Environment Variables

▶ **Examples**

Next.js comes with built-in support for environment variables, which allows you to do the following:

- Use `.env.local` to load environment variables
- Expose environment variables to the browser by prefixing with `NEXT_PUBLIC_`

Loading Environment Variables

Next.js has built-in support for loading environment variables from `.env.local` into `process.env`.

.env.local

```
DB_HOST=localhost
DB_USER=myuser
DB_PASS=mypassword
```

This loads `process.env.DB_HOST`, `process.env.DB_USER`, and `process.env.DB_PASS` into the Node.js environment automatically allowing you to use them in Next.js data fetching methods and API routes.

For example, using `getStaticProps`:

pages/index.js

```
export async function getStaticProps() {
  const db = await myDB.connect({
    host: process.env.DB_HOST,
    username: process.env.DB_USER,
    password: process.env.DB_PASS,
  });
  // ...
}
```

Referencing Other Variables

Next.js will automatically expand variables that use `$` to reference other variables e.g. `$VARIABLE` inside of your `.env*` files. This allows you to reference other secrets. For example:

.env

```
HOSTNAME=localhost
PORT=8080
HOST=http://$HOSTNAME:$PORT
```

In the above example, `process.env.HOST` would be set to `http://localhost:8080`.

> **Note**: If you need to use variable with a `$` in the actual value, it needs to be escaped e.g. `\$`.

Exposing Environment Variables to the Browser

By default environment variables are only available in the Node.js environment, meaning they won't be exposed to the browser.

In order to expose a variable to the browser you have to prefix the variable with `NEXT_PUBLIC_`. For example:

Terminal

```
NEXT_PUBLIC_ANALYTICS_ID=abcdefghijk
```

This loads `process.env.NEXT_PUBLIC_ANALYTICS_ID` into the Node.js environment automatically, allowing you to use it anywhere in your code. The value will be inlined into JavaScript sent to the browser because of the `NEXT_PUBLIC_` prefix. This inlining occurs at build time, so your various `NEXT_PUBLIC_` envs need to be set when the project is built.

pages/index.js

```js
import setupAnalyticsService from '../lib/my-analytics-service';

```

```
// 'NEXT_PUBLIC_ANALYTICS_ID' can be used here as it's prefixed by 'NEXT_PUBLIC_'.
// It will be transformed at build time to `setupAnalyticsService('abcdefghijk')`.
setupAnalyticsService(process.env.NEXT_PUBLIC_ANALYTICS_ID);

function HomePage() {
  return <h1>Hello World</h1>;
}

export default HomePage;
```

Note that dynamic lookups will *not* be inlined, such as:

```
// This will NOT be inlined, because it uses a variable
const varName = 'NEXT_PUBLIC_ANALYTICS_ID';
setupAnalyticsService(process.env[varName]);

// This will NOT be inlined, because it uses a variable
const env = process.env;
setupAnalyticsService(env.NEXT_PUBLIC_ANALYTICS_ID);
```

Default Environment Variables

In general only one `.env.local` file is needed. However, sometimes you might want to add some defaults for the `development` (`next dev`) or `production` (`next start`) environment.

Next.js allows you to set defaults in `.env` (all environments), `.env.development` (development environment), and `.env.production` (production environment).

`.env.local` always overrides the defaults set.

> **Note**: `.env`, `.env.development`, and `.env.production` files should be included in your repository as they define defaults. **`.env*.local` should be added to `.gitignore`**, as those files are intended to be ignored. `.env.local` is where secrets can be stored.

Environment Variables on Vercel

When deploying your Next.js application to Vercel↗, Environment Variables can be configured in the Project Settings↗.

All types of Environment Variables should be configured there. Even Environment Variables used in Development – which can be downloaded onto your local device ↗ afterwards.

If you've configured Development Environment Variables ↗ you can pull them into a `.env.local` for usage on your local machine using the following command:

```
Terminal

vercel env pull .env.local
```

Test Environment Variables

Apart from `development` and `production` environments, there is a 3rd option available: `test`. In the same way you can set defaults for development or production environments, you can do the same with a `.env.test` file for the `testing` environment (though this one is not as common as the previous two). Next.js will not load environment variables from `.env.development` or `.env.production` in the `testing` environment.

This one is useful when running tests with tools like `jest` or `cypress` where you need to set specific environment vars only for testing purposes. Test default values will be loaded if `NODE_ENV` is set to `test`, though you usually don't need to do this manually as testing tools will address it for you.

There is a small difference between `test` environment, and both `development` and `production` that you need to bear in mind: `.env.local` won't be loaded, as you expect tests to produce the same results for everyone. This way every test execution will use the same env defaults across different executions by ignoring your `.env.local` (which is intended to override the default set).

Note: similar to Default Environment Variables, `.env.test` file should be included in your repository, but `.env.test.local` shouldn't, as `.env*.local` are intended to be ignored through `.gitignore`.

While running unit tests you can make sure to load your environment variables the same way Next.js does by leveraging the `loadEnvConfig` function from the `@next/env` package.

```
// The below can be used in a Jest global setup file or similar for your testing set-up
import { loadEnvConfig } from '@next/env';

export default async () => {
  const projectDir = process.cwd();
  loadEnvConfig(projectDir);
};
```

Environment Variable Load Order

Environment variables are looked up in the following places, in order, stopping once the variable is found.

1. `process.env`
2. `.env.$(NODE_ENV).local`
3. `.env.local` (Not checked when `NODE_ENV` is `test`.)
4. `.env.$(NODE_ENV)`
5. `.env`

For example, if `NODE_ENV` is `development` and you define a variable in both `.env.development.local` and `.env`, the value in `.env.development.local` will be used.

> **Note:** The allowed values for `NODE_ENV` are `production`, `development` and `test`.

Good to know

- If you are using a `/src` directory, `env.*` files should remain in the root of your project.