Temporal + Next.js Example

This is a starter project for creating resilient Next.js applications with Temporal. Whenever our API routes need to do any of the following, we can greatly increase our code's fault tolerance by using Temporal:

- Perform a series of network requests (to a database, another function, an internal service, or an external API), any of which may fail. Temporal will automatically set timeouts and retry, as well as remember the state of execution in the event of power loss.
- Long-running tasks
- Cron jobs

The starter project has this logic flow:

- Load `localhost:3000`
- Click the "Create order" button
- The click handler POSTs a new order object to `/api/orders`
- The serverless function at `pages/api/orders/index.ts`:
  - Parses the order object
  - Tells Temporal Server to start a new Order Workflow, and passes the user ID and order info
  - Waits for the Workflow result and returns it to the client
- Temporal Server puts Workflow tasks on the `orders` task queue
- The Worker executes chunks of Workflow and Activity code. The Activity code logs:

```
Reserving 2 of item B102
Charging user 123 for 2 of item B102
```

- The Workflow completes and Temporal Server sends the result back to the serverless function, which returns it to the client, which alerts the result.

Here is the Temporal code:

- The Workflow: `temporal/src/workflows/order.ts`
- The Activites: `temporal/src/activities/{payment|inventory}.ts`

There are three parts of this starter project that are left unimplemented:

- Authentication (currently, the client sends their user ID in the authorization header): `pages/api/orders/index.ts`
- Doing database queries to check and alter inventory levels: `temporal/src/activities/inventory.ts`
- Communicating with a payments service (or function) to charge the user: `temporal/src/activities/payment.ts`

Deploy your own

Web server

The Next.js server can be deployed using Vercel:

Worker

One or more instances of the worker (`temporal/src/worker/`) can be deployed to a PaaS (each worker is a long-running Node process, so it can't run on a FaaS/serverless platform).

Temporal Server

Temporal Server is a cluster of internal services, a database of record, and a search database. It can be run locally with Docker Compose and deployed with a container orchestration service like Kubernetes or ECS.

How to use

Execute `create-next-app` with npm or Yarn to bootstrap the example:

```
npx create-next-app --example with-temporal next-temporal-app
# or
yarn create next-app --example with-temporal next-temporal-app
# or
pnpm create next-app -- --example with-temporal next-temporal-app
```

The Temporal Node SDK requires Node `>= 14` , `node-gyp` , and Temporal Server. Once you have everything installed, you can develop locally with the below commands in four different shells:

In the Temporal Server docker directory:

```
docker-compose up
```

In the `next-temporal-app/` directory:

```
npm run dev
```

```
npm run build-worker.watch
```

```
npm run start-worker
```