MERN TypeScript Multi-Step Form — Full Project

This repository implements a full-stack, multi-step form using the MERN stack with **TypeScript**, applying SOLID (OCP, LSP, SRP), OOP, low coupling & high cohesion, React Query, and a clean folder structure. No `any` types are used — all types and interfaces are defined.

Features

- Multi-step form (personal → address → documents → review & submit)
- File upload (handled via Multer, stored locally in `uploads/`) with ability to upload multiple files
- Backend: Express + TypeScript + Mongoose
- Frontend: React + TypeScript + React Query + Axios
- OOP Service & Repository layers
- Clean architecture and folder structure
- Responsive basic UI (Tailwind optional — code uses CSS module for simplicity)

Project structure (high-level)

```
multi-step-form/
├─ backend/
│  ├─ src/
│  │  ├─ config/
│  │  │  └─ db.ts
│  │  ├─ controllers/
│  │  │  └─ form.controller.ts
│  │  ├─ services/
│  │  │  └─ form.service.ts
│  │  ├─ repositories/
│  │  │  └─ form.repository.ts
│  │  ├─ models/
│  │  │  └─ form.model.ts
│  │  ├─ routes/
│  │  │  └─ form.routes.ts
│  │  ├─ utils/
│  │  │  └─ multer.ts
│  │  ├─ app.ts
│  │  └─ server.ts
│  └─ package.json
├─ frontend/
│  ├─ src/
│  │  ├─ api/
│  │  │  └─ apiClient.ts
│  │  ├─ services/
│  │  │  └─ formService.ts
│  │  ├─ types/
│  │  │  └─ form.ts
│   │   ├─ components/
│   │   │   ├─ formSteps/
│   │   │   │   ├─ Step1Personal.tsx
│   │   │   │   ├─ Step2Address.tsx
│   │   │   │   ├─ Step3Documents.tsx
│   │   │   │   └─ Step4Review.tsx
│   │   │   └─ MultiStepForm.tsx
│   │   ├─ hooks/
│   │   │   └─ useFormMutation.ts
│   │   ├─ App.tsx
│   │   ├─ main.tsx
│   │   └─ index.css
│   └─ package.json
└─ README.md
```

Backend code

backend/package.json

```json
{
  "name": "ms-form-backend",
  "version": "1.0.0",
  "main": "dist/server.js",
  "scripts": {
    "dev": "ts-node-dev --respawn --transpile-only src/server.ts",
    "build": "tsc",
    "start": "node dist/server.js"
  },
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.0.0",
    "express": "^4.18.2",
    "mongoose": "^7.0.0",
    "multer": "^1.4.5"
  },
  "devDependencies": {
    "ts-node-dev": "^2.0.0",
    "typescript": "^5.0.0",
    "@types/express": "^4.17.13",
    "@types/multer": "^1.4.7",
    "@types/node": "^18.0.0"
  }
}
```

backend/src/config/db.ts

```
import mongoose from 'mongoose';
import { config } from 'dotenv';
config();

const MONGO_URI: string = process.env.MONGO_URI ?? 'mongodb://localhost:
27017/msform';

export async function connectDB(): Promise<void> {
  await mongoose.connect(MONGO_URI);
  console.log('MongoDB connected');
}
```

backend/src/models/form.model.ts

```
import { Schema, model, Document } from 'mongoose';

export interface IUploadedFile {
  originalname: string;
  filename: string;
  path: string;
  size: number;
}

export interface IForm extends Document {
  firstName: string;
  lastName: string;
  email: string;
  phone?: string;
  address: {
    line1: string;
    line2?: string;
    city: string;
    state: string;
    country: string;
    zip: string;
  };
  files: IUploadedFile[];
  createdAt: Date;
}

const FileSchema = new Schema<IUploadedFile>({
  originalname: { type: String, required: true },
  filename: { type: String, required: true },
  path: { type: String, required: true },
  size: { type: Number, required: true },
});
```

```
const FormSchema = new Schema<IForm>({
  firstName: { type: String, required: true },
  lastName: { type: String, required: true },
  email: { type: String, required: true },
  phone: { type: String },
  address: {
    line1: { type: String, required: true },
    line2: { type: String },
    city: { type: String, required: true },
    state: { type: String, required: true },
    country: { type: String, required: true },
    zip: { type: String, required: true },
  },
  files: { type: [FileSchema], default: [] },
  createdAt: { type: Date, default: Date.now },
});

export const FormModel = model<IForm>('Form', FormSchema);
```

backend/src/repositories/form.repository.ts

```
import { FormModel, IForm } from '../models/form.model';

export interface IFormRepository {
  save(form: Partial<IForm>): Promise<IForm>;
  findById(id: string): Promise<IForm | null>;
}

export class FormRepository implements IFormRepository {
  async save(form: Partial<IForm>): Promise<IForm> {
    const created = new FormModel(form);
    return created.save();
  }

  async findById(id: string): Promise<IForm | null> {
    return FormModel.findById(id).exec();
  }
}
```

backend/src/services/form.service.ts

```
import { IFormRepository } from '../repositories/form.repository';
import { IForm, IUploadedFile } from '../models/form.model';

export interface FormDTO {
  firstName: string;
  lastName: string;
  email: string;
  phone?: string;
  address: {
    line1: string;
    line2?: string;
    city: string;
    state: string;
    country: string;
    zip: string;
  };
  files: IUploadedFile[];
}

export interface IFormService {
  create(form: FormDTO): Promise<IForm>;
  getById(id: string): Promise<IForm | null>;
}

export class FormService implements IFormService {
  private repository: IFormRepository;

  constructor(repository: IFormRepository) {
    this.repository = repository;
  }

  async create(form: FormDTO): Promise<IForm> {
    // SRP: Only creation/validation logic here.
    // Basic validation — keep small but explicit
    if (!form.firstName || !form.lastName || !form.email) {
      throw new Error('Missing required personal fields');
    }

    return this.repository.save(form as Partial<IForm>);
  }

  async getById(id: string): Promise<IForm | null> {
    return this.repository.findById(id);
  }
}
```

backend/src/controllers/form.controller.ts

```
import { Request, Response } from 'express';
import { IFormService } from '../services/form.service';

export class FormController {
  private service: IFormService;

  constructor(service: IFormService) {
    this.service = service;
  }

```

```
// single method to handle multipart form submission
async submit(req: Request, res: Response): Promise<void> {
  try {
    const body = req.body;
    const files = (req.files as Express.Multer.File[] | undefined) ?? [];

    // transform files into typed objects
    const uploaded = files.map((f) => ({
      originalname: f.originalname,
      filename: f.filename,
      path: f.path,
      size: f.size,
    }));

    const dto = {
      firstName: body.firstName,
      lastName: body.lastName,
      email: body.email,
      phone: body.phone,
      address: {
        line1: body.line1,
        line2: body.line2,
        city: body.city,
        state: body.state,
        country: body.country,
        zip: body.zip,
      },
      files: uploaded,
    };

    const created = await this.service.create(dto);
    res.status(201).json({ id: created._id });
  } catch (err) {
    console.error(err);
    res.status(400).json({ message: (err as Error).message });
  }
}

async get(req: Request, res: Response): Promise<void> {
  try {
    const id = req.params.id;
    const record = await this.service.getById(id);
    if (!record) {
      res.status(404).json({ message: 'Not found' });
      return;
    }
    res.json(record);
  } catch (err) {
    res.status(400).json({ message: (err as Error).message });
  }
  }
}
```

backend/src/utils/multer.ts

```
import multer from 'multer';
import path from 'path';
import fs from 'fs';

const UPLOAD_DIR = path.join(__dirname, '../../uploads');
if (!fs.existsSync(UPLOAD_DIR)) fs.mkdirSync(UPLOAD_DIR, { recursive:
true });

const storage = multer.diskStorage({
  destination: (_req, _file, cb) => cb(null, UPLOAD_DIR),
  filename: (_req, file, cb) => {
    const unique = `${Date.now()}-${file.originalname.replace(/\s+/g, '-')}`;
    cb(null, unique);
  },
});

export const upload = multer({ storage });
```

backend/src/routes/form.routes.ts

```
import { Router } from 'express';
import { upload } from '../utils/multer';
import { FormController } from '../controllers/form.controller';
import { FormService } from '../services/form.service';
import { FormRepository } from '../repositories/form.repository';

const repo = new FormRepository();
const service = new FormService(repo);
const controller = new FormController(service);

const router = Router();

// POST /api/forms (multipart form-data)
router.post('/', upload.array('files', 5), (req, res) =>
controller.submit(req, res));
router.get('/:id', (req, res) => controller.get(req, res));

export default router;
```

backend/src/app.ts

```
import express from 'express';
import cors from 'cors';
import formRoutes from './routes/form.routes';

const app = express();
app.use(cors());
// we still rely on multer for multipart; to parse JSON endpoints:
app.use(express.json());
app.use('/uploads', express.static('uploads'));
app.use('/api/forms', formRoutes);

export default app;
```

backend/src/server.ts

```
import app from './app';
import { connectDB } from './config/db';

const PORT: number = Number(process.env.PORT ?? 5000);

async function start(): Promise<void> {
  await connectDB();
  app.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
  });
}

start().catch((err) => {
  console.error('Failed to start', err);
  process.exit(1);
});
```

Frontend code

frontend/package.json

```
{
  "name": "ms-form-frontend",
  "version": "1.0.0",
  "private": true,
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-query": "^3.39.0",
    "axios": "^1.4.0",
    "react-router-dom": "^6.11.2"
  },
  "devDependencies": {
    "typescript": "^5.0.0",
    "vite": "^5.0.0",
    "@types/react": "^18.0.0",
    "@types/react-dom": "^18.0.0"
  },
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  }
}
```

frontend/src/types/form.ts

```
export interface Address {
  line1: string;
  line2?: string;
  city: string;
  state: string;
  country: string;
  zip: string;
}

export interface UploadedFile {
  originalname: string;
  filename: string;
  path: string;
  size: number;
}

export interface FormPayload {
  firstName: string;
  lastName: string;
  email: string;
  phone?: string;
  address: Address;
  files: UploadedFile[]; // stored as metadata after upload
}

export interface FormResponse {
  id: string;
}
```

frontend/src/api/apiClient.ts

```
import axios, { AxiosInstance } from 'axios';

export class ApiClient {
  private client: AxiosInstance;
  constructor(baseURL: string) {
    this.client = axios.create({ baseURL });
  }

  get instance(): AxiosInstance {
    return this.client;
  }
}

export const apiClient = new ApiClient(import.meta.env.VITE_API_URL ??
'http://localhost:5000');
```

frontend/src/services/formService.ts

```
import { ApiClient } from '../api/apiClient';
import { FormPayload, FormResponse } from '../types/form';

export interface IFormService {
  submit(form: FormData): Promise<FormResponse>;
}

export class FormService implements IFormService {
  private api: ApiClient;

  constructor(api: ApiClient) {
    this.api = api;
  }

  async submit(form: FormData): Promise<FormResponse> {
    const res = await this.api.instance.post('/api/forms', form, {
      headers: { 'Content-Type': 'multipart/form-data' },
    });
    return res.data as FormResponse;
  }
}

export const formService = new FormService(new
ApiClient(import.meta.env.VITE_API_URL ?? 'http://localhost:5000'));
```

frontend/src/hooks/useFormMutation.ts

```
import { useMutation, useQueryClient } from 'react-query';
import { formService } from '../services/formService';

export function useFormMutation() {
  const qc = useQueryClient();

  return useMutation(
    (form: FormData) => formService.submit(form),
    {
      onSuccess: (data) => {
        // keep cache small and simple; we may invalidate a "forms" list
later.
        console.log('submitted', data);
        qc.invalidateQueries(['forms']);
      },
    }
  );
}
```

frontend/src/components/formSteps/Step1Personal.tsx

```
import React from 'react';
import type { FormPayload } from '../../types/form';

interface Props {
  data: Partial<FormPayload>;
  update: (patch: Partial<FormPayload>) => void;
}

export const Step1Personal: React.FC<Props> = ({ data, update }) => {
  return (
    <div className="step">
      <label>
        First name
        <input
          type="text"
          value={data.firstName ?? ''}
          onChange={(e) => update({ firstName: e.target.value })}
          required
        />
      </label>
      <label>
        Last name
        <input
          type="text"
          value={data.lastName ?? ''}
          onChange={(e) => update({ lastName: e.target.value })}
          required
        />
      </label>
      <label>
        Email
        <input
          type="email"
          value={data.email ?? ''}
          onChange={(e) => update({ email: e.target.value })}
          required
        />
      </label>
      <label>
        Phone
        <input
          type="tel"
          value={data.phone ?? ''}
          onChange={(e) => update({ phone: e.target.value })}
        />
      </label>
    </div>
  );
};
```

frontend/src/components/formSteps/Step2Address.tsx

```
import React from 'react';
import type { FormPayload } from '../../types/form';

interface Props {
  data: Partial<FormPayload>;
  update: (patch: Partial<FormPayload>) => void;
}

export const Step2Address: React.FC<Props> = ({ data, update }) => {
  const addr = data.address ?? { line1: '', city: '', state: '', country:
'', zip: '' };

  return (
    <div className="step">
      <label>
        Address line 1
        <input
          type="text"
          value={addr.line1}
          onChange={(e) => update({ address: { ...(data.address ?? {}),
line1: e.target.value } })}
          required
        />
      </label>
      <label>
        Address line 2
        <input
          type="text"
          value={addr.line2 ?? ''}
          onChange={(e) => update({ address: { ...(data.address ?? {}),
line2: e.target.value } })}
        />
      </label>
      <label>
        City
        <input
          type="text"
          value={addr.city}
          onChange={(e) => update({ address: { ...(data.address ?? {}),
city: e.target.value } })}
          required
        />
      </label>
      <label>
        State
        <input
          type="text"
          value={addr.state}
          onChange={(e) => update({ address: { ...(data.address ?? {}),
state: e.target.value } })}
          required
        />
      </label>
      <label>
        Country
        <input
          type="text"
          value={addr.country}
          onChange={(e) => update({ address: { ...(data.address ?? {}),
country: e.target.value } })}
          required
        />
      </label>
      <label>
        ZIP
        <input
          type="text"
          value={addr.zip}
          onChange={(e) => update({ address: { ...(data.address ?? {}), zip:
e.target.value } })}
          required
        />
      </label>
    </div>
  );
};
```

frontend/src/components/formSteps/Step3Documents.tsx

```
import React, { useRef } from 'react';
import type { FormPayload } from '../../types/form';

interface Props {
  data: Partial<FormPayload>;
  update: (patch: Partial<FormPayload>) => void;
}

export const Step3Documents: React.FC<Props> = ({ data, update }) => {
  const inputRef = useRef<HTMLInputElement | null>(null);

  function onFilesChange(e: React.ChangeEvent<HTMLInputElement>): void {
    const files = e.target.files;
    if (!files) return;

    // Keep files in FormData during submission; front stores only metadata
later
    const array = Array.from(files);
    // just update with filenames as client preview
    const uploaded = array.map((f) => ({ originalname: f.name, filename:
f.name, path: '', size: f.size }));
    update({ files: uploaded });
  }

  return (
    <div className="step">
      <label>
        Upload documents (max 5)
        <input
          ref={inputRef}
          type="file"
          accept="image/*,application/pdf"
          multiple
          onChange={onFilesChange}
        />
      </label>
      <div className="preview">
        {(data.files ?? []).map((f, idx) => (
          <div key={idx} className="file-row">
            <span>{f.originalname}</span>
          </div>
        ))}
      </div>
    </div>
  );
};
```

frontend/src/components/formSteps/Step4Review.tsx

```
import React from 'react';
import type { FormPayload } from '../../types/form';

interface Props {
  data: Partial<FormPayload>;
}

```

```
export const Step4Review: React.FC<Props> = ({ data }) => {
  return (
    <div className="step">
      <h3>Review</h3>
      <pre style={{ whiteSpace: 'pre-wrap' }}>{JSON.stringify(data, null, 2)}
</pre>
    </div>
  );
};
```

frontend/src/components/MultiStepForm.tsx

```
import React, { useState } from 'react';
import { Step1Personal } from './formSteps/Step1Personal';
import { Step2Address } from './formSteps/Step2Address';
import { Step3Documents } from './formSteps/Step3Documents';
import { Step4Review } from './formSteps/Step4Review';
import type { FormPayload } from '../types/form';
import { useFormMutation } from '../hooks/useFormMutation';

type Step = 1 | 2 | 3 | 4;

export const MultiStepForm: React.FC = () => {
  const [step, setStep] = useState<Step>(1);
  const [data, setData] = useState<Partial<FormPayload>>({
    address: { line1: '', city: '', state: '', country: '', zip: '' },
    files: [],
  });

  const mutation = useFormMutation();

  const update = (patch: Partial<FormPayload>): void => {
    setData((prev) => ({ ...prev, ...patch }));
  };

  function next(): void {
    setStep((s) => (s < 4 ? ((s + 1) as Step) : s));
  }
  function prev(): void {
    setStep((s) => (s > 1 ? ((s - 1) as Step) : s));
  }

  function buildFormData(): FormData {
    const form = new FormData();
    form.append('firstName', data.firstName ?? '');
    form.append('lastName', data.lastName ?? '');
    form.append('email', data.email ?? '');
    form.append('phone', data.phone ?? '');

    const addr = data.address ?? { line1: '', line2: '', city: '', state:
```

```
'', country: '', zip: '' };
    form.append('line1', addr.line1);
    form.append('line2', addr.line2 ?? '');
    form.append('city', addr.city);
    form.append('state', addr.state);
    form.append('country', addr.country);
    form.append('zip', addr.zip);

    // Grab files from input element in Step3 — we will find the input by
querySelector
    const fileInput = document.querySelector('input[type=file]') as
HTMLInputElement | null;
    if (fileInput && fileInput.files) {
      const list = Array.from(fileInput.files).slice(0, 5);
      list.forEach((f) => form.append('files', f));
    }

    return form;
  }

  function onSubmit(e?: React.FormEvent): void {
    if (e) e.preventDefault();
    const fd = buildFormData();
    mutation.mutate(fd);
  }

  return (
    <div className="container">
      <form onSubmit={onSubmit} className="form">
        <div className="steps-indicator">Step {step} / 4</div>
        {step === 1 && <Step1Personal data={data} update={update} />}
        {step === 2 && <Step2Address data={data} update={update} />}
        {step === 3 && <Step3Documents data={data} update={update} />}
        {step === 4 && <Step4Review data={data} />}

        <div className="actions">
          <button type="button" onClick={prev} disabled={step === 1}>
            Back
          </button>
          {step < 4 ? (
            <button type="button" onClick={next}>
              Next
            </button>
          ) : (
            <button type="submit" disabled={mutation.isLoading}>
              {mutation.isLoading ? 'Submitting...' : 'Submit'}
            </button>
          )}
        </div>

        {mutation.isError && <div className="error">Error: {(mutation.error
as Error).message}</div>}
        {mutation.isSuccess && <div className="success">Submitted! ID:
{mutation.data?.id}</div>}
      </form>
      <style>{`
        .container{max-width:720px;margin:0 auto;padding:16px}
        .form{display:flex;flex-direction:column;gap:12px}
        label{display:flex;flex-direction:column;gap:6px}
        input[type=text],input[type=email],input[type=tel],input[type=file]
{padding:8px;border:1px solid #ddd;border-radius:6px}
        .actions{display:flex;gap:8px}
        @media (max-width:600px){.container{padding:8px}}
      `}</style>
    </div>
  );
};
```

frontend/src/App.tsx

```
import React from 'react';
import { QueryClient, QueryClientProvider } from 'react-query';
import { MultiStepForm } from './components/MultiStepForm';

const qc = new QueryClient();

export const App: React.FC = () => {
  return (
    <QueryClientProvider client={qc}>
      <div style={{ padding: 16 }}>
        <h2>Multi-step Form (TypeScript + MERN)</h2>
        <MultiStepForm />
      </div>
    </QueryClientProvider>
  );
};

export default App;
```

frontend/src/main.tsx

```
import React from 'react';
import { createRoot } from 'react-dom/client';
import App from './App';
import './index.css';

createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

frontend/index.css

```
body{font-family:Inter,system-ui,-apple-system,BlinkMacSystemFont,'Segoe
UI',Roboto,'Helvetica Neue',Arial}
.step{display:flex;flex-direction:column;gap:8px}
.file-row{padding:6px;border:1px solid #eee;border-radius:6px;margin-top:6px}
.preview{margin-top:8px}
```

How the solution follows SOLID / OOP / Best Practices

- **SRP**: Controllers handle HTTP, Services handle business logic, Repositories handle persistence.
- **OCP**: Service and repository depend on interfaces; to add new persistence (e.g., S3 or another DB) implement `IFormRepository` without changing service code.
- **LSP**: Repository implementations follow the same interface and can be substituted.
- **Low coupling**: Modules depend on interfaces and `ApiClient` abstraction.
- **High cohesion**: Each file has a focused responsibility.
- **React Query**: handles caching, mutations, and network state for fast UX.
- **No** `any`: All types are explicit. Use `Partial<T>` and carefully typed DTOs.

Running locally

1. Create `.env` files

backend/.env

```
MONGO_URI=mongodb://localhost:27017/msform
PORT=5000
```

frontend/.env

```
VITE_API_URL=http://localhost:5000
```

1. Start backend

```
cd backend
npm install
npm run dev
```

1. Start frontend

```
cd frontend
npm install
npm run dev
```

Open the frontend (vite) local URL.

Notes & Extensibility

- To store files on cloud (S3), create a new repository implementing `IFormRepository` that accepts `IUploadedFile` with S3 URLs — due to OCP you won't change service/controller.
- Add form step components and wire them in `MultiStepForm` to extend the UI.

If you'd like, I can also: - generate full `tsconfig.json` and `vite` configs, - add unit tests for service/repository, - replace local disk storage with S3, - or scaffold a Docker Compose to run Mongo + backend + frontend.

Tell me which of the above you'd like next and I will add it as well.