# T3 Stack

応用編: Next Auth & SSRの実装紹介

Press Space for next page →

# 自己紹介

- 📝 飯野陽平（wheatandcat）
- 🏢 法人設立（合同会社UNICORN 代表社員）
- 💻 Work: シェアフル株式会社CTO
- 📚 Blog: https://www.wheatandcat.me/
- 🛠️ 今までに作ったもの
  - memoir
  - ペペロミア
  - MarkyLinky

# T3 Stackとは？

- 最近、話題の Web アプリ開発のアーキテクチャ
- 以下の 3 つの思想に集点を当てて設計された技術スタック
  - simplicity(簡潔さ)
  - modularity(モジュール性)
  - full-stack typesafety(フルスタックの型安全)
- t3-appとしてコマンドラインツールも公開されている
- 全体の紹介は以前スライドを作成したので、そちらを参照
  - T3 Stack + Supabaseでアプリを作ってみる - Speaker Deck
- このスライドでは、T3 Stackの応用編として、**Next Auth**と**SSR対応**を紹介

# Next Authとは？

- Next.jsアプリケーション専用に設計されたオープンソースの認証ライブラリ
- SSRや、React Server Components等のサーバーサイドでも認証が可能
- 様々な認証プロバイダーをサポート（Google、Facebookなど）
- 簡単な設定で認証が実装可能

# Next Authで認証を実装

各プロバイダーの設定は以下のように行う

```
export const authOptions: NextAuthOptions = {
  providers: [
    DiscordProvider({
      clientId: env.DISCORD_CLIENT_ID,
      clientSecret: env.DISCORD_CLIENT_SECRET,
    }),
    AppleProvider({
      clientId: env.APPLE_ID,
      clientSecret: env.APPLE_SECRET,
    }),
    GoogleProvider({
      clientId: env.GOOGLE_CLIENT_ID,
      clientSecret: env.GOOGLE_CLIENT_SECRET,
    }),
  ],
```

- Google | NextAuth.js

# Next Authで認証を実装

各プロバイダーの設定は以下のように行う

```
export const authOptions: NextAuthOptions = {
  providers: [
    DiscordProvider({
      clientId: env.DISCORD_CLIENT_ID,
      clientSecret: env.DISCORD_CLIENT_SECRET,
    }),
    AppleProvider({
      clientId: env.APPLE_ID,
      clientSecret: env.APPLE_SECRET,
    }),
    GoogleProvider({
      clientId: env.GOOGLE_CLIENT_ID,
      clientSecret: env.GOOGLE_CLIENT_SECRET,
    }),
  ],
```

- Google | NextAuth.js

# Next Authで認証を実装

各プロバイダーの設定は以下のように行う

```
export const authOptions: NextAuthOptions = {
  providers: [
    DiscordProvider({
      clientId: env.DISCORD_CLIENT_ID,
      clientSecret: env.DISCORD_CLIENT_SECRET,
    }),
    AppleProvider({
      clientId: env.APPLE_ID,
      clientSecret: env.APPLE_SECRET,
    }),
    GoogleProvider({
      clientId: env.GOOGLE_CLIENT_ID,
      clientSecret: env.GOOGLE_CLIENT_SECRET,
    }),
  ],
```

- Google | NextAuth.js

# Next Authで認証を実装

各プロバイダーの設定は以下のように行う

```
export const authOptions: NextAuthOptions = {
  providers: [
    DiscordProvider({
      clientId: env.DISCORD_CLIENT_ID,
      clientSecret: env.DISCORD_CLIENT_SECRET,
    }),
    AppleProvider({
      clientId: env.APPLE_ID,
      clientSecret: env.APPLE_SECRET,
    }),
    GoogleProvider({
      clientId: env.GOOGLE_CLIENT_ID,
      clientSecret: env.GOOGLE_CLIENT_SECRET,
    }),
  ],
```

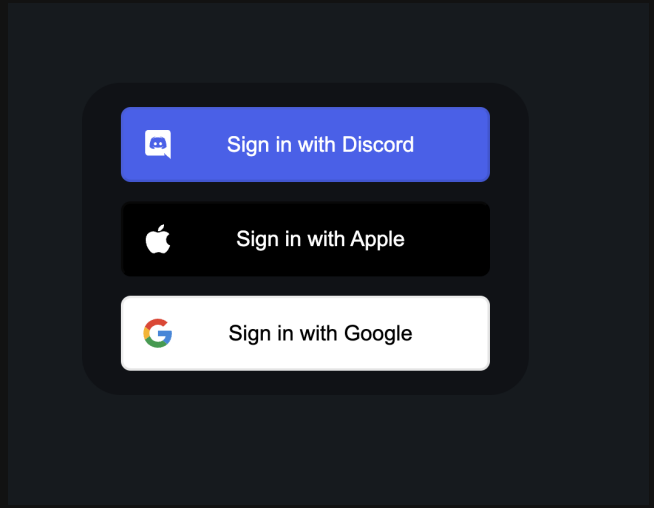
- Google | NextAuth.js

# ログイン画面の実装

プロバイダーの設定が完了したら、**signIn**のメソッドをコールするだけで以下の画面

```
import { signIn, signOut, useSession } from "next-auth/react";

<button onClick={() ⇒ signIn("credentials", {callbackUrl: "/",})}>
  ログイン
</button>
```



# ログイン画面のカスタマイズ

ログイン画面をカスタマイズしたい場合は以下を追加することで可能

```
export const authOptions: NextAuthOptions = {
  pages: {
    signIn: "/auth/signin",
    error: "/auth/signin",
  },
```

# 認証実装①（クライアントサイドで取得）

認証したユーザーの値は以下で取得可能

```
import { signIn } from "next-auth/react";

function Home() {
  const { data: session } = useSession();

  if (!session) {
    return (<div>ログイン前です</div>)
  }

  return (
    <div>
      <div>ログイン後です</div>
      <div>{session.user.id}</div>
      <div>{session.user.name}</div>
      <div>{session.user.email}</div>
    </div>
  )
}
```

# 認証実装①（クライアントサイドで取得）

認証したユーザーの値は以下で取得可能

```
import { signIn } from "next-auth/react";

function Home() {
  const { data: session } = useSession();

  if (!session) {
    return (<div>ログイン前です</div>)
  }

  return (
    <div>
      <div>ログイン後です</div>
      <div>{session.user.id}</div>
      <div>{session.user.name}</div>
      <div>{session.user.email}</div>
    </div>
  )
}
```

# 認証実装①（クライアントサイドで取得）

認証したユーザーの値は以下で取得可能

```
import { signIn } from "next-auth/react";

function Home() {
  const { data: session } = useSession();

  if (!session) {
    return (<div>ログイン前です</div>)
  }

  return (
    <div>
      <div>ログイン後です</div>
      <div>{session.user.id}</div>
      <div>{session.user.name}</div>
      <div>{session.user.email}</div>
    </div>
  )
}
```

# 認証実装①（クライアントサイドで取得）

認証したユーザーの値は以下で取得可能

```
import { signIn } from "next-auth/react";

function Home() {
  const { data: session } = useSession();

  if (!session) {
    return (<div>ログイン前です</div>)
  }

  return (
    <div>
      <div>ログイン後です</div>
      <div>{session.user.id}</div>
      <div>{session.user.name}</div>
      <div>{session.user.email}</div>
    </div>
  )
}
```

# 認証実装①（クライアントサイドで取得）

認証したユーザーの値は以下で取得可能

```
import { signIn } from "next-auth/react";

function Home() {
  const { data: session } = useSession();

  if (!session) {
    return (<div>ログイン前です</div>)
  }

  return (
    <div>
      <div>ログイン後です</div>
      <div>{session.user.id}</div>
      <div>{session.user.name}</div>
      <div>{session.user.email}</div>
    </div>
  )
}
```

# 認証実装②（SSRで認証）

SSRで認証したい場合は以下のように実装

```
import { signIn } from "next-auth/react";
import { type GetServerSideProps } from "next";
import { getServerAuthSession } from "~/server/auth";

export const getServerSideProps: GetServerSideProps = async (ctx) ⇒ {
  const session = await getServerAuthSession(ctx);
  return {
    props: { session },
  };
};

function Home() {
  const { data: session } = useSession();
```

# 認証実装②（SSRで認証）

SSRで認証したい場合は以下のように実装

```
import { signIn } from "next-auth/react";
import { type GetServerSideProps } from "next";
import { getServerAuthSession } from "~/server/auth";

export const getServerSideProps: GetServerSideProps = async (ctx) ⇒ {
  const session = await getServerAuthSession(ctx);
  return {
    props: { session },
  };
};

function Home() {
  const { data: session } = useSession();
```

# 認証実装②（SSRで認証）

SSRで認証したい場合は以下のように実装

```
import { signIn } from "next-auth/react";
import { type GetServerSideProps } from "next";
import { getServerAuthSession } from "~/server/auth";

export const getServerSideProps: GetServerSideProps = async (ctx) ⇒ {
  const session = await getServerAuthSession(ctx);
  return {
    props: { session },
  };
};

function Home() {
  const { data: session } = useSession();
```

# 認証実装②（SSRで認証）

SSRで認証したい場合は以下のように実装

```
import { signIn } from "next-auth/react";
import { type GetServerSideProps } from "next";
import { getServerAuthSession } from "~/server/auth";

export const getServerSideProps: GetServerSideProps = async (ctx) ⇒ {
  const session = await getServerAuthSession(ctx);
  return {
    props: { session },
  };
};

function Home() {
  const { data: session } = useSession();
```

# 認証実装③（RSCで認証）

RSCで認証したい場合は以下のように実装
現状はこれが最速の認証方法

```
import { getServerSession } from "next-auth/next"
import { authOptions } from "~/server/auth";

export default async function Home() {
  const session = await getServerSession(authOptions)
```

# 認証実装③（RSCで認証）

RSCで認証したい場合は以下のように実装
現状はこれが最速の認証方法

```
import { getServerSession } from "next-auth/next"
import { authOptions } from "~/server/auth";

export default async function Home() {
  const session = await getServerSession(authOptions)
```

# 認証実装③（RSCで認証）

RSCで認証したい場合は以下のように実装
現状はこれが最速の認証方法

```
import { getServerSession } from "next-auth/next"
import { authOptions } from "~/server/auth";

export default async function Home() {
  const session = await getServerSession(authOptions)
```

# tRPCでSSRを実装①

T3 StackではtRPCでデータ取得を行っている
SSRに対応するには、以下のoptionを**true**にするだけでOK 👌

```
import { createTRPCNext } from "@trpc/next";

export const api = createTRPCNext<AppRouter>({
  /**
   * Whether tRPC should await queries when server rendering pages.
   *
   * @see https://trpc.io/docs/nextjs#ssr-boolean-default-false
   */
  ssr: true,
```

# tRPCでSSRを実装①

T3 StackではtRPCでデータ取得を行っている
SSRに対応するには、以下のoptionを**true**にするだけでOK 👌

```
import { createTRPCNext } from "@trpc/next";

export const api = createTRPCNext<AppRouter>({
  /**
   * Whether tRPC should await queries when server rendering pages.
   *
   * @see https://trpc.io/docs/nextjs#ssr-boolean-default-false
   */
  ssr: true,
```

# tRPCでSSRを実装②

**ssr:true**にするとtRPCでアクセスしている箇所をサーバーサイドで**prefetch**する挙動になる
以下のようなコードを実行した場合、クライアントサイドではキャッシュから取得するの動作になる

```
function Schedule() {
  const router = useRouter();
  const { id } = router.query;

  const schedules = api.schedule.fetch.useQuery({ urlId: String(id) });
```

- Server-Side Rendering | tRPC

# tRPCでSSRを実装②

**ssr:true**にするとtRPCでアクセスしている箇所をサーバーサイドで**prefetch**する挙動になる
以下のようなコードを実行した場合、クライアントサイドではキャッシュから取得するの動作になる

```
function Schedule() {
  const router = useRouter();
  const { id } = router.query;

  const schedules = api.schedule.fetch.useQuery({ urlId: String(id) });
```

- Server-Side Rendering | tRPC

# tRPCでSSRを実装②

**ssr:true**にするとtRPCでアクセスしている箇所をサーバーサイドで**prefetch**する挙動になる
以下のようなコードを実行した場合、クライアントサイドではキャッシュから取得するの動作になる

```
function Schedule() {
  const router = useRouter();
  const { id } = router.query;

  const schedules = api.schedule.fetch.useQuery({ urlId: String(id) });
```

- Server-Side Rendering | tRPC

# 補足: スライドで参照している開発物の紹介

- Repository
  - OOMAKA
- サービスURL
  - OOMAKA | 年間スケジュール、まとめるなら
- SSRしている画面を軽くDEMO

# まとめ

- T3 Stackを使うとSSR周りは、かなり楽
- **getServerSideProps**でゴニョゴニョする必要がなくなって開発が捗る
- Next Authは現状はNext.js専用だが、今後のロードマップで名前をAuthに変更して別フレームワークでも使用可能になる予定

ご清聴ありがとうございました🎉