# Axsem + AxcemEngine: Code Review & Refactor Plan *Generated on 2025-09-02 15:52:58*

**Scope - Front-end PWA (`axsem/axsem`) — React + Vite + Tailwind + TanStack Router + Swiper. - Back-end engine (`axcemengine/axcemengine`) — NestJS + Mongoose (MongoDB).**

**High-level Findings - Mixed TypeScript and JavaScript in both repos; duplicate entry files (`main.tsx` & `main.jsx`). - Hard-coded API base URLs found in **12 files**; recommend `.env` via Vite and centralized HTTP client. - Direct `axios` calls inside **3 components**; suggest React Query + service layer. - Controller injects Mongoose model directly (leaky layering); move all data access into services. - No request validation in controllers; add DTOs with `class-validator` and enable global `ValidationPipe`. - Static assets mixed into the backend; consider CDN/static hosting.**

**Project Structure Recommendations **Front-end (`axsem/axsem`)**: ``` src/ app/ router.tsx # TanStack Router config providers/Telemetry.tsx # Telemetry context/provider pages/ # Route components components/ # Reusable UI services/ apiClient.ts # Axios instance personApi.ts # Typed API calls types/ person.ts # Shared types hooks/ usePerson.ts # React Query hooks (usePerson, useRelatives, ...) ``` **Back-end (`axcemengine/axcemengine`)**: ``` src/ config/ configuration.ts # ConfigModule & typed env common/ interceptors/ filters/ pipes/ guards/ person/ dto/ schemas/ person.controller.ts person.service.ts person.module.ts main.ts app.module.ts ```**

**Front-end (axsem) — Detailed Suggestions - Adopt **TypeScript-only** (`.ts/.tsx`) and remove duplicate `.jsx` files. Keep a single `main.tsx`. - Centralize HTTP with an Axios instance and env-driven base URL. - Use **TanStack Query** for caching, retries, loading/error states, and prefetching. - Replace hard-coded `http://localhost:3000` with `import.meta.env.VITE_API_BASE_URL`. - Extract `PersonCard` into a pure, memoized component and add skeleton states. - Use **error boundaries** and suspense (``) for routes. - Fix className typos (e.g.,**

**`PersonDetaisCard` → `PersonDetailsCard`) and invalid props like `flex-1` as standalone props. - Split large components (`PreferedFamilyTreeSlider`) into smaller files: data fetching hook, presenter component, slide item. - Accessibility: add `alt` text to images, correct roles, and keyboard navigation for Swiper. - Testing: Vitest + Testing Library for components; Playwright/Cypress for smoke navigation flows.**

***Front-end Code Samples **Centralized Axios + React Query Hook****

```
// src/services/apiClient.ts
import axios from "axios";

export const api = axios.create({
baseURL: import.meta.env.VITE_API_BASE_URL, // e.g., http://localhost:3000
withCredentials: false,
timeout: 15000,
});

// src/services/personApi.ts
import { api } from "./apiClient";
import { Person } from "@/types/person";

export async function getPerson(id: string): Promise {
const { data } = await api.get(`/api/persons/${id}`);
return data;
}

// src/hooks/usePerson.ts
import { useQuery } from "@tanstack/react-query";
import { getPerson } from "@/services/personApi";

export function usePerson(id: string) {
return useQuery({ queryKey: ["person", id], queryFn: () => getPerson(id) });
}



// src/components/PreferredFamilyTree/Presenter.tsx
import React from "react";
import { Swiper, SwiperSlide } from "swiper/react";
import "swiper/css";
import { PersonCard } from "./PersonCard";
```

```
export function PreferredFamilyTreePresenter({ people, onSlideChange }: {
people: Array<{ id: string } & any>,
onSlideChange?: (index: number) => void
}) {
return (
onSlideChange?.(s.activeIndex)}>
{people.map((p) => (



))}

);
}
```

### *Back-end Code Samples ```ts*

```
// src/app.module.ts
import { Module } from "@nestjs/common";
import { ConfigModule } from "@nestjs/config";
import { MongooseModule } from "@nestjs/mongoose";
import { PersonModule } from "./person/person.module";
import * as Joi from "joi";

@Module({
imports: [
ConfigModule.forRoot({
isGlobal: true,
validationSchema: Joi.object({
MONGO_URI: Joi.string().uri().required(),
PORT: Joi.number().default(3000),
}),
}),
MongooseModule.forRoot(process.env.MONGO_URI),
PersonModule,
],
})
export class AppModule {}


// src/main.ts
import { NestFactory } from "@nestjs/core";
import { AppModule } from "./app.module";
import { ValidationPipe } from "@nestjs/common";
import { SwaggerModule, DocumentBuilder } from "@nestjs/swagger";
import helmet from "helmet";
```

```
async function bootstrap() {
const app = await NestFactory.create(AppModule);
app.use(helmet());
app.enableCors({ origin: process.env.CORS_ORIGIN?.split(",") ?? true });
app.useGlobalPipes(new ValidationPipe({ whitelist: true, forbidNonWhitelisted: true, transform: true }));

const config = new DocumentBuilder().setTitle("AXCEM API").setVersion("1.0").build();
const doc = SwaggerModule.createDocument(app, config);
SwaggerModule.setup("api-docs", app, doc);

await app.listen(process.env.PORT || 3000);
}
bootstrap();


// src/person/person.controller.ts
import { Controller, Get, Param, Post, Body } from "@nestjs/common";
import { ApiTags } from "@nestjs/swagger";
import { PersonService } from "./person.service";
import { CreatePersonDto } from "./dto/create-person.dto";

@ApiTags("Persons")
@Controller("api/persons")
export class PersonController {
constructor(private readonly persons: PersonService) {}

@Post()
create(@Body() dto: CreatePersonDto) {
return this.persons.create(dto);
}

@Get(":id")
findOne(@Param("id") id: string) {
return this.persons.findOne(id);
}

@Get(":id/children")
children(@Param("id") id: string) {
return this.persons.findChildren(id);
}
}


// src/person/person.service.ts
import { Injectable, NotFoundException } from "@nestjs/common";
import { InjectModel } from "@nestjs/mongoose";
import { Model, Types } from "mongoose";
import { Person, PersonDocument } from "./schemas/person.schema";
```

```
@Injectable()
export class PersonService {
constructor(@InjectModel(Person.name) private personModel: Model) {}

create(dto: Partial) {
return this.personModel.create(dto);
}

async findOne(id: string) {
const p = await this.personModel.findById(id).lean();
if (!p) throw new NotFoundException("Person not found");
return p;
}

async findChildren(id: string) {
const objId = new Types.ObjectId(id);
return this.personModel.find({ $or: [{ MOTHERID: objId }, { FATHERID: objId }] }).lean();
}
}
```

**Telemetry & Logging - Wrap telemetry in a provider that batches events and falls back to `console` in dev. - Redact PII (ID numbers, phone) before sending telemetry.**

**Developer Experience - ESLint + Prettier with a strict config. Enforce import aliases with TS path mapping. - Storybook for component docs (PersonCard, Content cards). - Generate OpenAPI types on the frontend using `openapi-typescript` to keep DTOs in sync.**

**Actionable Checklist - Frontend: remove `.jsx` duplicates; fix typos (`Preferred`, `PersonDetailsCard`). - Create `apiClient.ts` and migrate component `axios` calls into `personApi.ts`. - Install `@tanstack/react-query` and use hooks for data fetching states. - Replace hard-coded URLs with `VITE_API_BASE_URL` and `VITE_IMAGE_BASE_URL`. - Backend: move model injection out of controller; add DTOs and ValidationPipe. - Add indexes: MOTHERID, FATHERID, IDNUM; audit slow queries with Mongo logs. - Add CI: lint, type-check, unit tests, and E2E smoke tests on each push.**

```
src/
  app/
    router.tsx              # TanStack Router config
    providers/Telemetry.tsx  # Telemetry context/provider
```

```
  pages/                      # Route components
  components/                 # Reusable UI
  services/
    apiClient.ts              # Axios instance
    personApi.ts              # Typed API calls
  types/
    person.ts                 # Shared types
  hooks/
    usePerson.ts              # React Query hooks (usePerson, useRelatives, ...)
```

**Back-end (`axcemengine/axcemengine`)**:

```
src/
  config/
    configuration.ts          # ConfigModule & typed env
  common/
    interceptors/
    filters/
    pipes/
    guards/
  person/
    dto/
    schemas/
    person.controller.ts
    person.service.ts
    person.module.ts
  main.ts
  app.module.ts
```

## Front-end (axsem) — Detailed Suggestions

- Adopt **TypeScript-only** (`.ts/.tsx`) and remove duplicate `.jsx` files. Keep a single `main.tsx`.

- Centralize HTTP with an Axios instance and env-driven base URL.

- Use **TanStack Query** for caching, retries, loading/error states, and prefetching.

- Replace hard-coded `http://localhost:3000` with `import.meta.env.VITE_API_BASE_URL`.

- Extract `PersonCard` into a pure, memoized component and add skeleton states.

- Use **error boundaries** and suspense (`<React.Suspense>`) for routes.

- Fix className typos (e.g., `PersonDetaisCard` → `PersonDetailsCard`) and invalid props like `flex-1`

- Split large components (`PreferedFamilyTreeSlider`) into smaller files: data fetching hook, presenter

- Accessibility: add `alt` text to images, correct roles, and keyboard navigation for Swiper.

- Testing: Vitest + Testing Library for components; Playwright/Cypress for smoke navigation flows.

### Front-end Code Samples

**Centralized Axios + React Query Hook**

```
// src/services/apiClient.ts
import axios from "axios";

export const api = axios.create({
  baseURL: import.meta.env.VITE_API_BASE_URL, // e.g., http://localhost:3000
  withCredentials: false,
  timeout: 15000,
});

// src/services/personApi.ts
import { api } from "./apiClient";
import { Person } from "@/types/person";

export async function getPerson(id: string): Promise<Person> {
  const { data } = await api.get(`/api/persons/${id}`);
  return data;
}

// src/hooks/usePerson.ts
import { useQuery } from "@tanstack/react-query";
import { getPerson } from "@/services/personApi";

export function usePerson(id: string) {
```

```
  return useQuery({ queryKey: ["person", id], queryFn: () => getPerson(id) });
}

**PreferredFamilyTreeSlider (Presenter pattern)**

// src/components/PreferredFamilyTree/Presenter.tsx
import React from "react";
import { Swiper, SwiperSlide } from "swiper/react";
import "swiper/css";
import { PersonCard } from "./PersonCard";

export function PreferredFamilyTreePresenter({ people, onSlideChange }: {
  people: Array<{ id: string } & any>,
  onSlideChange?: (index: number) => void
}) {
  return (
    <Swiper onSlideChange={(s) => onSlideChange?.(s.activeIndex)}>
      {people.map((p) => (
        <SwiperSlide key={p.id}>
          <PersonCard person={p} />
        </SwiperSlide>
      ))}
    </Swiper>
  );
}

## Back-end (axcemengine) — Detailed Suggestions

- Use **ConfigModule** (`@nestjs/config`) and typed config schema; remove hard-coded Mongo URIs.

- Enable global `ValidationPipe` with `whitelist` & `forbidNonWhitelisted` to sanitize inputs.

- Keep controllers thin: remove `@InjectModel` from controller; delegate to `PersonService`.

- Optimize queries with **indexes** on `IDNUM`, `MOTHERID`, `FATHERID` and use `.lean()` for read paths.

- Standardize IDs — prefer ObjectId references; avoid mixing `IDNUM` and ObjectId unless necessary.

- Add DTOs for endpoints with `class-validator` decorators (`@IsString`, `@IsOptional`, etc.).

- Use NestJS `Logger` instead of `console.log`; add an error filter for Mongo errors.

- Security: add `helmet`, `rate-limiter-flexible` or `@nestjs/throttler`, and CORS origin allowlist.

- Swagger: enhance with `@ApiOkResponse({ type: PersonDto })` and examples; organize tags.

- Serving static: keep images in `public/` but consider moving to CDN; avoid serving at root to prevent



### Back-end Code Samples

// src/app.module.ts
import { Module } from "@nestjs/common";
import { ConfigModule } from "@nestjs/config";
import { MongooseModule } from "@nestjs/mongoose";
import { PersonModule } from "./person/person.module";
import * as Joi from "joi";

@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
      validationSchema: Joi.object({
        MONGO_URI: Joi.string().uri().required(),
        PORT: Joi.number().default(3000),
      }),
    }),
    MongooseModule.forRoot(process.env.MONGO_URI),
    PersonModule,
  ],
})
export class AppModule {}



// src/main.ts
import { NestFactory } from "@nestjs/core";
import { AppModule } from "./app.module";
import { ValidationPipe } from "@nestjs/common";
```

```
import { SwaggerModule, DocumentBuilder } from "@nestjs/swagger";
import helmet from "helmet";

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.use(helmet());
  app.enableCors({ origin: process.env.CORS_ORIGIN?.split(",") ?? true });
  app.useGlobalPipes(new ValidationPipe({ whitelist: true, forbidNonWhitelisted: true, transform: true }

  const config = new DocumentBuilder().setTitle("AXCEM API").setVersion("1.0").build();
  const doc = SwaggerModule.createDocument(app, config);
  SwaggerModule.setup("api-docs", app, doc);

  await app.listen(process.env.PORT || 3000);
}
bootstrap();



// src/person/person.controller.ts
import { Controller, Get, Param, Post, Body } from "@nestjs/common";
import { ApiTags } from "@nestjs/swagger";
import { PersonService } from "./person.service";
import { CreatePersonDto } from "./dto/create-person.dto";

@ApiTags("Persons")
@Controller("api/persons")
export class PersonController {
  constructor(private readonly persons: PersonService) {}

  @Post()
  create(@Body() dto: CreatePersonDto) {
    return this.persons.create(dto);
  }

  @Get(":id")
  findOne(@Param("id") id: string) {
    return this.persons.findOne(id);
  }

  @Get(":id/children")
  children(@Param("id") id: string) {
    return this.persons.findChildren(id);
  }
}



// src/person/person.service.ts
import { Injectable, NotFoundException } from "@nestjs/common";
import { InjectModel } from "@nestjs/mongoose";
import { Model, Types } from "mongoose";
import { Person, PersonDocument } from "./schemas/person.schema";

@Injectable()
export class PersonService {
  constructor(@InjectModel(Person.name) private personModel: Model<PersonDocument>) {}

  create(dto: Partial<Person>) {
    return this.personModel.create(dto);
  }

  async findOne(id: string) {
    const p = await this.personModel.findById(id).lean();
    if (!p) throw new NotFoundException("Person not found");
    return p;
  }

  async findChildren(id: string) {
    const objId = new Types.ObjectId(id);
    return this.personModel.find({ $or: [{ MOTHERID: objId }, { FATHERID: objId }] }).lean();
  }
}

## MongoDB Data Modeling & Indexing

- Create compound indexes to speed up relative lookups: `{ MOTHERID: 1 }`, `{ FATHERID: 1 }`, `{ IDNUM:

- Normalize references to use `ObjectId` consistently. Keep `IDNUM` as a unique field if needed but not

- Return lean objects for read-heavy endpoints to reduce overhead: `.lean()`.
```

- For complex traversals (cousins, uncles), consider precomputing relations or using `$graphLookup` with

## Telemetry & Logging
- Wrap telemetry in a provider that batches events and falls back to `console` in dev.
- Redact PII (ID numbers, phone) before sending telemetry.

## Developer Experience
- ESLint + Prettier with a strict config. Enforce import aliases with TS path mapping.
- Storybook for component docs (PersonCard, Content cards).
- Generate OpenAPI types on the frontend using `openapi-typescript` to keep DTOs in sync.

## Actionable Checklist
- Frontend: remove `.jsx` duplicates; fix typos (`Preferred`, `PersonDetailsCard`).
- Create `apiClient.ts` and migrate component `axios` calls into `personApi.ts`.
- Install `@tanstack/react-query` and use hooks for data fetching states.
- Replace hard-coded URLs with `VITE_API_BASE_URL` and `VITE_IMAGE_BASE_URL`.
- Backend: move model injection out of controller; add DTOs and ValidationPipe.
- Add indexes: MOTHERID, FATHERID, IDNUM; audit slow queries with Mongo logs.
- Add CI: lint, type-check, unit tests, and E2E smoke tests on each push.