Running Next.JS and React /inside/ of ActionHero

This server will render dynamic next.js/react pages on some routes, and normal ActionHero API requests on others. This configuration works with both Next and ActionHero hot reloading of code.

A more detailed example showcasing how to use fetch and web sockets to interact with your API can be found here: https://github.com/actionhero/next-in-actionhero

How to use

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

```
npx create-next-app --example custom-server-actionhero custom-server-actionhero-app
# or
yarn create next-app --example custom-server-actionhero custom-server-actionhero-app
# or
pnpm create next-app -- --example custom-server-actionhero custom-server-actionhero-app
```

How does this work?

1. Create an initializer to load next.js and create a handler that can extract the normal node `req` and `res` from the connection

```
// initializers/next.js

const { Initializer, api } = require('actionhero')
const next = require('next')

module.exports = class NextInitializer extends Initializer {
  constructor() {
    super()
    this.name = 'next'
  }

  async initialize() {
    api.next = {
      render: async (connection) => {
        if (connection.type !== 'web') {
          throw new Error('Connections for NEXT apps must be of type "web"')
        }
        const req = connection.rawConnection.req
        const res = connection.rawConnection.res
        return api.next.handle(req, res)
      },
```

```
    }

    api.next.dev = api.env === 'development'
    if (api.next.dev) {
      api.log('Running next in development mode...')
    }

    api.next.app = next({ dev: api.next.dev })
    api.next.handle = api.next.app.getRequestHandler()
    await api.next.app.prepare()
  }

  async stop() {
    await api.next.app.close()
  }
}
```

2. Create an action which will run the above `api.next.render(connection)`. Note that we will not be relying on ActionHero to respond to the client's request in this case, and leave that up to next (via: `data.toRender = false`)

```
// actions/next.js

const { Action, api } = require('actionhero')

module.exports = class CreateChatRoom extends Action {
  constructor() {
    super()
    this.name = 'render'
    this.description = 'I render the next.js react website'
  }

  async run(data) {
    data.toRender = false
    return api.next.render(data.connection)
  }
}
```

3. Tell ActionHero to use the api rather than the file server as the top-level route in `api.config.servers.web.rootEndpointType = 'api'`. This will allows "/" to listen to API requests. Also update `api.config.general.paths.public = [ path.join(__dirname, '/../static') ]`. In this configuration, the next 'static' renderer will take priority over the ActionHero 'public file' api. Note that any static assets (CSS, fonts, etc.) will need to be in "./static" rather than "./public".

Note that this is where the websocket server, if you enable it, will place the

`ActionheroWebsocketClient` library.

4. Configure a wild-card route at the lowest priority of your GET handler to catch all web requests that aren't caught by other actions:

```
// config/routes.js

exports['default'] = {
  routes: (api) => {
    return {
      get: [
        { path: '/time', action: 'time' },

        { path: '/', matchTrailingPathParts: true, action: 'render' },
      ],
    }
  },
}
```