

Next.js 15 + NextAuth v5 Authentication Project (Notes in Bangla)

প্রজেক্টের সারসংক্ষেপ

এই প্রজেক্টে আমরা Next.js 15, NextAuth v5, Prisma, PostgreSQL ব্যবহার করে একটি Authentication System তৈরি করেছি।


এর মাধ্যমে ইউজাররা করতে পারবে:

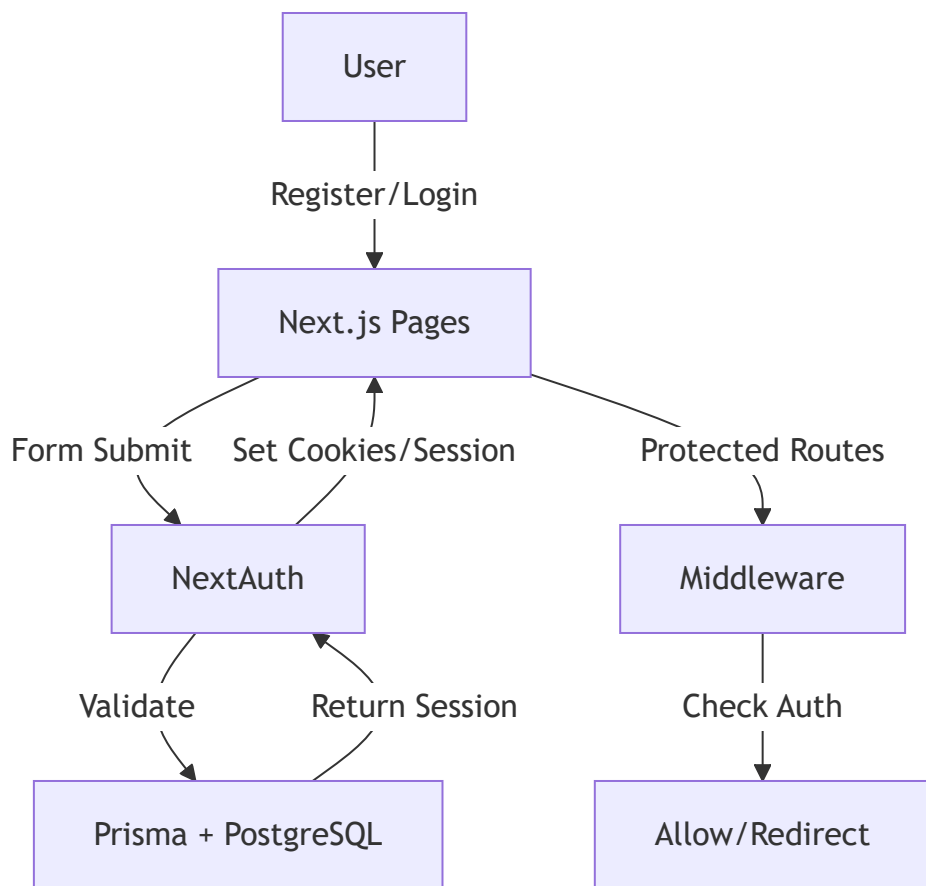
- নতুন একাউন্ট রেজিস্ট্রেশন
- লগইন (Credentials / OAuth Provider)
- সেশন ম্যানেজমেন্ট
- প্রটেক্টেড রাউট ব্যবহার
- Middleware দিয়ে রাউট প্রটেকশন
- লগআউট

ফোল্ডার স্ট্রাকচার

```
src/
├─ app/
│   ├─ (auth)/
│   │   ├─ login/
│   │   │   ├─ page.tsx
│   │   │   └─ loginForm.tsx
│   │   └─ register/
│   │       ├─ page.tsx
│   │       └─ registerForm.tsx
│   └─ settings/
│       └─ page.tsx
└─ layout.tsx

├─ lib/
│   ├─ schemas/
│   │   ├─ loginSchema.ts
│   │   └─ registerSchema.ts
│   ├─ prisma.ts
│   └─ utils.ts
├─ components/
│   ├─ ui/
│   │   └─ button.tsx
│   └─ form.tsx
├─ auth.config.ts
├─ auth.ts
├─ middleware.ts
└─ routes.ts
```

 প্রজেক্ট আর্কিটেকচার (High Level Flow)



📌 প্রধান প্রধান ফাইলের কাজ

- auth.ts → NextAuth এর server-side config এবং helper ফাংশন (signIn, signOut, auth)
- auth.config.ts → Authentication Provider, Adapter, Callbacks ইত্যাদি কনফিগারেশন
- middleware.ts → Public / Private / Auth routes handle করার জন্য Middleware
- routes.ts → Public routes, Auth routes এবং Redirect path এর কনফিগারেশন
- prisma.ts → Prisma client setup
- schemas/ → Form Validation schemas (Zod দিয়ে)
- (auth)/login & register → Login / Register UI + Form submit

Core Authentication ফাইলগুলো:

- auth.config.ts
- auth.ts
- routes.ts
- middleware.ts

🔑 Authentication Core Files (Bangla Note)

1 auth.config.ts

👉 এখানে আমরা NextAuth এর configuration define করি।

```

import { PrismaAdapter } from "@auth/prisma-adapter";
import { db } from "@lib/prisma";
import type { NextAuthConfig } from "next-auth";
import Credentials from "next-auth/providers/credentials";

export default {
  adapter: PrismaAdapter(db),
  providers: [
    Credentials({
      name: "Credentials",
      credentials: {
        email: {},
        password: {}
      },
      authorize: async (credentials) => {
        // এখানে ইউজারের ইমেল + পাসওয়ার্ড চেক করা হবে
      }
    })
  ],
  pages: {
    signIn: "/login",
    error: "/error"
  },
  session: { strategy: "jwt" },
  callbacks: {
    async session({ token, session }) {
      if (token) {
        session.user.id = token.id;
        session.user.email = token.email;
      }
      return session;
    },
    async jwt({ token, user }) {
      if (user) {
        token.id = user.id;
        token.email = user.email;
      }
      return token;
    }
  }
} satisfies NextAuthConfig;

```

📌 কাজ:

- Prisma Adapter ব্যবহার করে DB এর সাথে কানেক্ট করে
- Credentials provider (email/password) ব্যবহার করে লগইন করা যায়
- Session + JWT callback দিয়ে সেশন কাস্টমাইজ করা যায়

2 auth.ts

👉 এই ফাইলে আমরা NextAuth ফাংশন initialize করি।

```
import NextAuth from "next-auth";
import authConfig from "./auth.config";

export const { auth, signIn, signOut } = NextAuth(authConfig);
```

📌 কাজ:

- auth → server side এ ইউজারের session check করার জন্য
- signIn → লগইন করার জন্য
- signOut → লগআউট করার জন্য

3 routes.ts

👉 এখানে আমরা public, private, auth routes define করি।

```
export const publicRoutes = ["/"];
export const authRoutes = ["/login", "/register"];
export const apiAuthPrefix = "/api/auth";
export const DEFAULT_LOGIN_REDIRECT = "/settings";
```

📌 কাজ:

- publicRoutes → যেসব রাউট সবার জন্য ওপেন
- authRoutes → শুধু লগইন/রেজিস্ট্রেশন পেজ
- apiAuthPrefix → API authentication routes
- DEFAULT_LOGIN_REDIRECT → লগইনের পর ইউজার কোথায় যাবে

4 middleware.ts

👉 এখানে middleware দিয়ে route protect করা হয়।

```

import authConfig from "./auth.config";
import NextAuth from "next-auth";
import { apiAuthPrefix, authRoutes, DEFAULT_LOGIN_REDIRECT, publicRoutes } from "./routes";

const { auth } = NextAuth(authConfig);

export default auth((req) => {
  const { nextUrl } = req;
  const isLoggedIn = !!req.auth;

  const isApiAuthRoute = nextUrl.pathname.startsWith(apiAuthPrefix);
  const isPublicRoute = publicRoutes.includes(nextUrl.pathname);
  const isAuthRoute = authRoutes.includes(nextUrl.pathname);

  if (isApiAuthRoute) return null;

  if (isAuthRoute) {
    if (isLoggedIn) {
      return Response.redirect(new URL(DEFAULT_LOGIN_REDIRECT, nextUrl));
    }
    return null;
  }

  if (!isLoggedIn && !isPublicRoute) {
    return Response.redirect(new URL("/login", nextUrl));
  }

  return null;
});

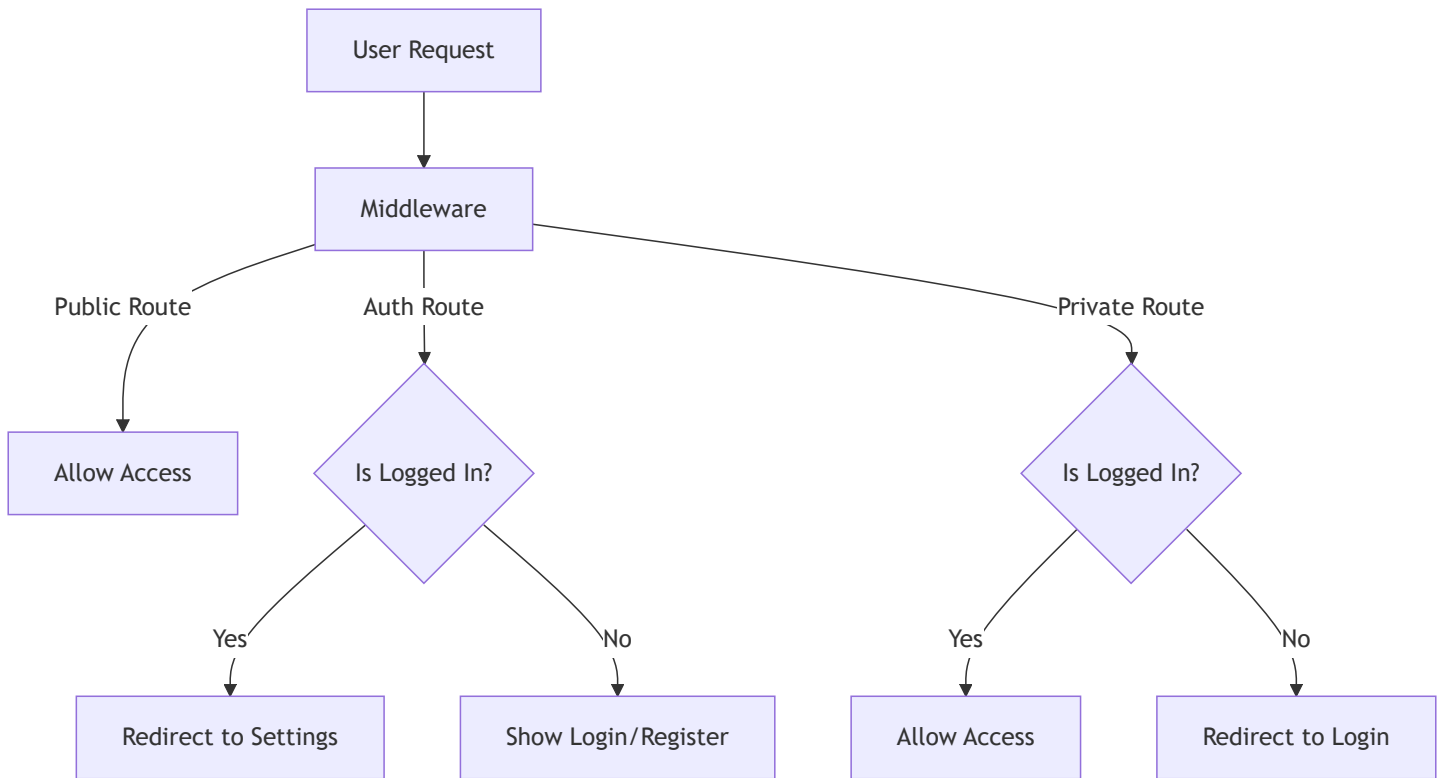
export const config = {
  matcher: ["/((?!api|_next/static|_next/image|favicon.ico|sitemap.xml|robots.txt).*)"],
};

```

কাজ:

- API route গুলোতে middleware কাজ করবে না
- যদি ইউজার লগইন করা থাকে → আবার login/register এ ঢুকতে পারবে না (redirect হবে settings এ)
- যদি লগইন ছাড়া private route এ যায় → redirect হবে login এ

Flow Diagram



📖 Prisma + PostgreSQL Setup (Next.js 15 + NextAuth@5)

1 Prisma Install

প্রথমে প্রজেক্টে Prisma এবং PostgreSQL ড্রাইভার ইন্সটল করতে হবে:

```
pnpm install prisma @prisma/client
pnpm install --save-dev ts-node
pnpm install pg
```

2 Prisma Init

- টার্মিনালে রান করো:

```
npx prisma init
```

👉 এতে তোমার প্রজেক্টে prisma/schema.prisma এবং .env ফাইল তৈরি হবে।

3 .env ফাইল

.env এ PostgreSQL connection URL লিখতে হবে:

```
DATABASE_URL="postgresql://username:password@localhost:5432/mydb?schema=public"
```

4 schema.prisma

একটি সাধারণ User মডেল NextAuth compatible ভাবে লিখি:


```

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id          String   @id @default(cuid())
  name        String?
  email       String   @unique
  emailVerified DateTime?
  passwordHash String?
  image       String?
  profileComplete Boolean @default(false)
  role        Role      @default(USER)
  accounts    Account[]
  sessions    Session[]
}

model Account {
  id          String   @id @default(cuid())
  userId      String
  type        String
  provider     String
  providerAccountId String
  refresh_token String?
  access_token String?
  expires_at   Int?
  token_type   String?
  scope        String?
  id_token     String?
  session_state String?
  user        User     @relation(fields: [userId], references: [id])

  @@unique([provider, providerAccountId])
}

model Session {
  id          String   @id @default(cuid())
  sessionToken String   @unique

```

```

    userId      String
    expires      DateTime
    user         User    @relation(fields: [userId], references: [id])
  }

model VerificationToken {
  identifier String
  token       String    @unique
  expires     DateTime

  @@unique([identifier, token])
}

enum Role {
  USER
  ADMIN
}

```

5 Prisma Migrate

- ডাটাবেজে টেবিল তৈরি করতে:

```
npx prisma migrate dev --name init
```

6 Prisma Client

lib/prisma.ts ফাইল তৈরি করো:

```

import { PrismaClient } from "@prisma/client";

declare global {
  var prisma: PrismaClient | undefined;
}

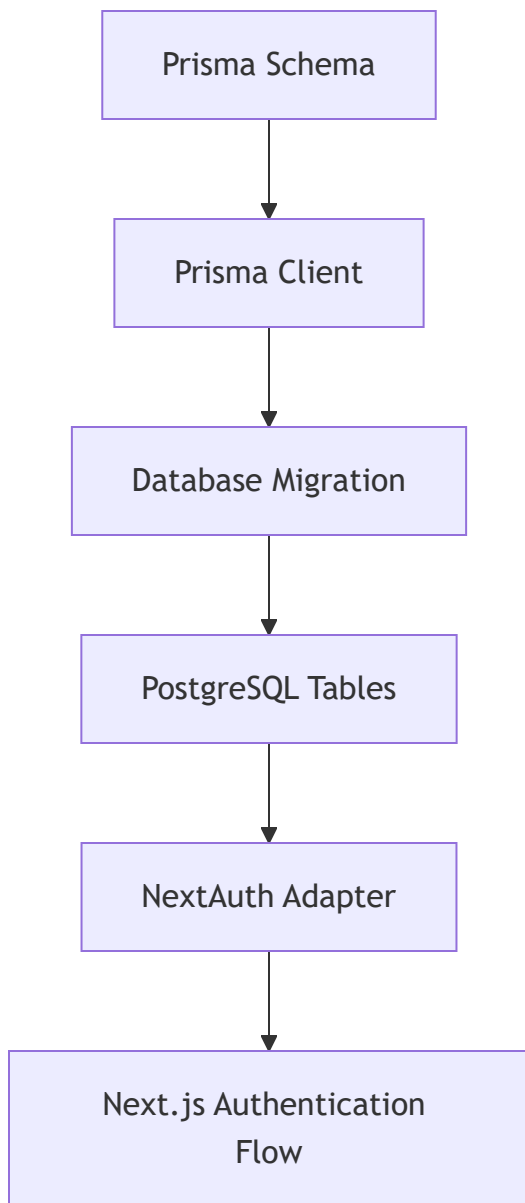
export const db = globalThis.prisma || new PrismaClient();

if (process.env.NODE_ENV !== "production") globalThis.prisma = db;

```

👉 এখন প্রজেক্টে db ব্যবহার করে যেকোনো জায়গায় Prisma query চালানো যাবে।

7 Flow Chart



📌 মূল পয়েন্ট

- prisma/schema.prisma → Database schema define
- npx prisma migrate dev → DB তে টেবিল তৈরি
- db (PrismaClient) → Query চালানোর জন্য ব্যবহার
- NextAuth এর সাথে PrismaAdapter দিয়ে connect করা হয়

🔒 Login & Register Flow (Next.js 15 + NextAuth@5)

1 Schema Validation (Zod দিয়ে)

আমরা Zod ব্যবহার করছি ফর্ম ইনপুট validate করার জন্য।

loginSchema.ts

```
import * as z from "zod";

export const loginSchema = z.object({
  email: z.string().email({ message: "সঠিক ইমেইল দিন" }),
  password: z.string().min(6, { message: "পাসওয়ার্ড কমপক্ষে ৬ অক্ষরের হতে হবে" }),
});

export type LoginSchema = z.infer<typeof loginSchema>;

registerSchema.ts
import * as z from "zod";

export const registerSchema = z.object({
  name: z.string().min(2, { message: "নাম কমপক্ষে ২ অক্ষরের হতে হবে" }),
  email: z.string().email({ message: "সঠিক ইমেইল দিন" }),
  password: z.string().min(6, { message: "পাসওয়ার্ড কমপক্ষে ৬ অক্ষরের হতে হবে" }),
});

export type RegisterSchema = z.infer<typeof registerSchema>;
```

2 Login Form (Formik + Shadcn UI)

(auth)/login/loginForm.tsx

```

"use client";

import { useFormik } from "formik";
import { loginSchema } from "@lib/schemas/loginSchema";
import { login } from "@actions/login";
import { useState } from "react";

export default function LoginForm() {
  const [status, setStatus] = useState<{ success?: string; error?: string }>({});

  const formik = useFormik({
    initialValues: { email: "", password: "" },
    validate: (values) => {
      const result = loginSchema.safeParse(values);
      if (!result.success) {
        return result.error.flatten().fieldErrors;
      }
      return {};
    },
    onSubmit: async (values, { setSubmitting }) => {
      const data = await login(values);
      if (data.error) {
        setStatus({ error: data.error });
      } else if (data.success) {
        setStatus({ success: data.success });
      }
      setSubmitting(false);
    },
  });

  return (
    <form onSubmit={formik.handleSubmit} className="space-y-4">
      <input
        type="email"
        name="email"
        placeholder="ইমেইল"
        onChange={formik.handleChange}
        className="border p-2 w-full rounded"
      />
      <input
        type="password"
        name="password"
        placeholder="পাসওয়ার্ড"

```

```

        onChange={formik.handleChange}
        className="border p-2 w-full rounded"
      />
      <button
        type="submit"
        className="bg-purple-600 text-white px-4 py-2 rounded hover:bg-purple-700"
      >
        লগইন
      </button>

      {/* ✅ Success/Error Message */}
      {status.error && <p className="text-red-600">{status.error}</p>}
      {status.success && <p className="text-green-600">{status.success}</p>}
    </form>
  );
}

```

3 Register Form (Formik + Shadcn UI)

(auth)/register/registerForm.tsx

```

"use client";

import { useFormik } from "formik";
import { registerSchema } from "@lib/schemas/registerSchema";
import { register } from "@actions/register";
import { useState } from "react";

export default function RegisterForm() {
  const [status, setStatus] = useState<{ success?: string; error?: string }>({});

  const formik = useFormik({
    initialValues: { name: "", email: "", password: "" },
    validate: (values) => {
      const result = registerSchema.safeParse(values);
      if (!result.success) {
        return result.error.flatten().fieldErrors;
      }
      return {};
    },
    onSubmit: async (values, { setSubmitting }) => {
      const data = await register(values);
      if (data.error) {
        setStatus({ error: data.error });
      } else if (data.success) {
        setStatus({ success: data.success });
      }
      setSubmitting(false);
    },
  });

  return (
    <form onSubmit={formik.handleSubmit} className="space-y-4">
      <input
        type="text"
        name="name"
        placeholder="নাম"
        onChange={formik.handleChange}
        className="border p-2 w-full rounded"
      />
      <input
        type="email"
        name="email"
        placeholder="ইমেইল"

```

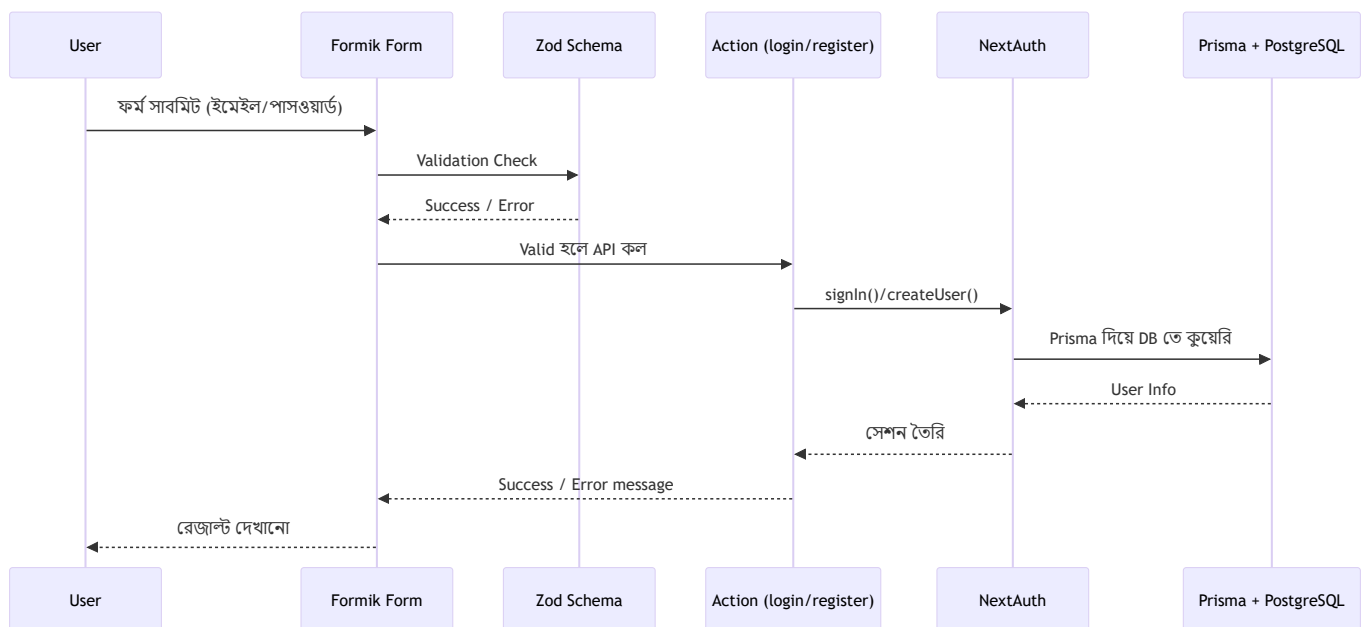
```

    onChange={formik.handleChange}
    className="border p-2 w-full rounded"
  />
  <input
    type="password"
    name="password"
    placeholder="পাসওয়ার্ড"
    onChange={formik.handleChange}
    className="border p-2 w-full rounded"
  />
  <button
    type="submit"
    className="bg-green-600 text-white px-4 py-2 rounded hover:bg-green-700"
  >
    রেজিস্টার
  </button>

  { /* ✅ Success/Error Message */ }
  {status.error && <p className="text-red-600">{status.error}</p>}
  {status.success && <p className="text-green-600">{status.success}</p>}
</form>
);
}

```

4 Login/Register Flow Diagram



- Formik ফর্ম হ্যান্ডলিং এর জন্য
- Zod ফর্ম validation এর জন্য
- Action ফাইল (login.ts / register.ts) → সার্ভার সাইড auth handle করার জন্য
- Success/Error message UI তে show করা

Middleware & Route Protection (Next.js 15 + NextAuth@5)

1 Route Configuration (routes.ts)

প্রথমে আমরা কোন রুটগুলো পাবলিক এবং কোনগুলো প্রোটেক্টেড তা সংজ্ঞায়িত করি।

```
// src/routes.ts

// পাবলিক রুট
export const publicRoutes = ["/"];

// লগইন/রেজিস্টার রুট (Auth route)
export const authRoutes = ["/login", "/register"];

// API authentication prefix
export const apiAuthPrefix = "/api/auth";

// লগইন সফল হলে redirect করার ডিফল্ট পথ
export const DEFAULT_LOGIN_REDIRECT = "/settings";
```

ব্যাখ্যা:

- publicRoutes → যেকোনো ব্যবহারকারী এক্সেস করতে পারবে, লগইন না করলেও।
- authRoutes → লগইন/রেজিস্টার পেজ। লগইন থাকা অবস্থায় এখানে গেলে redirect করা হবে।
- apiAuthPrefix → NextAuth এর API route এর জন্য prefix।
- DEFAULT_LOGIN_REDIRECT → লগইন বা register সফল হলে redirect করার পথ।

2 Middleware (middleware.ts)

Middleware দিয়ে আমরা route access control করি।

```

// src/middleware.ts

import authConfig from "./auth.config";
import NextAuth from "next-auth";
import { apiAuthPrefix, authRoutes, DEFAULT_LOGIN_REDIRECT, publicRoutes } from "./routes";

const { auth } = NextAuth(authConfig);

export default auth((req) => {
  const { nextUrl } = req;
  const isLoggedIn = !!req.auth;

  const isApiAuthRoute = nextUrl.pathname.startsWith(apiAuthPrefix);
  const isPublicRoute = publicRoutes.includes(nextUrl.pathname);
  const isAuthRoute = authRoutes.includes(nextUrl.pathname);

  if (isApiAuthRoute) {
    return null; // API auth route এ middleware skip
  }

  // Auth pages এ লগইন থাকলে redirect
  if (isAuthRoute) {
    if (isLoggedIn) {
      return Response.redirect(new URL(DEFAULT_LOGIN_REDIRECT, nextUrl));
    }
    return null;
  }

  // Non-public route এ লগইন না থাকলে redirect to /login
  if (!isLoggedIn && !isPublicRoute) {
    return Response.redirect(new URL("/login", nextUrl));
  }

  return null; // অন্য সব ক্ষেত্রে allow
});

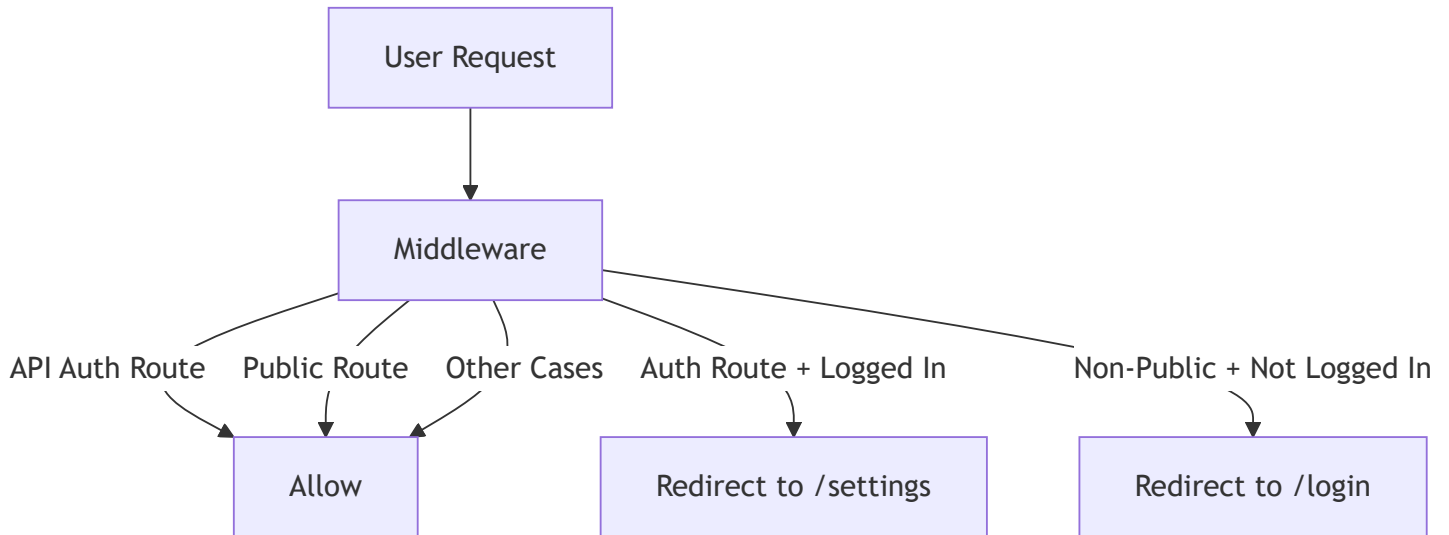
// Middleware কনফিগারেশন
export const config = {
  matcher: ['/(?!api|_next/static|_next/image|favicon.ico|sitemap.xml|robots.txt).*'],
};

```


ব্যাখ্যা:

- NextAuth(authConfig) → middleware instance তৈরি।
- req.auth → ব্যবহারকারী logged in কি না চেক করে।
- Response.redirect(new URL(...)) → ব্যবহারকারীকে redirect করে।
- matcher → কোন রুটে middleware apply হবে। _next/static, _next/image ইত্যাদি exclude করা হয়েছে।

3 Middleware Flow Diagram



4 Key Points

- Middleware ensures route protection and automatic redirection।
- Auth routes এ logged-in ব্যবহারকারীকে prevent করা হয় /login বা /register এ যাওয়া থেকে।
- Non-public pages এ logged-out ব্যবহারকারীকে /login এ redirect করা হয়।
- API auth routes এ middleware skip করা হয়, যাতে NextAuth API ঠিকভাবে কাজ করে।
-  Middleware + routes এর মাধ্যমে তুমি পুরো অ্যাপের প্রোটেকশন সহজে control করতে পারছ।

NextAuth Setup & SignIn/SignOut Flow (Next.js 15 + NextAuth@5)

1 Auth Config (auth.config.ts)

Credentials provider দিয়ে custom login logic তৈরি করা হয়েছে।

```
// src/auth.config.ts
import bcrypt from "bcryptjs";
import type { NextAuthConfig } from "next-auth";
import Credentials from "next-auth/providers/credentials";
import { loginSchema } from "@/lib/schemas/loginSchema";
import { getUserByEmail } from "@/app/actions/user/user";

export default {
  providers: [
    Credentials({
      async authorize(credentials) {
        const validatedFields = loginSchema.safeParse(credentials);

        if (validatedFields.success) {
          const { email, password } = validatedFields.data;
          const user = await getUserByEmail(email);

          if (!user || !user.password) return null;

          const passwordsMatch = await bcrypt.compare(password, user.password);
          if (passwordsMatch) return user;
        }

        return null;
      }
    })
  ]
} satisfies NextAuthConfig;
```

ব্যাখ্যা:

- Credentials → custom email/password login।
- authorize → ব্যবহারকারীর credentials validate করে।
- bcrypt.compare → hashed password verify করে।
- যদি credentials সঠিক হয়, তাহলে ব্যবহারকারীর object return হয়, অন্যথায় null।

2 Auth Handler (auth.ts)

NextAuth instance তৈরি করে export করা হয়েছে।

```
// src/auth.ts
import NextAuth from "next-auth";
import { prisma } from "../lib/prisma";
import authConfig from "../auth.config";
import { PrismaAdapter } from "@auth/prisma-adapter";

export const {
  handlers: { GET, POST },
  auth,
  signIn,
  signOut
} = NextAuth({
  adapter: PrismaAdapter(prisma),
  session: { strategy: "jwt" },
  ...authConfig
});
```

ব্যাখ্যা:

- PrismaAdapter(prisma) → Prisma দিয়ে DB সংযোগ।
- session: { strategy: "jwt" } → JWT based session।
- handlers → NextAuth API route এর GET & POST handlers।
- signIn, signOut → ফ্রন্টএন্ড থেকে লগইন/লগআউট handle করার জন্য export করা হয়েছে।

3 SignIn Action (login.ts)

```

"use server";
import { loginSchema, LoginSchema } from "@lib/schemas/loginSchema";
import { signIn } from "@auth";
import { DEFAULT_LOGIN_REDIRECT } from "@routes";

export const login = async (values: LoginSchema): Promise<{ error?: string; success?: string }>
  const validatedFields = loginSchema.safeParse(values);

  if (!validatedFields.success) {
    return { error: "Invalid fields" };
  }

  const { email, password } = validatedFields.data;

  try {
    await signIn("credentials", {
      email,
      password,
      redirect: false, // redirect নিজে handle করতে হবে
      callbackUrl: DEFAULT_LOGIN_REDIRECT
    });

    return { success: "Login successful!" };
  } catch (error) {
    return { error: "Something went wrong" };
  }
};

```

ব্যাখ্যা:

- Formik / Zod validated input নিয়ে server-side signIn handle করে।
- redirect: false → NextAuth auto redirect বন্ধ করে। আমরা নিজেই handle করি।
- Success/Error message return করে UI এ দেখানো হয়।

4 SignOut Button

```

"use client";
import { signOut } from "next-auth/react";

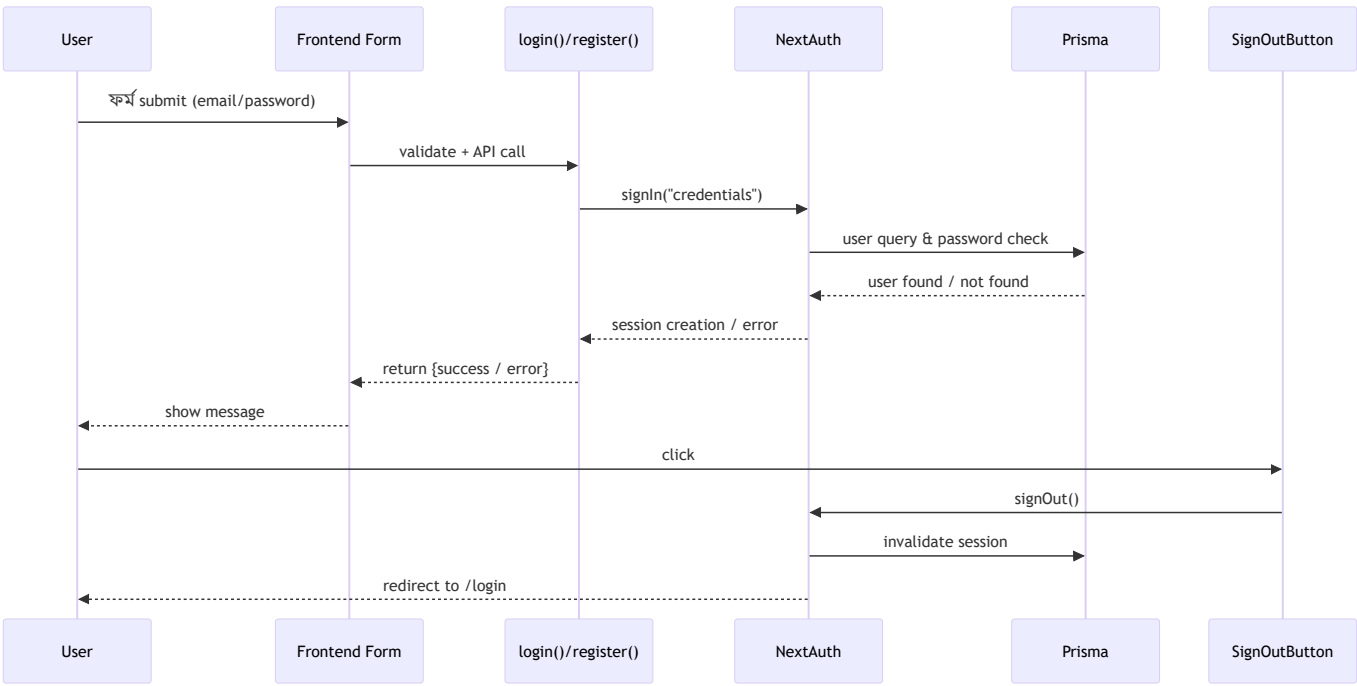
export default function SignOutButton() {
  return (
    <button
      onClick={() => signOut({ redirect: true, callbackUrl: "/login" })}
      className="bg-red-600 text-white px-4 py-2 rounded hover:bg-red-700"
    >
      Sign Out
    </button>
  );
}

```

ব্যাখ্যা:

- ব্যবহারকারী button click করলে signOut() call হয়।
- redirect: true → লগআউটের পরে redirect।
- callbackUrl → লগআউট হলে redirect page (এখানে /login)।

5 NextAuth Flow Diagram



ব্যাখ্যা:

- লগইন ফ্লো → Formik → Zod → login action → NextAuth → Prisma → Success/Error → UI।
- লগআউট ফ্লো → signOut → session invalid → redirect /login।

✅ এই Markdown নোট + Flow Diagram দিয়ে পুরো Next.js 15 + NextAuth@5 authentication logic সম্পূর্ণভাবে বোঝা যাবে।