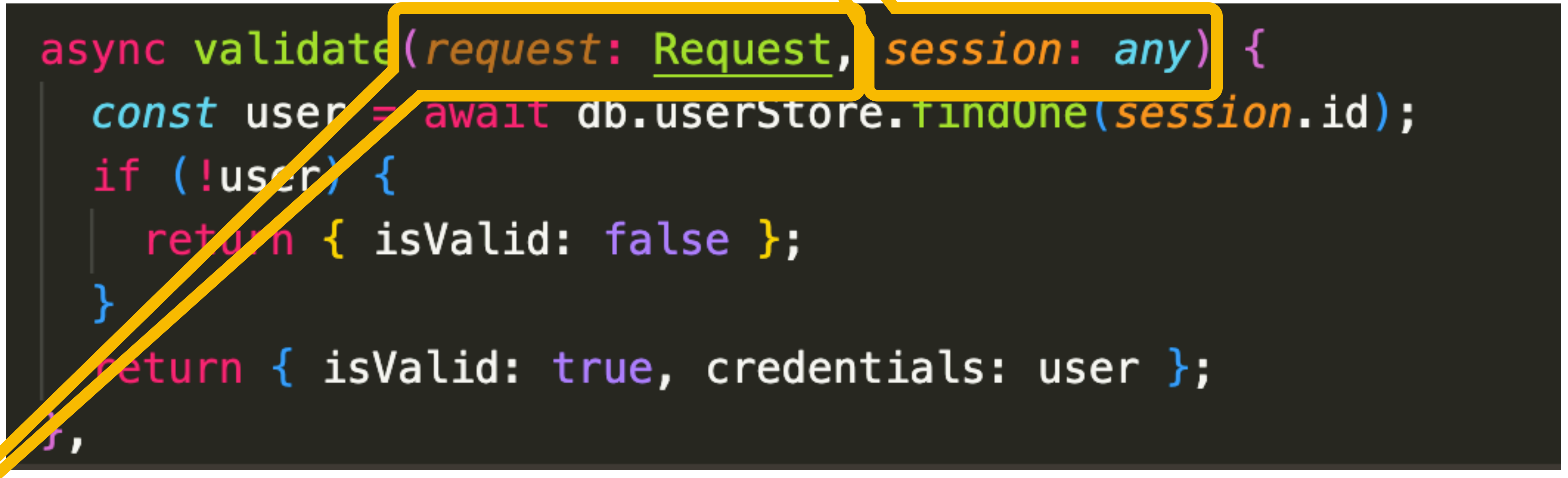
```
async validate(request: Request, session: any) {
  const user = await db.userStore.findOne(session.id);
  if (!user) {
    return { isValid: false };
  }
  return { isValid: true, credentials: user };
},
```

accounts-controller.js

acounts-controller.ts

Validate

Session not available (or we failed to find it), so use ‘**any**’ until we figure it out



```
async validate(request: Request, session: any) {
  const user = await db.userStore.findOne(session.id);
  if (!user) {
    return { isValid: false };
  }
  return { isValid: true, credentials: user };
};
```

Request type defined in Hapi

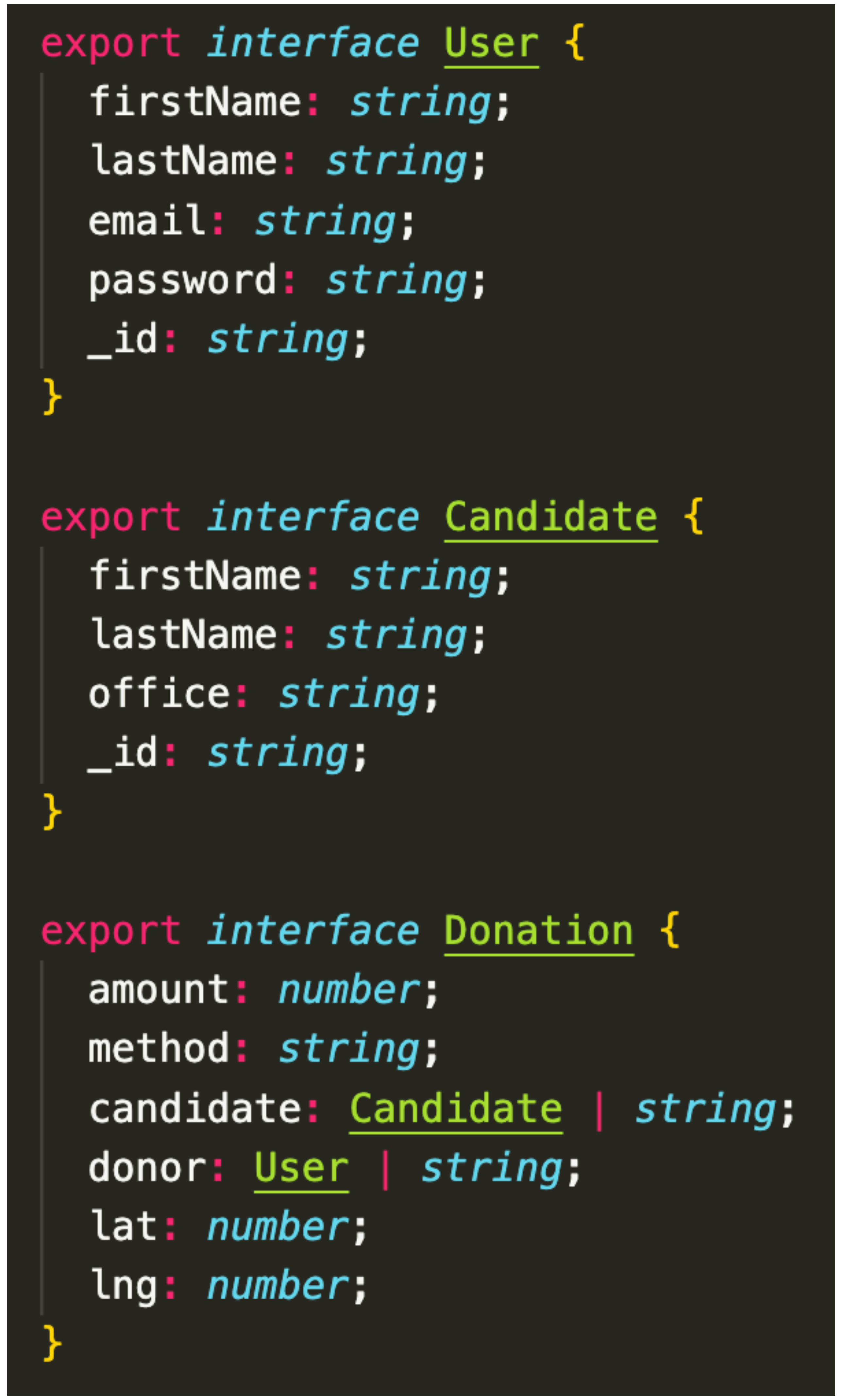
acounts-controller.ts

Donation Types

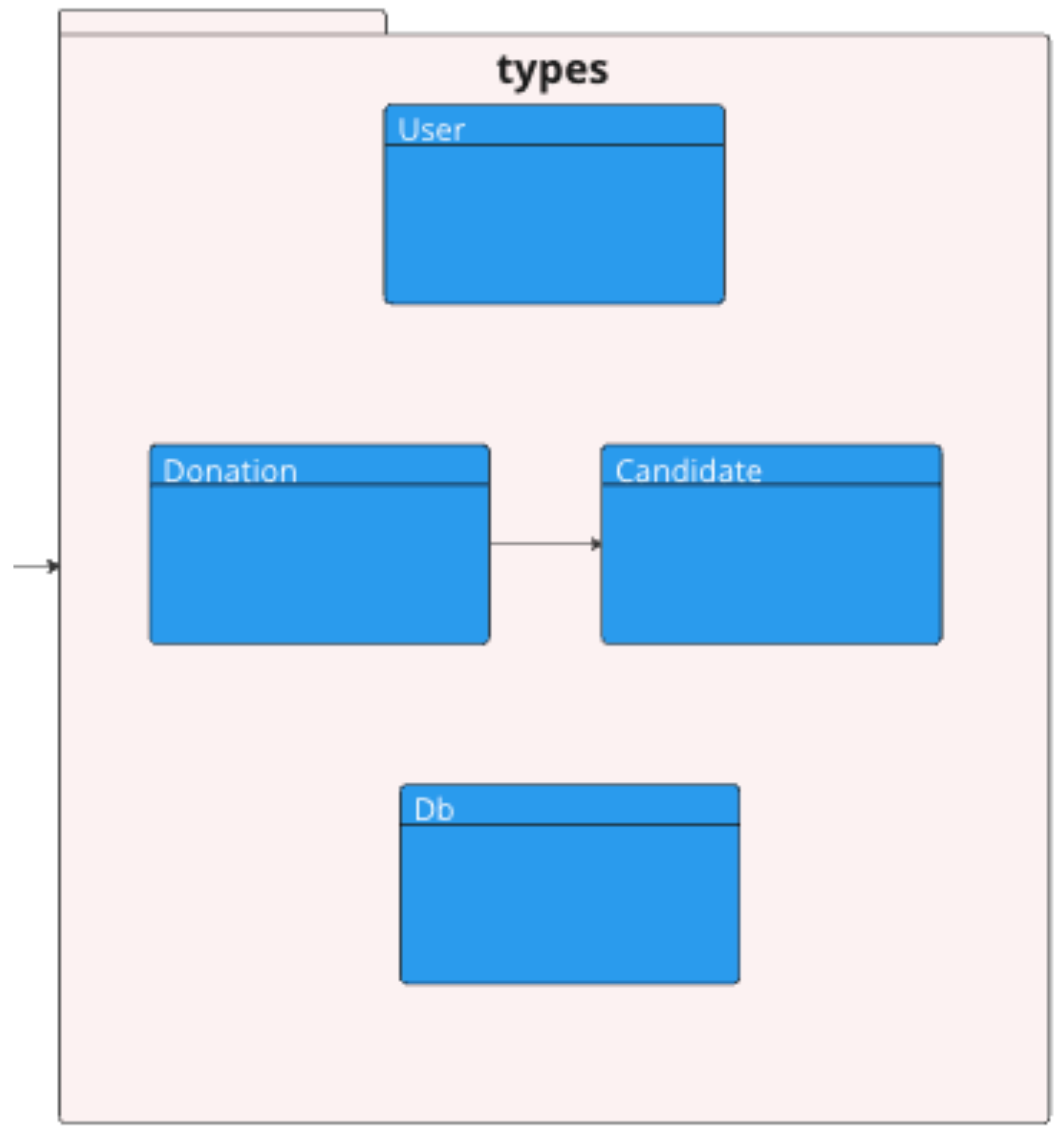
- Interfaces define the shape of objects

- Correspond approximately to interfaces in other languages (Java, Go)

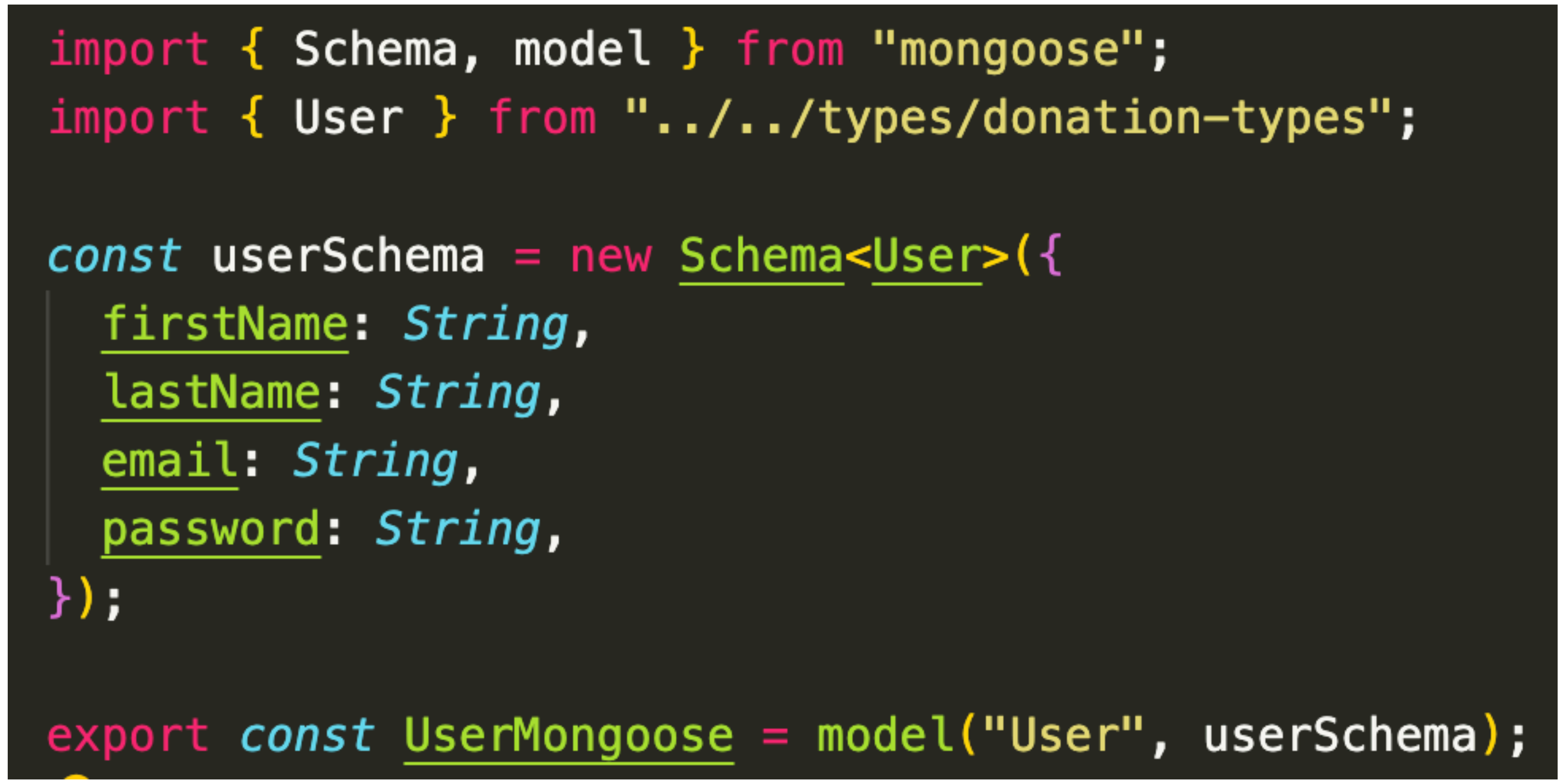
- Use these in parameters and variable declarations to enhance specificity



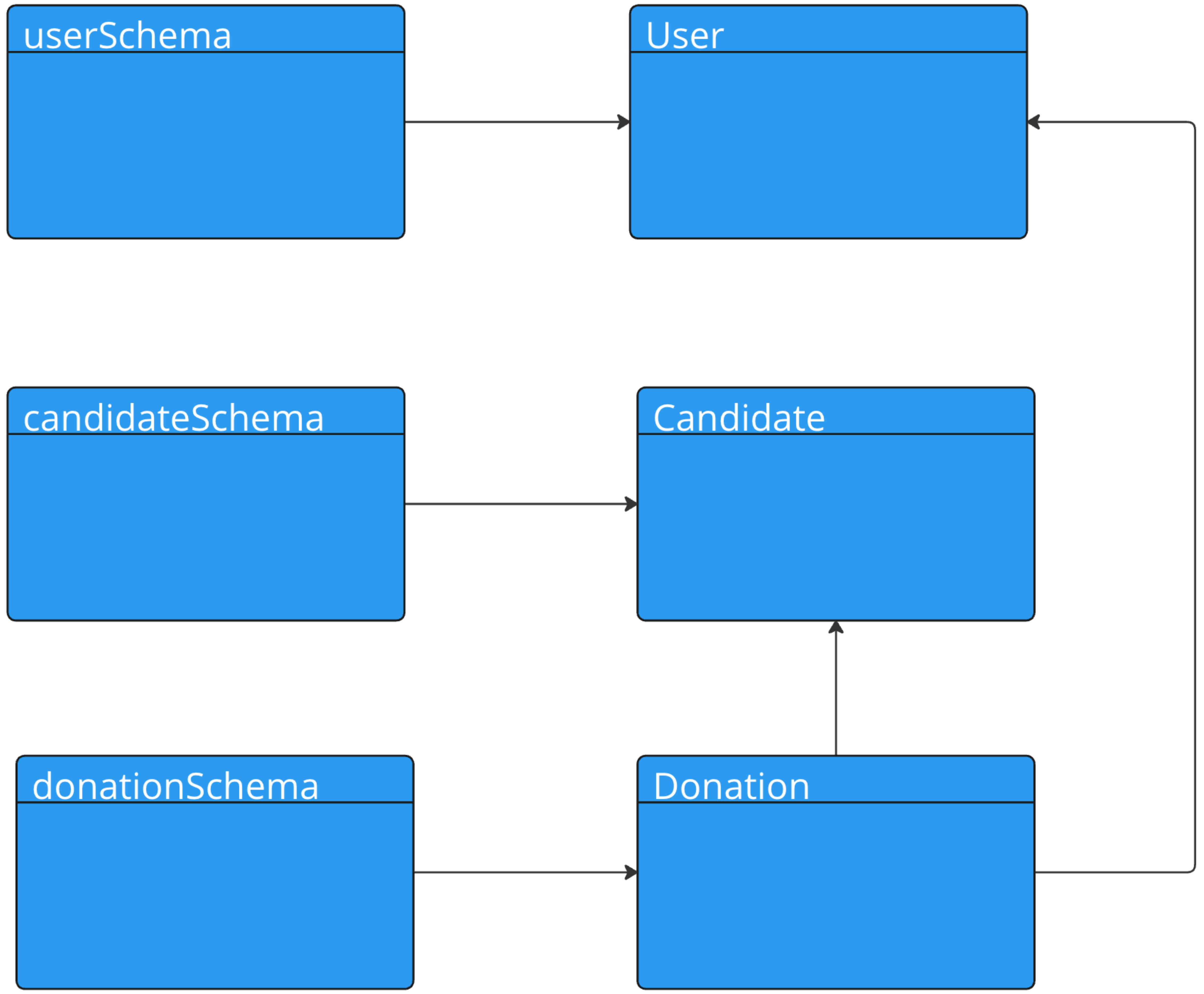
```
export interface User {
  firstName: string;
  lastName: string;
  email: string;
  password: string;
  _id: string;
}

export interface Candidate {
  firstName: string;
  lastName: string;
  office: string;
  _id: string;
}

export interface Donation {
  amount: number;
  method: string;
  candidate: Candidate | string;
  donor: User | string;
  lat: number;
  lng: number;
}
```



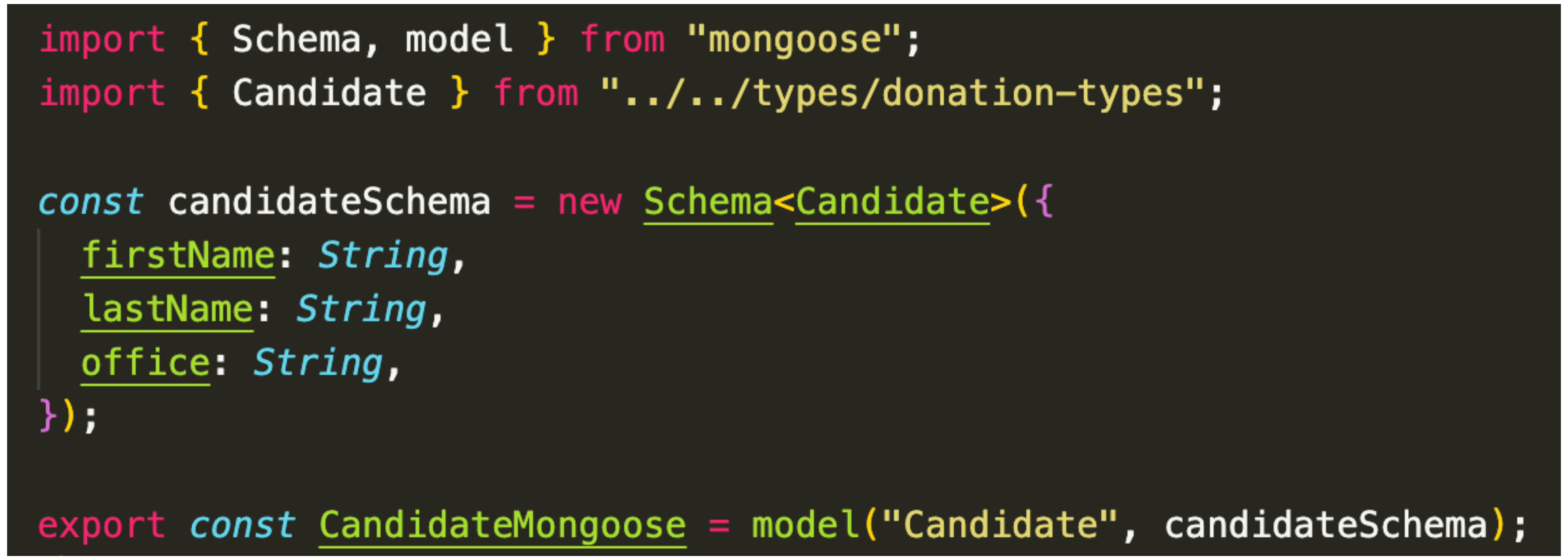
Mongoose userSchema



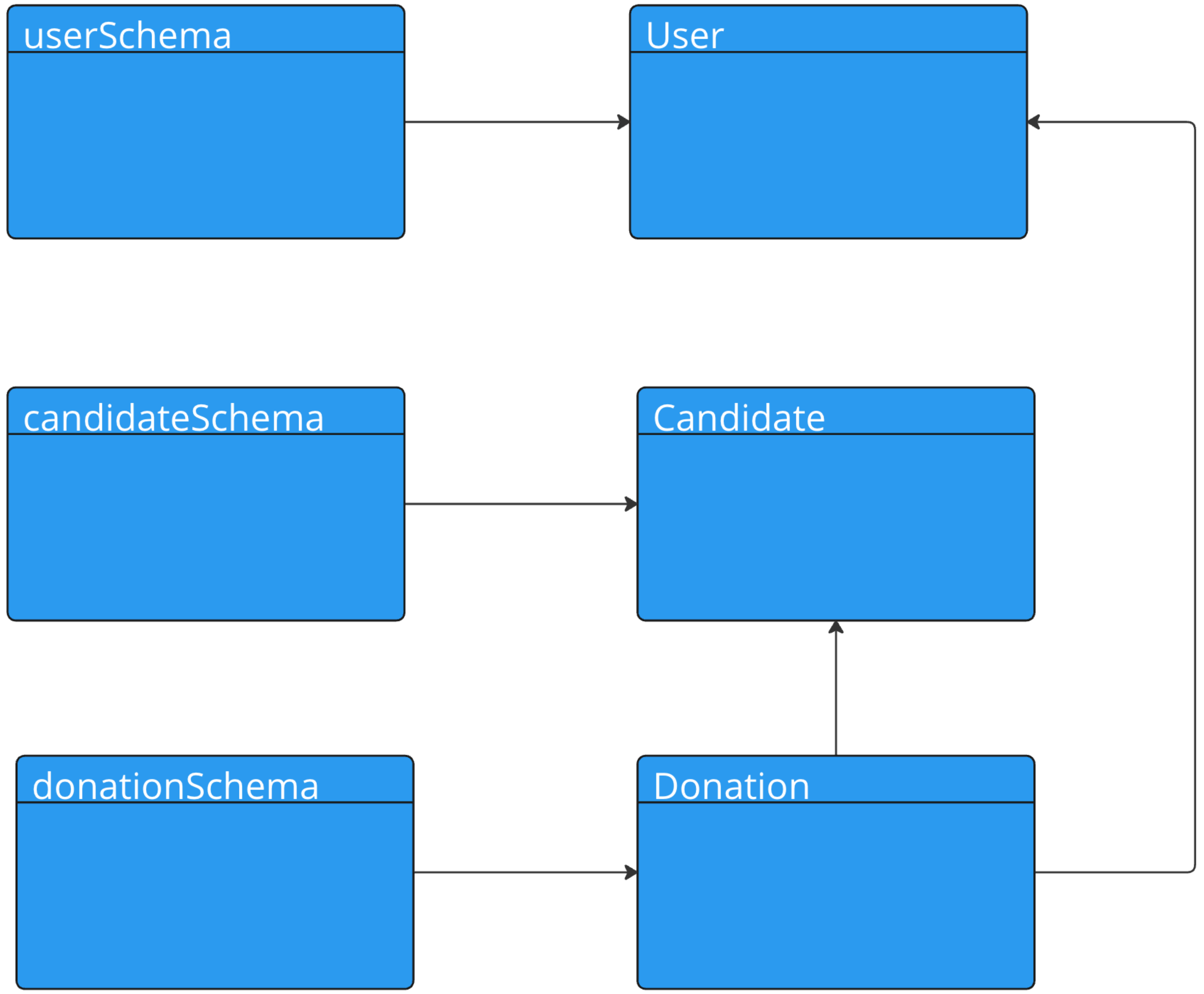
```
import { Schema, model } from "mongoose";
import { User } from "../../types/donation-types";

const userSchema = new Schema<User>({
  firstName: String,
  lastName: String,
  email: String,
  password: String,
});

export const UserMongoose = model("User", userSchema);
```



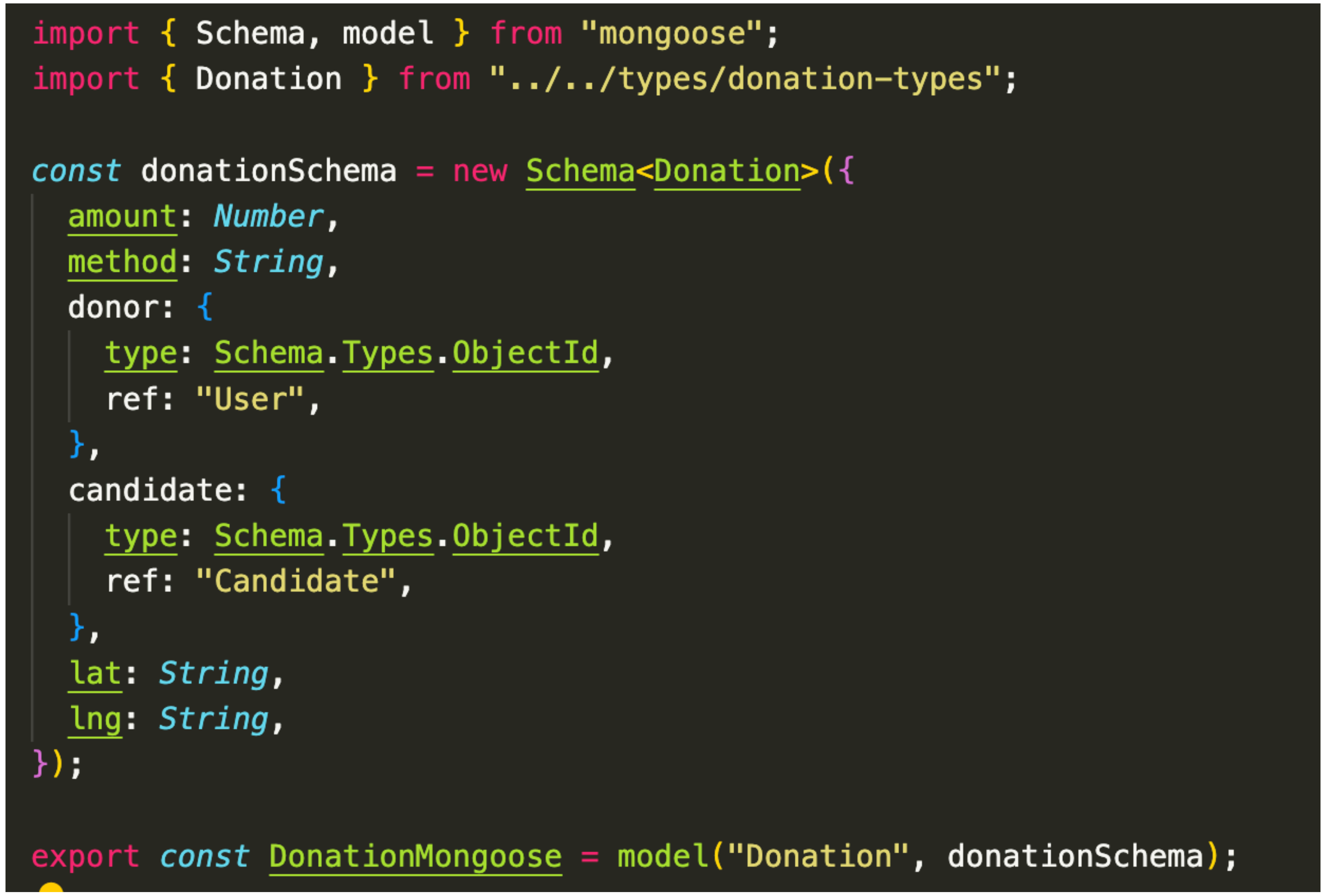
Mongoose candidateSchema



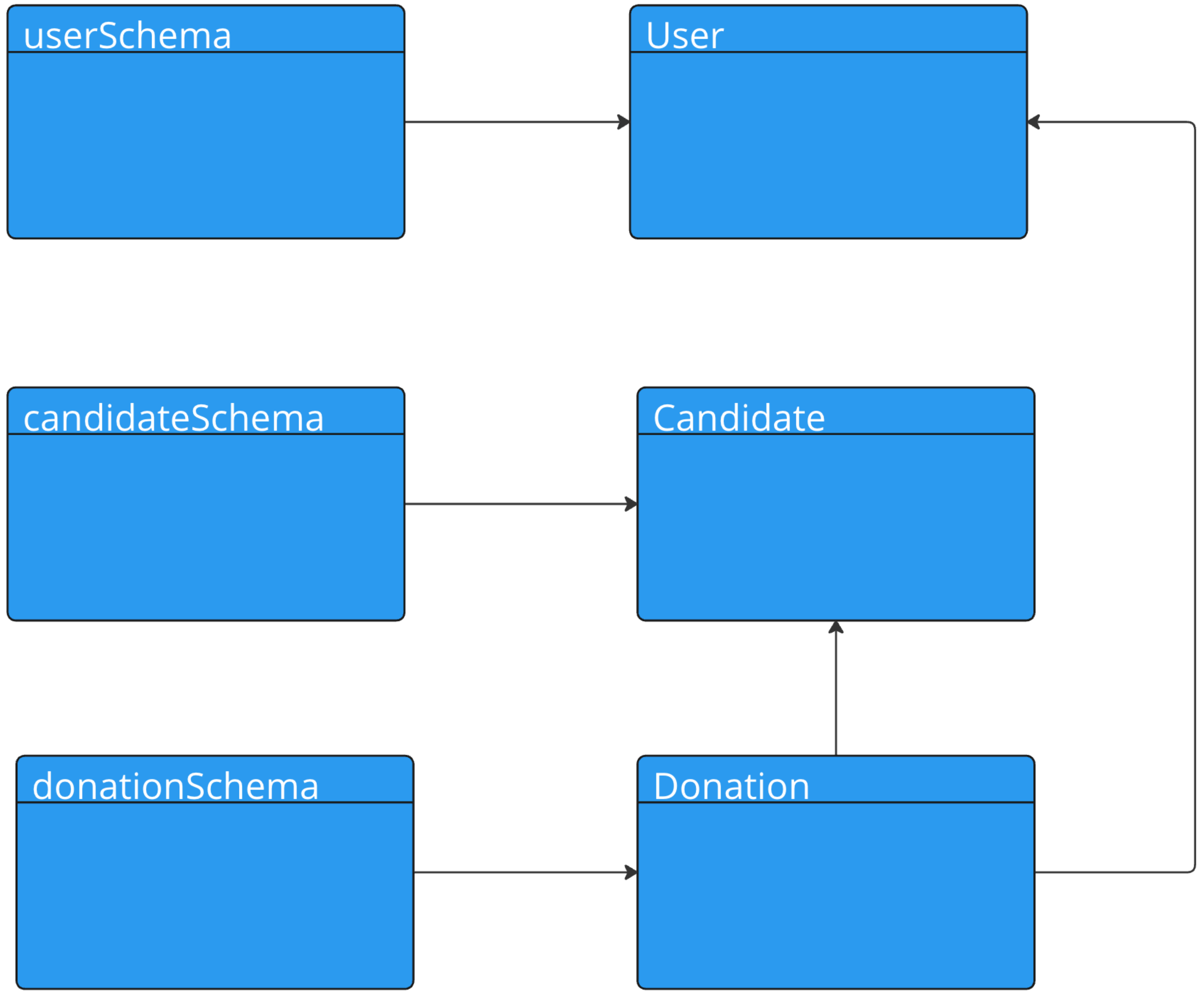
```
import { Schema, model } from "mongoose";
import { Candidate } from "../../types/donation-types";

const candidateSchema = new Schema<Candidate>({
  firstName: String,
  lastName: String,
  office: String,
});

export const CandidateMongoose = model("Candidate", candidateSchema);
```



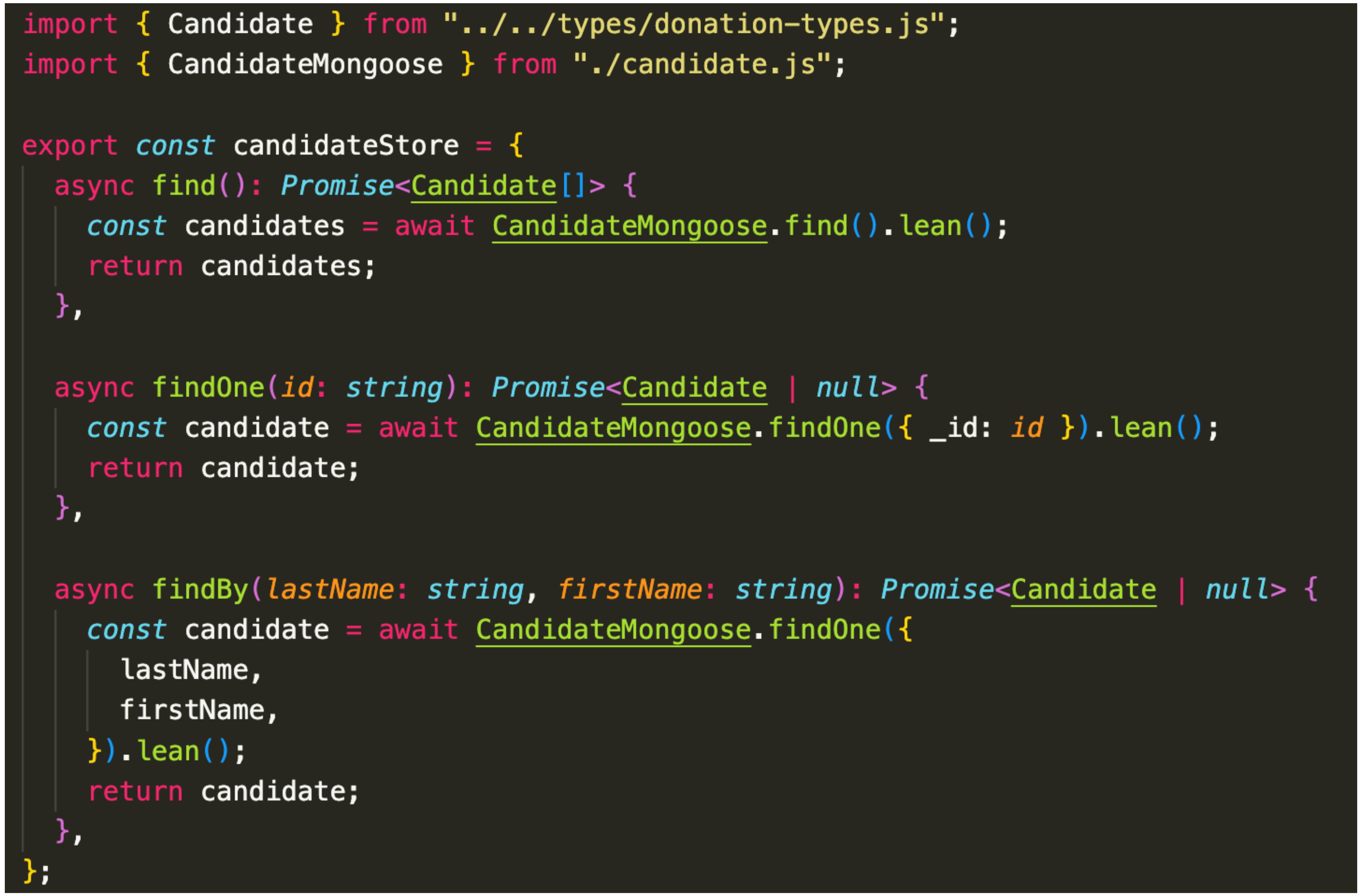
Mongoose donationSchema



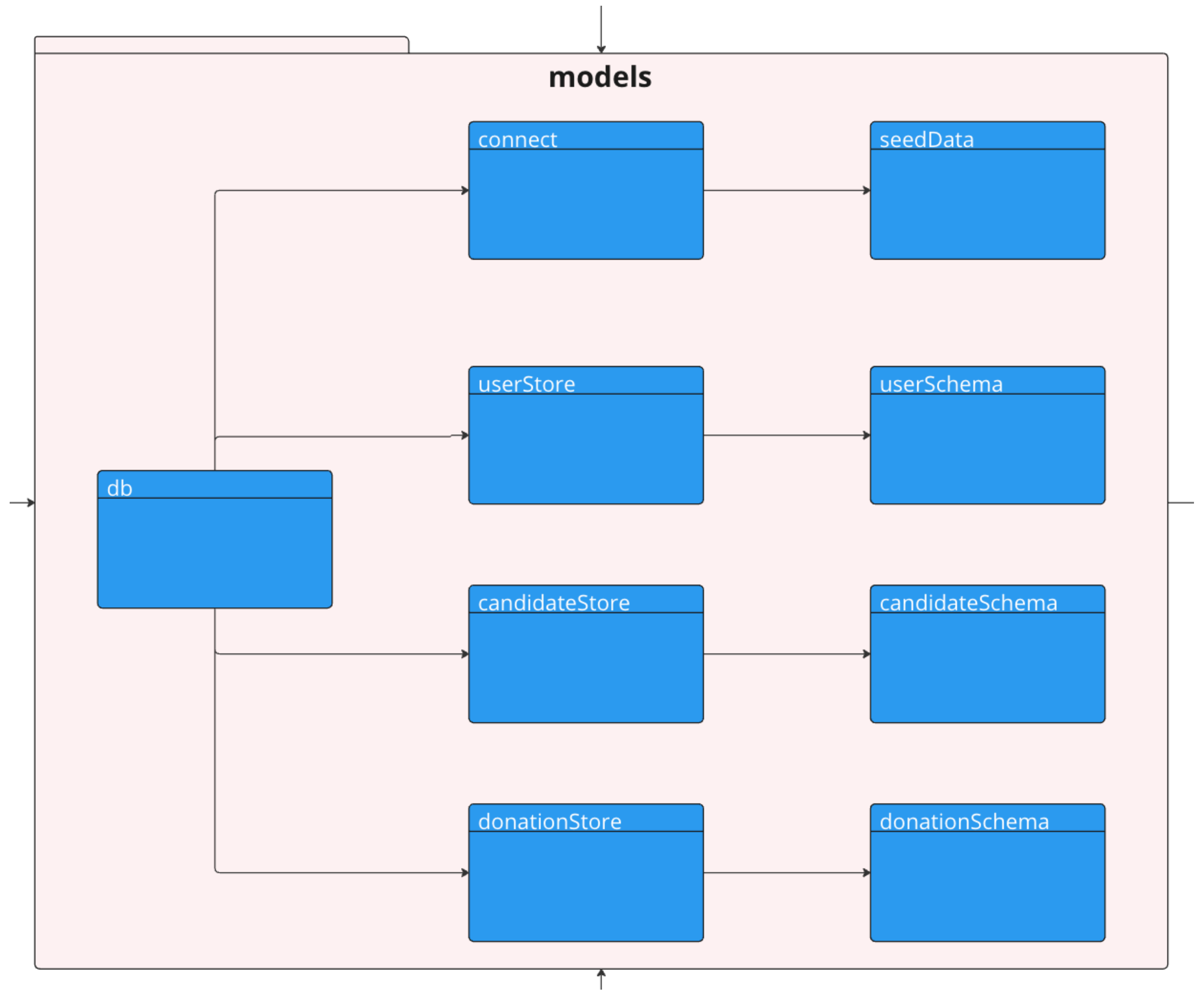
```
import { Schema, model } from "mongoose";
import { Donation } from "../../types/donation-types";

const donationSchema = new Schema<Donation>({
  amount: Number,
  method: String,
  donor: {
    type: Schema.Types.ObjectId,
    ref: "User",
  },
  candidate: {
    type: Schema.Types.ObjectId,
    ref: "Candidate",
  },
  lat: String,
  lng: String,
});

export const DonationMongoose = model("Donation", donationSchema);
```



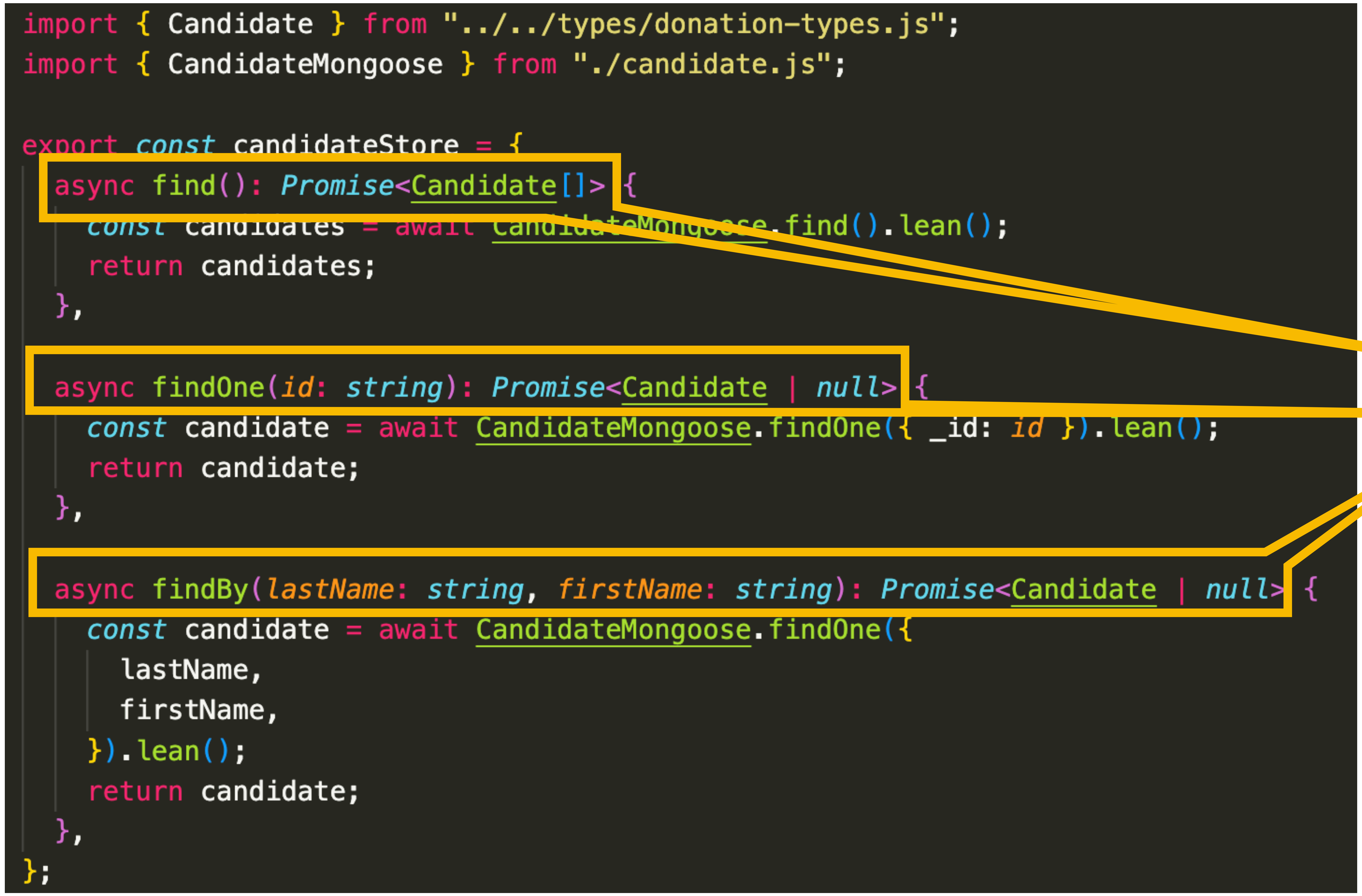
candidateStore



```
import { Candidate } from "../../types/donation-types.js";
import { CandidateMongoose } from "./candidate.js";

export const candidateStore = {
  async find(): Promise<Candidate[]> {
    const candidates = await CandidateMongoose.find().lean();
    return candidates;
  },

  async findOne(id: string): Promise<Candidate | null> {
    const candidate = await CandidateMongoose.findOne({ _id: id }).lean();
    return candidate;
  },

  async findBy(lastName: string, firstName: string): Promise<Candidate | null> {
    const candidate = await CandidateMongoose.findOne({
      lastName,
      firstName,
    }).lean();
    return candidate;
  },
};
```



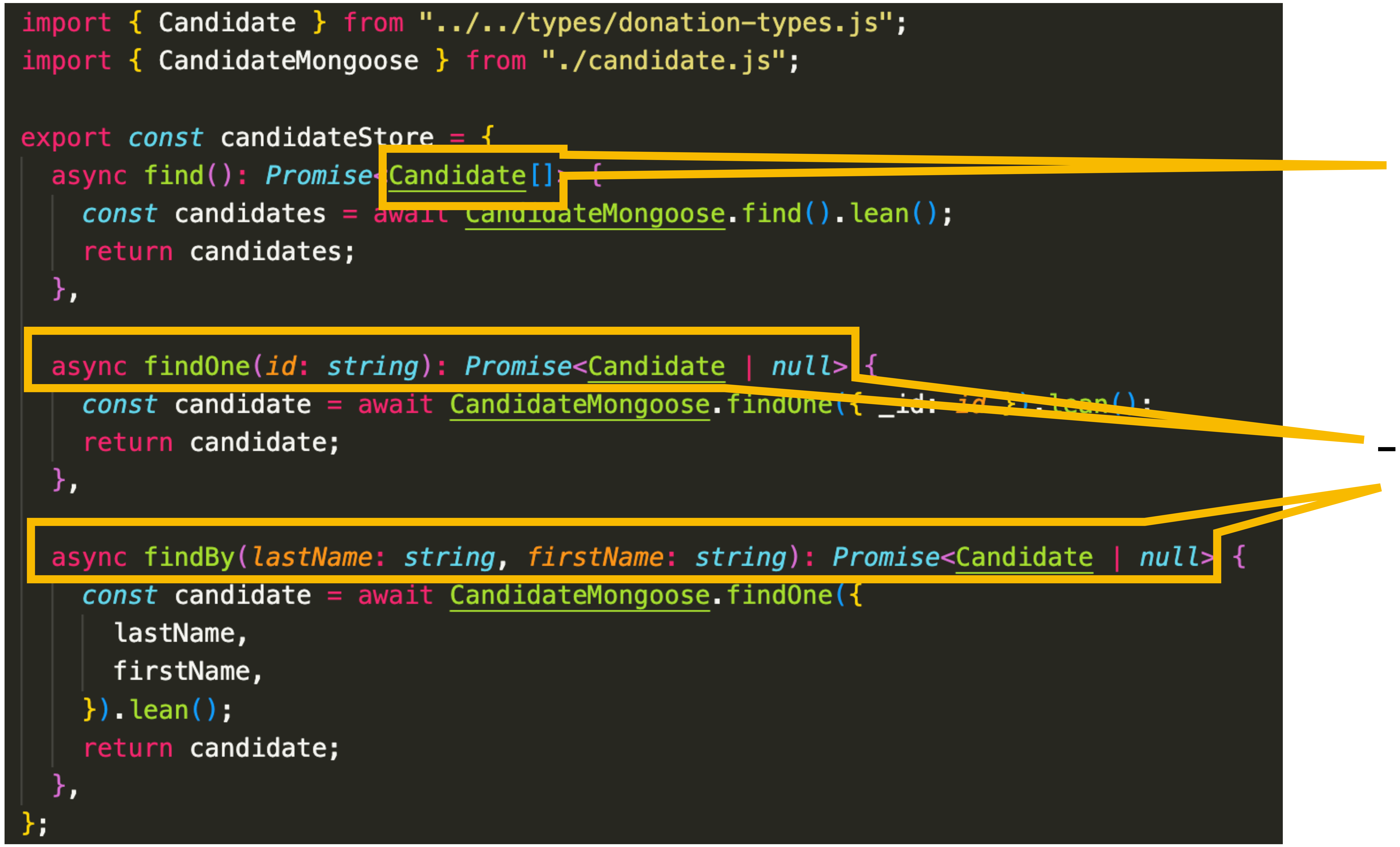
Async function Return Promises



```
import { Candidate } from "../../types/donation-types.js";
import { CandidateMongoose } from "./candidate.js";

export const candidateStore = {
  async find(): Promise<Candidate[]> {
    const candidates = await CandidateMongoose.find().lean();
    return candidates;
  },

  async findOne(id: string): Promise<Candidate | null> {
    const candidate = await CandidateMongoose.findOne({ _id: id }).lean();
    return candidate;
  },

  async findBy(lastName: string, firstName: string): Promise<Candidate | null> {
    const candidate = await CandidateMongoose.findOne({
      lastName,
      firstName,
    }).lean();
    return candidate;
  },
};
```

- Async functions always return Promises

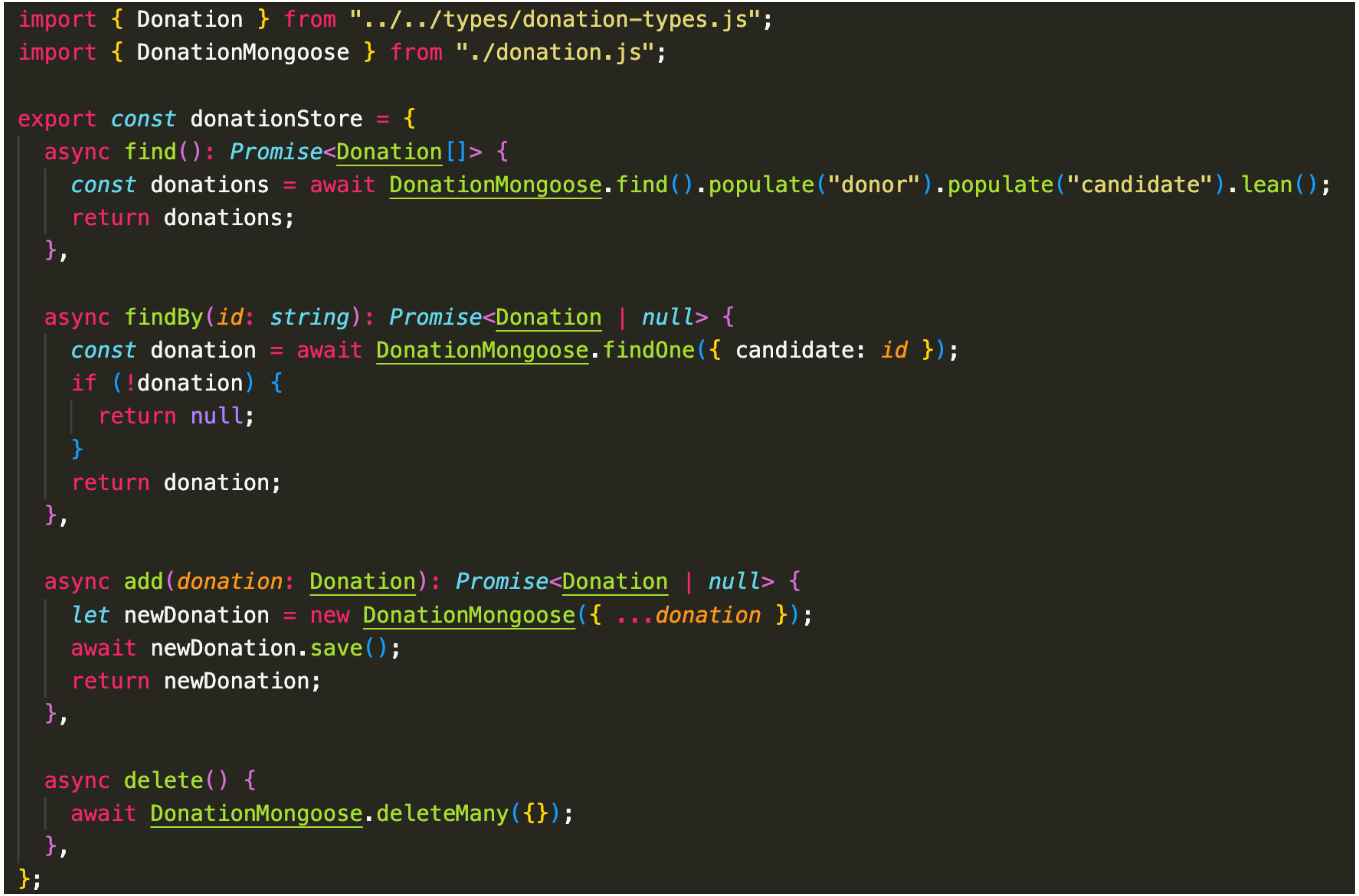
Promises wrap return type



```
import { Candidate } from "../../types/donation-types.js";
import { CandidateMongoose } from "./candidate.js";

export const candidateStore = {
  async find(): Promise<Candidate[]> {
    const candidates = await CandidateMongoose.find().lean();
    return candidates;
  },

  async findOne(id: string): Promise<Candidate | null> {
    const candidate = await CandidateMongoose.findOne({ _id: id }).lean();
    return candidate;
  },

  async findBy(lastName: string, firstName: string): Promise<Candidate | null> {
    const candidate = await CandidateMongoose.findOne({
      lastName,
      firstName,
    }).lean();
    return candidate;
  },
};
```

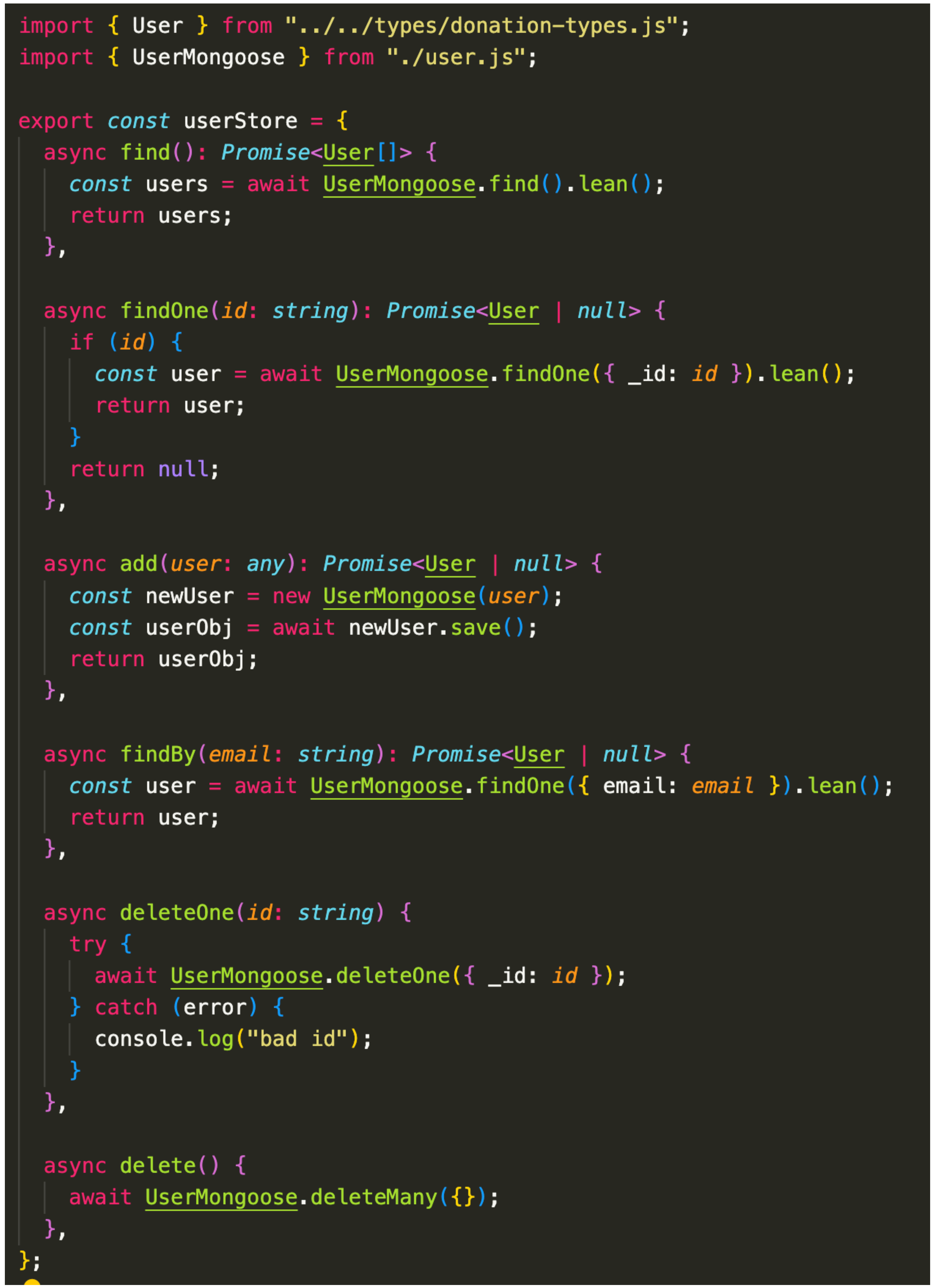
- Array of Candidates
- A single Candidate or Null

donationStore



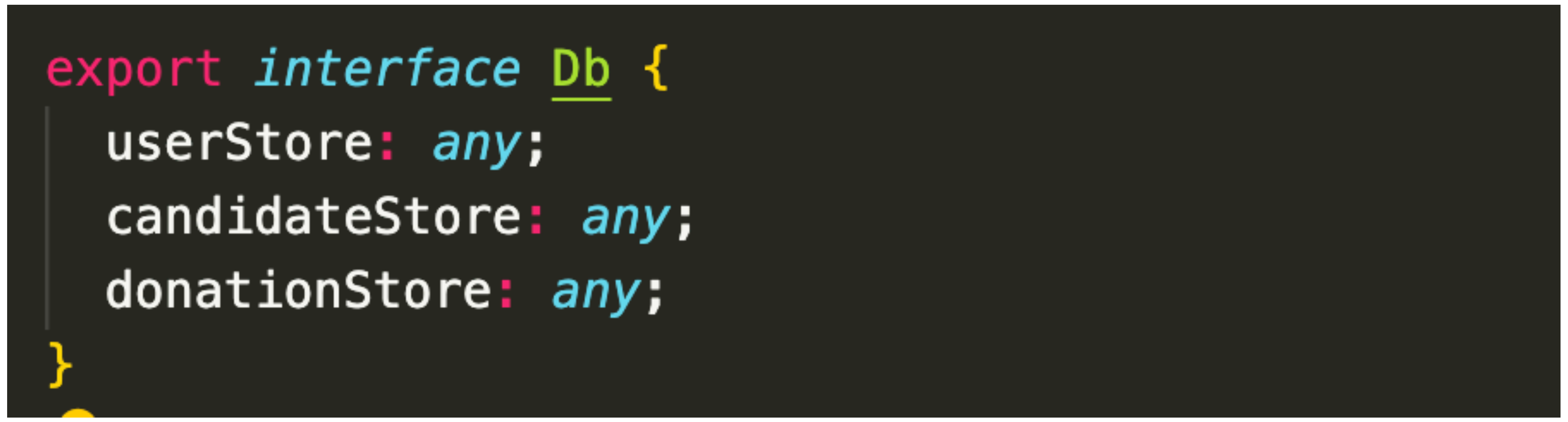
```
import { Donation } from "../../types/donation-types.js";
import { DonationMongoose } from "./donation.js";

export const donationStore = {
  async find(): Promise<Donation[]> {
    const donations = await DonationMongoose.find().populate("donor").populate("candidate").lean();
    return donations;
  },

  async findBy(id: string): Promise<Donation | null> {
    const donation = await DonationMongoose.findOne({ candidate: id });
    if (!donation) {
      return null;
    }
    return donation;
  },

  async add(donation: Donation): Promise<Donation | null> {
    let newDonation = new DonationMongoose({ ...donation });
    await newDonation.save();
    return newDonation;
  },

  async delete() {
    await DonationMongoose.deleteMany({});
  },
};
```

userStore

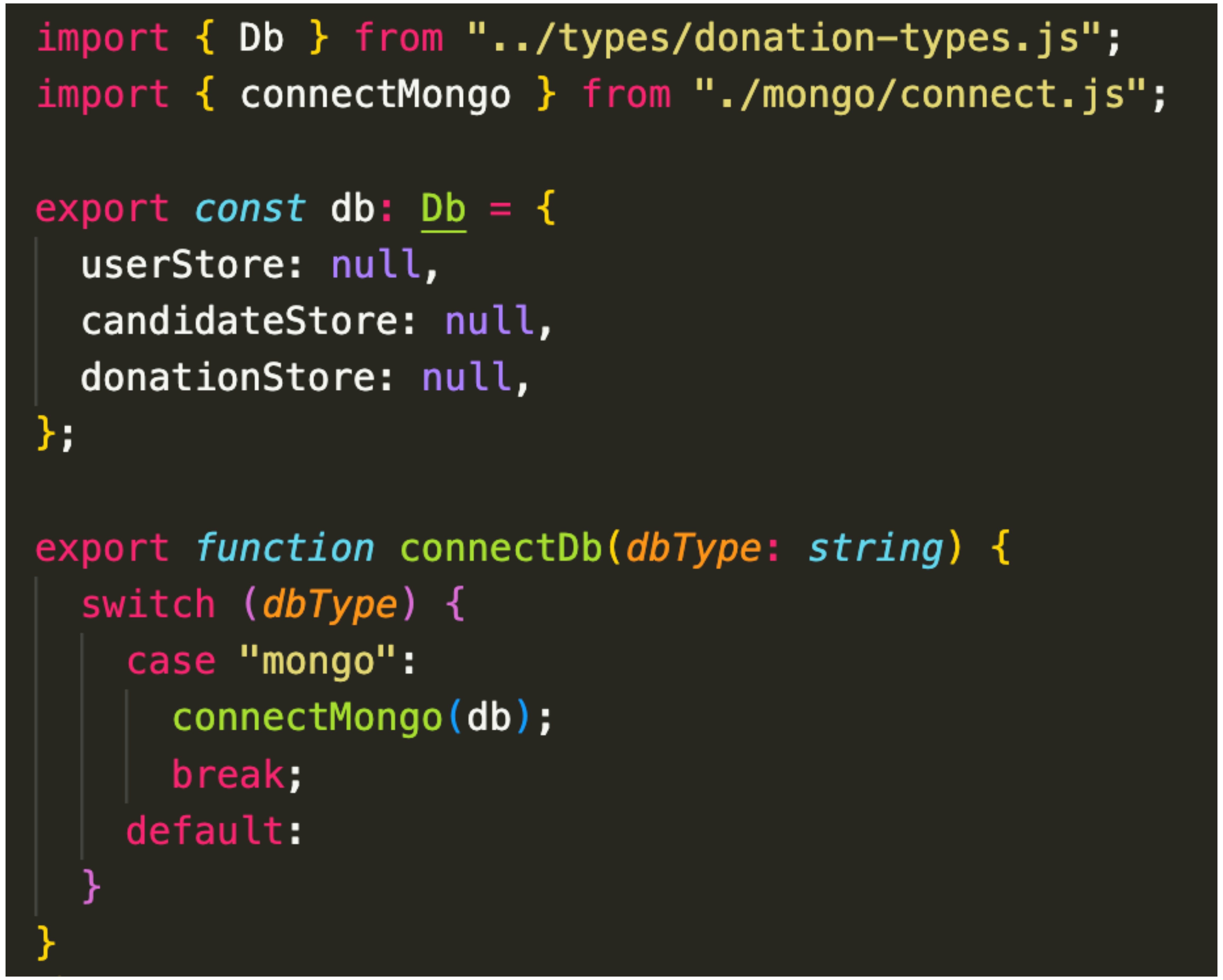


```
import { User } from "../../types/donation-types.js";
import { UserMongoose } from "./user.js";

export const userStore = {
  async find(): Promise<User[]> {
    const users = await UserMongoose.find().lean();
    return users;
  },

  async findOne(id: string): Promise<User | null> {
    if (id) {
      const user = await UserMongoose.findOne({ _id: id }).lean();
      return user;
    }
    return null;
  },

  async add(user: any): Promise<User | null> {
    const newUser = new UserMongoose(user);
    const userObj = await newUser.save();
    return userObj;
  },

  async findBy(email: string): Promise<User | null> {
    const user = await UserMongoose.findOne({ email: email }).lean();
    return user;
  },

  async deleteOne(id: string) {
    try {
      await UserMongoose.deleteOne({ _id: id });
    } catch (error) {
      console.log("bad id");
    }
  },

  async delete() {
    await UserMongoose.deleteMany({});
  },
};
```

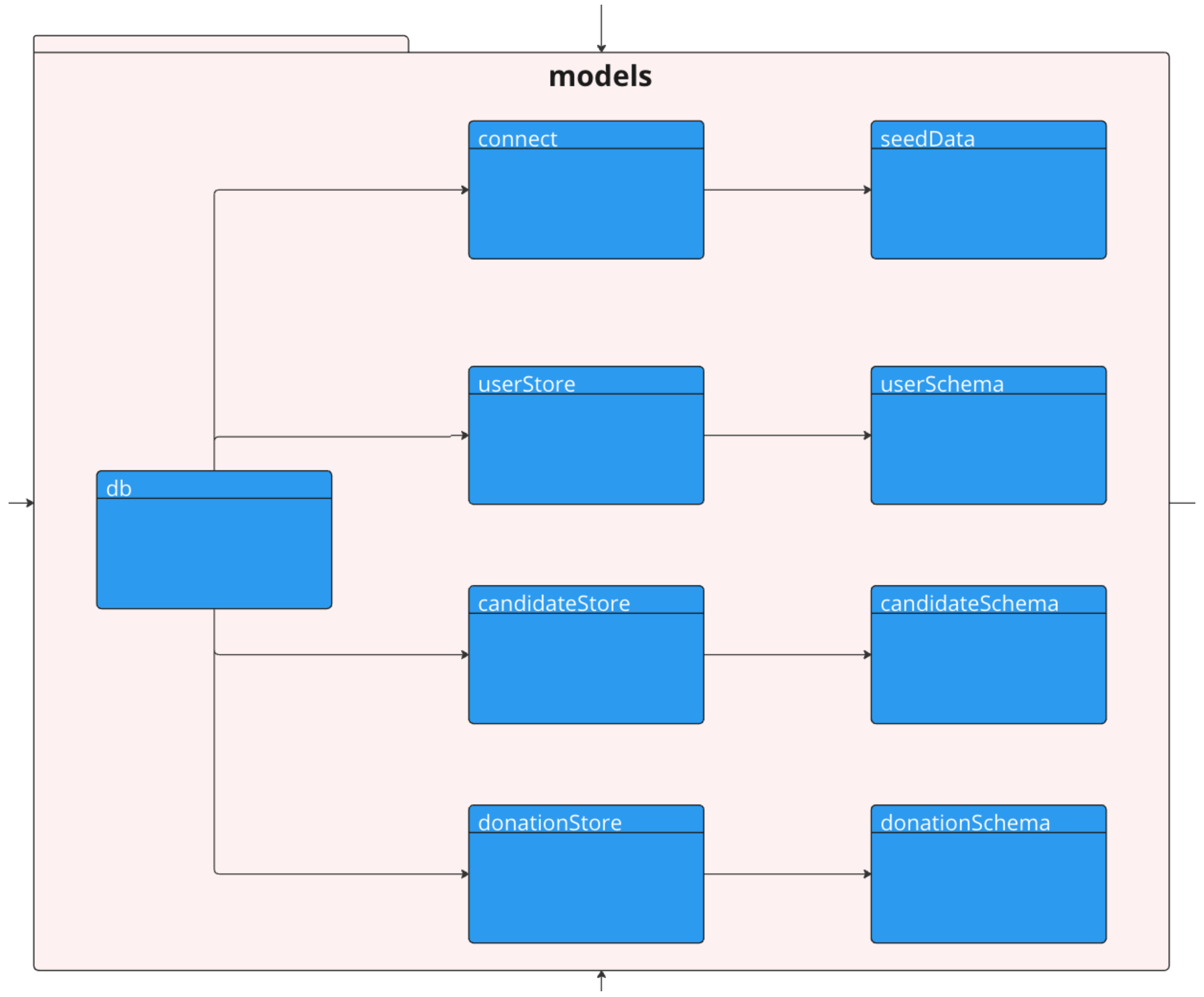
Db



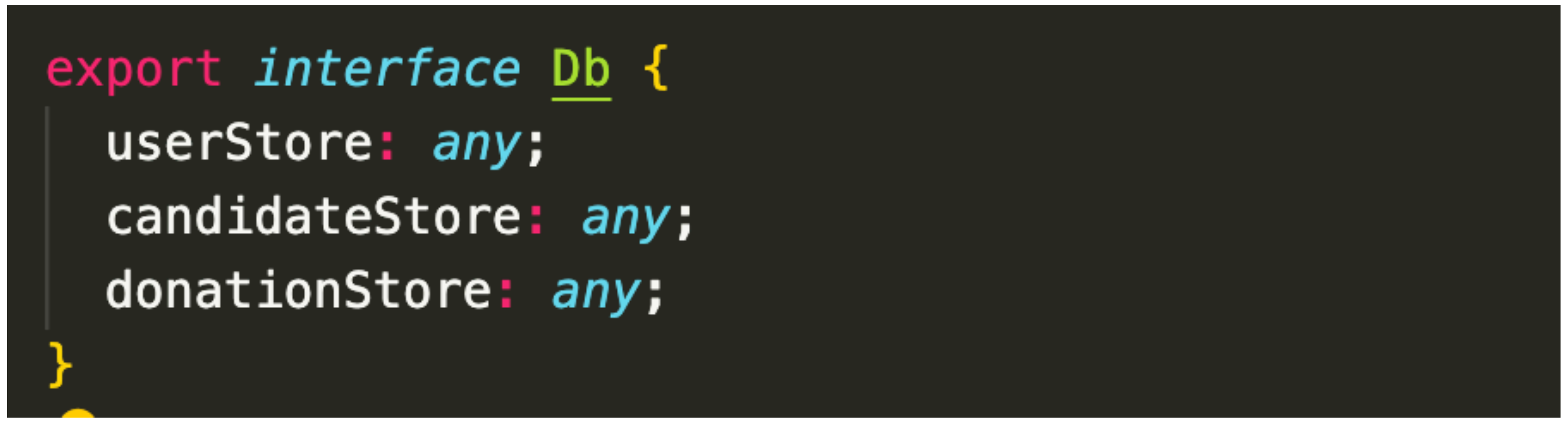
```
export interface Db {
  userStore: any;
  candidateStore: any;
  donationStore: any;
}
```



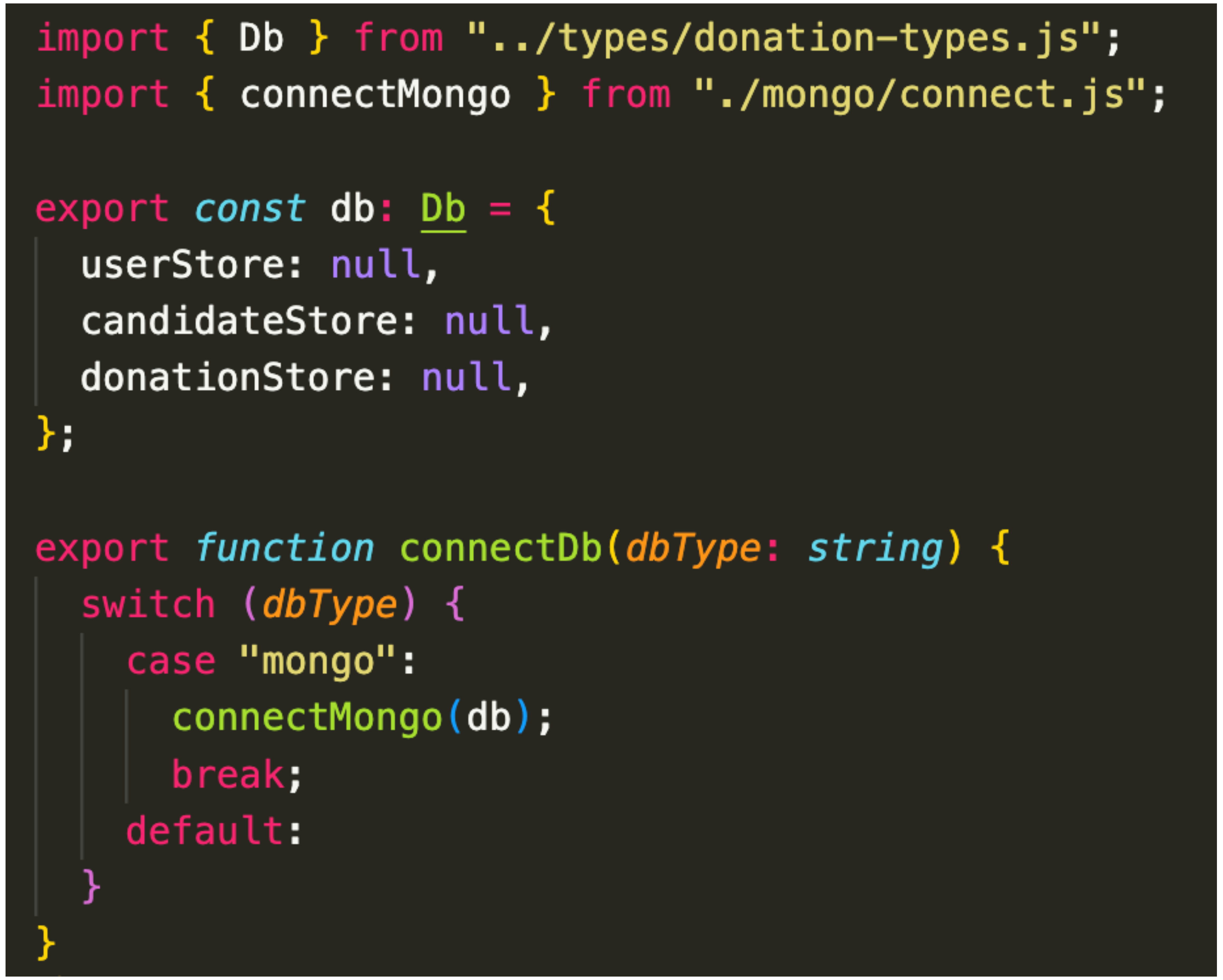
```
import { Db } from "../types/donation-types.js";
import { connectMongo } from "./mongo/connect.js";

export const db: Db = {
  userStore: null,
  candidateStore: null,
  donationStore: null,
};

export function connectDb(dbType: string) {
  switch (dbType) {
    case "mongo":
      connectMongo(db);
      break;
    default:
  }
}
```



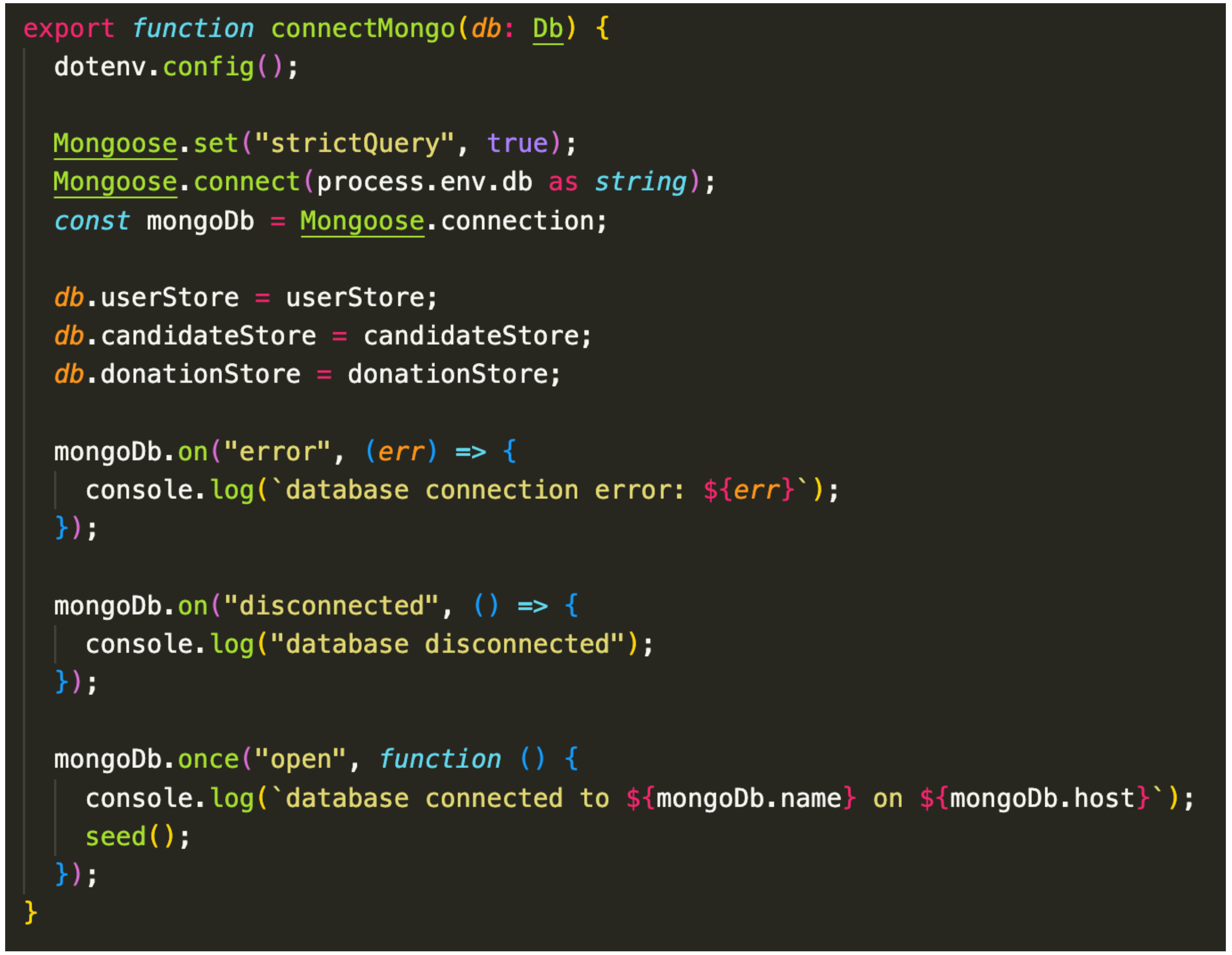
Db



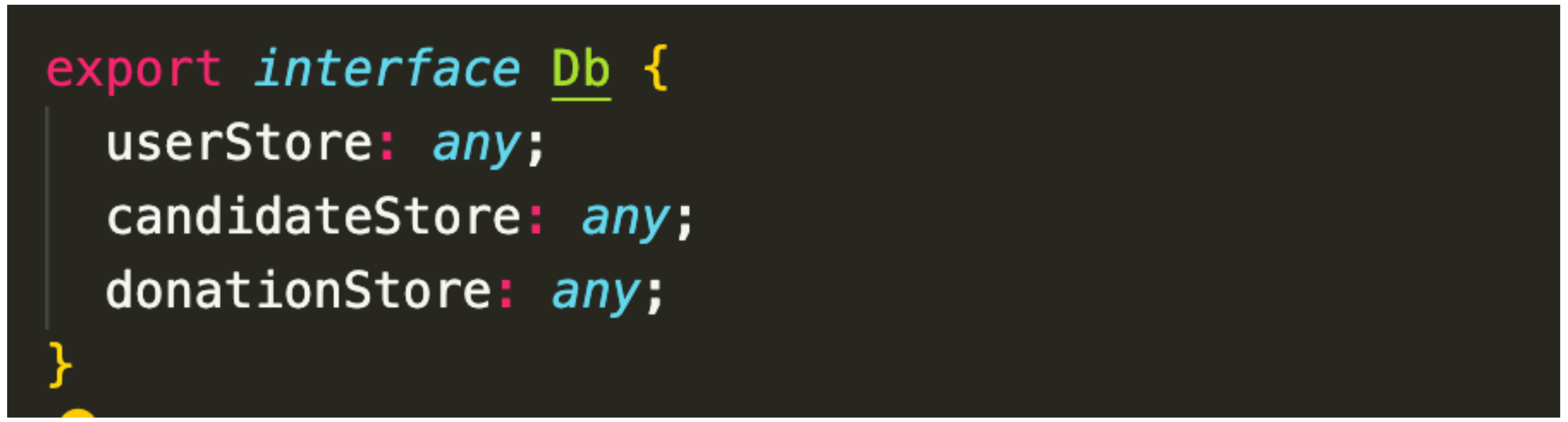
```
export interface Db {
  userStore: any;
  candidateStore: any;
  donationStore: any;
}
```



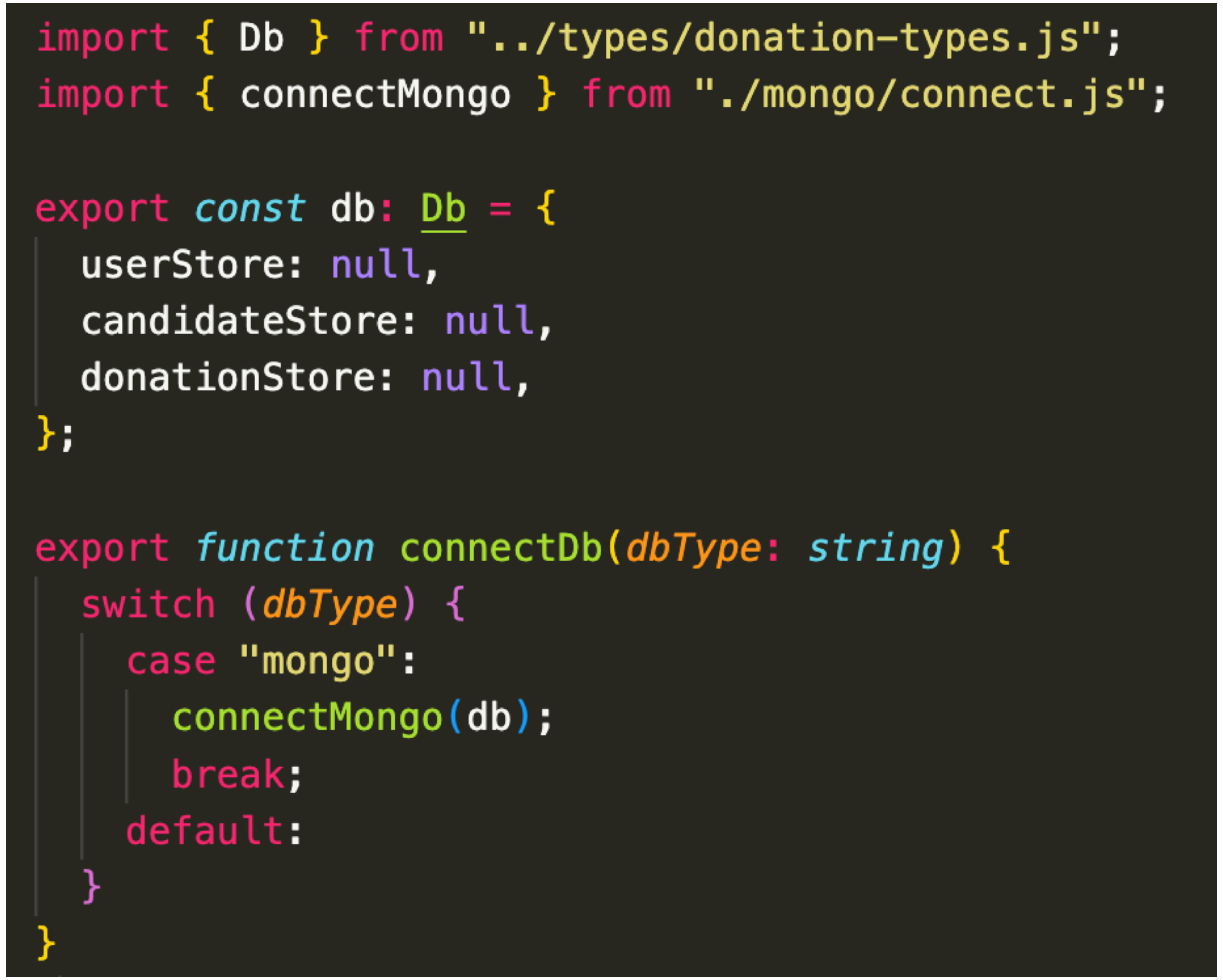
```
import { Db } from "../types/donation-types.js";
import { connectMongo } from "./mongo/connect.js";

export const db: Db = {
  userStore: null,
  candidateStore: null,
  donationStore: null,
};

export function connectDb(dbType: string) {
  switch (dbType) {
    case "mongo":
      connectMongo(db);
      break;
    default:
  }
}
```



```
export function connectMongo(db: Db) {
  dotenv.config();

  Mongoose.set("strictQuery", true);
  Mongoose.connect(process.env.db as string);
  const mongoDb = Mongoose.connection;

  db.userStore = userStore;
  db.candidateStore = candidateStore;
  db.donationStore = donationStore;

  mongoDb.on("error", (err) => {
    console.log(`database connection error: ${err}`);
  });

  mongoDb.on("disconnected", () => {
    console.log("database disconnected");
  });

  mongoDb.once("open", function () {
    console.log(`database connected to ${mongoDb.name} on ${mongoDb.host}`);
    seed();
  });
}
```



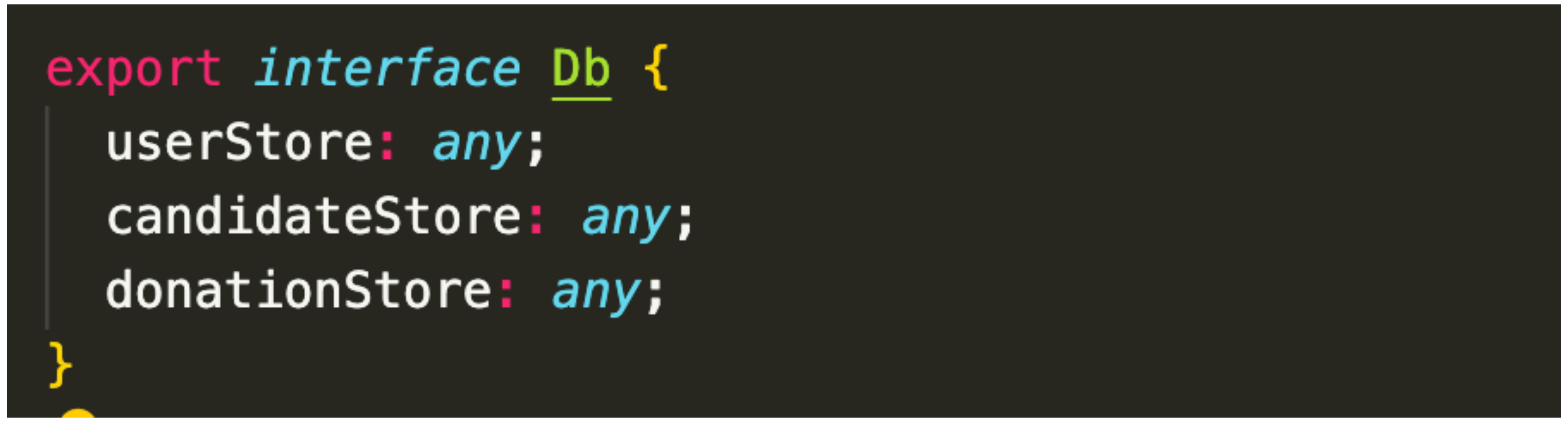
```
export interface Db {
  userStore: any;
  candidateStore: any;
  donationStore: any;
}
```



```
import { Db } from "../types/donation-types.js";
import { connectMongo } from "./mongo/connect.js";

export const db: Db = {
  userStore: null,
  candidateStore: null,
  donationStore: null,
};

export function connectDb(dbType: string) {
  switch (dbType) {
    case "mongo":
      connectMongo(db);
      break;
    default:
  }
}
```

Db

- Use **‘any’** to capture an unspecified object type
- Use sparingly, as it does not communicate any meaningful information
- Can be useful if we are incrementally introducing types



```
export interface Db {
  userStore: any;
  candidateStore: any;
  donationStore: any;
}
```

Db: TODO

- Define interfaces for UserStore, CandidateStore and DonationStore
- These could be three separate interfaces
- OR
- Generic Interfaces

Types

Introduce Types into Donation