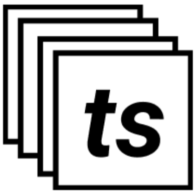
How to Setup a TypeScript + Node.js Project

TypeScript

Last updated Aug 29th, 2019

In this guide, we walk through the process of creating a TypeScript project from scratch with cold-reloading, and scripts for building, development, and production environments.

guides typescript starters



We talk about a lot of **advanced Node.js and TypeScript** concepts on this blog, particularly Domain Driven Design and large-scale enterprise interested in seeing what a basic TypeScript starter project looks like, I've put together just that.

Prequisites

- You should have Node and npm installed
- You should be familiar with Node and the npm ecosystem
- You have a code editor installed (preferably VS Code, it's the champ for TypeScript)

Goals

In this short guide, I'll walk you through the process of creating a basic TypeScript application and compiling it. It's actually *really* easy!

- View the source

Afterwards, we'll setup a few scripts for hot-reloading in `development`, building for `production`, and running in `production`.

About TypeScript

TypeScript, developed and appropriated labeled by Microsoft as "JavaScript that scales", is a superset of JavaScript, meaning that everything JavaScript can do, TypeScript can do (~~and more~~ better).

2. Provide JavaScript developers with the ability to utilize **planned features from future JavaScript editions** against *current JavaScript engines*.

We use TypeScript for most of the topics on this blog because it's a lot better suited for creating long-lasting software and having the compiler help catch bugs and validate types is tremendously helpful.

Initial Setup

Let's create a folder for us to work in.

```
mkdir typescript-starter
cd typescript-starter
```

Next, we'll setup the project `package.json` and add the dependencies.

Setup Node.js package.json

Using the `-y` flag when creating a `package.json` will simply approve all the defaults.

```
npm init -y
```

Add TypeScript as a dev dependency

This probably doesn't come as a surprise ;)

After we install `typescript`, we get access to the command line TypeScript compiler through the `tsc` command. More on that below.

Install ambient Node.js types for TypeScript

TypeScript has Implicit, Explicit, and *Ambient* types. Ambient types are types that get added to the global execution scope. Since we're using Node, it would be good if we could get type safety and auto-completion on the Node apis like `file`, `path`, `process`, etc. That's what installing the DefinitelyTyped type definition for Node will do.

```
npm install @types/node --save-dev
```

Create a `tsconfig.json`.

The `tsconfig.json` is where we define the TypeScript compiler options. We can create a tsconfig with several options set.

```
npx tsc --init --rootDir src --outDir build \
--esModuleInterop --resolveJsonModule --lib es6 \
--module commonjs --allowJs true --noImplicitAny true
```

- `rootDir`: This is where TypeScript looks for our code. We've configured it to look in the `src/` folder. That's where we'll write our TypeScript.
- `outDir`: Where TypeScript puts our compiled code. We want it to go to a `build/` folder.

Modules, etc). For a topic that requires a much longer discussion, if we're using `commonjs` as our module system (for Node apps, you should be), then we need this to be set to `true` .

- `resolveJsonModule` : If we use JSON in this project, this option allows TypeScript to use it.
- `lib` : This option adds *ambient* types to our project, allowing us to rely on features from different Ecmascript versions, testing libraries, and even the browser DOM api. We'd like to utilize some `es6` language features. This all gets compiled down to `es5` .
- `module` : `commonjs` is the standard Node module system in 2019. Let's use that.
- `allowJs` : If you're converting an old JavaScript project to TypeScript, this option will allow you to include `.js` files among `.ts` ones.
- `noImplicitAny` : In TypeScript files, don't allow a type to be unexplicitly specified. Every type needs to either have a specific type or be explicitly declared `any` . No implicit `any` s.

At this point, you should have a `tsconfig.json` that looks like this:

```
{
  "compilerOptions": {
    /* Basic Options */
    // "incremental": true,                   /* Enable incremental compilation */
    "target": "es5",                          /* Specify ECMAScript target version: 'ES3' (def
    "module": "commonjs",                     /* Specify module code generation: 'none', 'comm
    // "declaration": true,                   /* Generates corresponding '.d.ts' file. */
    // "declarationMap": true,                /* Generates a sourcemap for each corresponding
    // "sourceMap": true,                     /* Generates corresponding '.map' file. */
    // "outFile": "./",                       /* Concatenate and emit output to single file. *
    "outDir": "build",                          /* Redirect output structure to the directory.
    "rootDir": "src",                         /* Specify the root directory of input files. Us
    // "composite": true,                     /* Enable project compilation */
    // "tsBuildInfoFile": "./",               /* Specify file to store incremental compilation
    // "removeComments": true,                /* Do not emit comments to output. */
    // "noEmit": true,                        /* Do not emit outputs. */
    // "importHelpers": true,                 /* Import emit helpers from 'tslib'. */
    // "downlevelIteration": true,            /* Provide full support for iterables in 'for-of
    // "isolatedModules": true,               /* Transpile each file as a separate module (sim

    /* Strict Type-Checking Options */
    "strict": true,                           /* Enable all strict type-checking options. */
    "noImplicitAny": true,                    /* Raise error on expressions and declarations w
    // "strictNullChecks": true,              /* Enable strict null checks. */
    // "strictFunctionTypes": true,           /* Enable strict checking of function types. */
    // "strictBindCallApply": true,           /* Enable strict 'bind', 'call', and 'apply' met
    // "strictPropertyInitialization": true,  /* Enable strict checking of property initializa
    // "noImplicitThis": true,                /* Raise error on 'this' expressions with an imp
    // "alwaysStrict": true,                  /* Parse in strict mode and emit "use strict" fo

    /* Additional Checks */
    // "noUnusedLocals": true,                /* Report errors on unused locals. */
    // "noUnusedParameters": true,            /* Report errors on unused parameters. */
    // "noImplicitReturns": true,             /* Report error when not all code paths in funct
    // "noFallthroughCasesInSwitch": true,    /* Report errors for fallthrough cases in switch

    /* Module Resolution Options */
    // "moduleResolution": "node",            /* Specify module resolution strategy: 'node' (N
    // "baseUrl": "./",                       /* Base directory to resolve non-absolute module
    // "paths": {},                           /* A series of entries which re-map imports to l
    // "rootDirs": [],                        /* List of root folders whose combined content r
    // "typeRoots": [],                       /* List of folders to include type definitions f
    // "types": [],                           /* Type declaration files to be included in comp
    // "allowSyntheticDefaultImports": true,  /* Allow default imports from modules with no de
    "esModuleInterop": true,                  /* Enables emit interoperability between CommonJ
```

khalilstemmler.com

```
    // "sourceRoot": "",                      /* Specify the location where debugger should lo
    // "mapRoot": "",                         /* Specify the location where debugger should lo
    // "inlineSourceMap": true,               /* Emit a single file with source maps instead o
    // "inlineSources": true,                 /* Emit the source alongside the sourcemaps with

    /* Experimental Options */
    // "experimentalDecorators": true,        /* Enables experimental support for ES7 decorato
    // "emitDecoratorMetadata": true,         /* Enables experimental support for emitting typ

    /* Advanced Options */
    "resolveJsonModule": true                 /* Include modules imported with '.json' extensi
  }
}
```

We can go ahead and clean the commented out stuff that we don't need. Our `tsconfig.json` should look like this:

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "lib": ["es6"],
    "allowJs": true,
    "outDir": "build",
    "rootDir": "src",
    "strict": true,
    "noImplicitAny": true,
    "esModuleInterop": true,
    "resolveJsonModule": true
  }
}
```

We're set to run our first TypeScript file.

```
mkdir src
touch src/index.ts
```

And let's write some code.

```
console.log('Hello world!')
```

Compiling our TypeScript

To compile our code, we'll need to run the `tsc` command using `npx`, the Node package executer. `tsc` will read the `tsconfig.json` in the current directory, and apply the configuration against the TypeScript compiler to generate the compiled JavaScript code.

```
npx tsc
```

Our compiled code

Check out `build/index.js`, we've compiled our first TypeScript file.

```
"use strict";
console.log('Hello world!');
```

Useful configurations & scripts

Cold reloading

watch for changes to our code and automatically restart when a file is changed.

```
npm install --save-dev ts-node nodemon
```

Add a `nodemon.json` config.

```
{
  "watch": ["src"],
  "ext": ".ts,.js",
  "ignore": [],
  "exec": "ts-node ./src/index.ts"
}
```

And then to run the project, all we have to do is run `nodemon`. Let's add a script for that.

```
"start:dev": "nodemon",
```

By running `npm run start:dev`, `nodemon` will start our app using `ts-node ./src/index.ts`, watching for changes to `.ts` and `.js` files from within `/src`.

Creating production builds

In order to *clean* and compile the project for production, we can add a `build` script.

Install `rimraf`, a cross-platform tool that acts like the `rm -rf` command

```
npm install --save-dev rimraf
```

And then, add this to your `package.json`.

```
"build": "rimraf ./build && tsc",
```

Now, when we run `npm run build`, `rimraf` will remove our old `build` folder before the TypeScript compiler emits new code to `dist`.

Production startup script

In order to start the app in production, all we need to do is run the `build` command first, and then execute the compiled JavaScript at `build/index.js`.

The startup script looks like this.

```
"start": "npm run build && node build/index.js"
```

I told you it was simple! Now, off you go. **Create great things, my friend.**

View the source

A reminder that you can view [the entire source code](#) for this here.

Scripts overview

Starts the application in development using `nodemon` and `ts-node` to do cold reloading.

`npm run build`

Builds the app at `build`, cleaning the folder first.

`npm run start`

Starts the app in production by first building the project with `npm run build`, and then executing the compiled JavaScript at `build/index.js`.

Linting

Oh yeah, linting is another thing you'll most likely want to do. If you're interested in that, read the next post, "How to use ESLint with TypeScript".

Discussion

Liked this? Sing it loud and proud 👨‍🎤.

Share on Twitter

47 Comments

Name

khalilstemmler.com

Submit

Kyrell Dixon 3 years ago

Great article and beautiful site! I've been thinking about writing one like this myself.

> **Khalil Stemmler** 3 years ago
>
> Thanks man! You definitely should. GatsbyJS is my favourite thing.

Paul Edwards 3 years ago

Nice clear intro. I think the build task should be - "build": "rimraf ./build && node build/index.js" - the "dist" folder is actually the "build" folder.

> **Khalil Stemmler** 3 years ago
>
> Good eye! Fixed it. Thanks :)

Igor 3 years ago

The files property in the tsconfig.json file makes problem when trying to compile the code with tsc. I've need to comment or remove the property so I can transpile typescript to javascript. I have this issue with typescript version 3.6.3

Josh 3 years ago

Thanks for this tutorial.

polo 3 years ago

Thanks dude finally a clear, no-nonsense tutorial

khalilstemmler.com

Thank you for this Khalil.This helped me setup my project right

Jan 2 years ago

Hi, I am interested to hear what do you think about nest.js? as it seems to do most of the stuff you are talking about. thx

Rafael Rocha 2 years ago

Great article!
What would be the advantages/disadvantages of having the same setup but using babel instead?
I've seen a lot of starters, even in Microsofts' repo, using it and I'm wondering if it's just for consistency with FE or if it brings anything new.
Keep up the great writing!

Adam 2 years ago

Crazy refreshing to see someone write up for a (ironically) 'vanilla' Typescript setup. I can't tell you how much time you've saved me drilling into the frameworks that do it out of the box and don't explain.

Event the TypeScript github just consists of cloning their repos. So frustrating.

Much love.

Elliott W 2 years ago

You should modify this tutorial to use **ts-node-dev** instead of **ts-node** and **nodemon**. It's easier to setup and is significantly faster because it only recompiles the files that changed :D

Frank P 2 years ago

Very nice: to the point & 100% correct!

Sathish 2 years ago

This is the best detailed explanation that I have seen which could be understood by all. Kudos for the great job.

rokas 2 years ago

Great article!! greetings from Spain. This help me :)

sooperdooper 2 years ago

finally good one too much mess and pckgs in node template project, node is finally beutiful and simple big Thx :)

Vitor Batista 2 years ago

Great article!
I only changed my .gitignore adding the folder "build" to be ignored

Herick 2 years ago

Thanks for sharing this, man, it helped me a lot!
Btw right before the section "Production startup script" where you wrote "dist" did you perhaps mean "build"?
Anyway, thanks again!

mknig 2 years ago

great! Just add "in package.json" in the "...Let's add a script for that." section about config nodemon ...i had to find out, a node newbee

Prathamesh Madur 2 years ago

Very Nice! Keep it up.

Dima 2 years ago

Good article, thanks!

Mathew 2 years ago

```
Where to add the "start:dev": "nodemon",
```

I mean in which file?

Lahiru Udayanga 2 years ago

khalilstemmler.com

You are real MVP, thanks!

Marcus 2 years ago

Thanks for a helpful and easy to follow tutorial.

Lee Wilson a year ago

Cheers bud!

Eric a year ago

finally something relevant that actually works! cheers :)

Paulo a year ago

Finally I can understand Lint! Thank you!

One thing: node_modules files are ignored by default

Ivaan a year ago

Thank you my friend, that was really helpful

andy.h.nguyen a year ago

Nice article, detailed explanation keep up the good work bro.

Jeremy a year ago

I like having concurrently in the mix.. npm start and thats it.

"scripts": {
"start": "concurrently npm:start:*",
"start:build": "tsc -w",
"start:run": "nodemon build/index.js",
},

jonah a year ago

Just found your site. Such great information here! All the straightforward typescript information is so good. :) Thanks so much.

jithin dfeverx a year ago

Man super useful article.

khalilstemmler.com

```
{
  "watch": ["src"],
  "ext": ".ts,.js",
  "ignore": [],
  "exec": "ts-node ./src/index.ts"
}
```

the above code exce statement gave an error ts-node error,

```
{
   "watch": ["src"],
  "ext": ".ts,.js",
  "ignore": [],
    "exec": "npx ts-node ./src/index.ts"
}
```

Dev a year ago

Thank you!. Was super easy to follow.

I made a few modifications though. In nodemon.json instead of "ts-node", "npx ts-node". And in package.json instead of "nodemon" "npx nodemon".

If npx is used to execute the node commands, I need not have those node commands installed globally. I think this especially helps if I have multiple people working on the project. I wont have to specify extra instructions to first install the required global packages

hakim 9 months ago

wow,That was nice!

harsh 7 months ago

'ts-node' is not recognized as an internal or external command

Ankush Agrawal 6 months ago

Thank you for writing such an awesome article. This is a very informative blog for an

Great article! Helped me migrate a JS node application I was developing into a more robust Typescript powered app. I had to include typescript, rimraf and ts-node as production dependencies because I use a Dockerfile to build and start my application and those libraries are needed to create the build and compile to JS.

MarioE777 3 months ago

✨ Pretty clear info ✨
Thanks a lot!

Syed Masani 3 months ago

Excellent article. Well organized and no unless talk.

Viacheslav 2 months ago

Very good article! Didn't transpile util this: ***npm install @types/node --save-dev*** was added.

susant dasari 2 months ago

awesome blog , things are explained very clearly

Yash Tibrewal 2 months ago

Thanks Khalil.

Xande Torres a month ago

It is very nice guide to start node project with advanced settings and configuration. Thank you very much for you sharing your knoweledge with us. you are the best in a fact

Prem Sai Vittal 23 days ago

Great Article

Sabloger 12 days ago

Great job bro!

Dimmy 3 days ago

Cool article, thanks!
Must note that tsconfig.json need an extra line to specify source directory:

It is not in the article, but it is in related repository: https://github.com/stemmlerjs/simple-typescript-starter/blob/master/tsconfig.json

Stay in touch!

We're just getting started 🔥 Interested in how to write professional JavaScript and TypeScript? Join 10000+ other developers learning about Domain-Driven Design and Enterprise Node.js. I won't spam ya. 🖖 Unsubscribe anytime.

Email | Get notified

About the author

Khalil Stemmler,
Developer Advocate @ Apollo GraphQL ⚡

Khalil is a software developer, writer, and musician. He frequently publishes articles about Domain-Driven Design, software design and Advanced TypeScript & Node.js best practices for large-scale applications.

Follow @stemmlerjs 6,940 followers

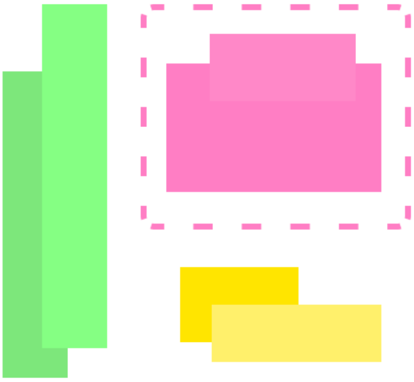
Follow 2,593

khalilstemmler.com

View more in TypeScript

You may also enjoy...

A few more related articles

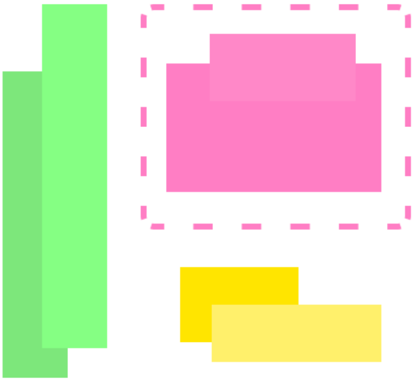


khalilstemmler.com

Test-Driven Development

typescript test-driven development bdd core code infrastructure code

In the real-world, there's more to test than pure functions and React components. We have entire bodies of code that rely on datab...



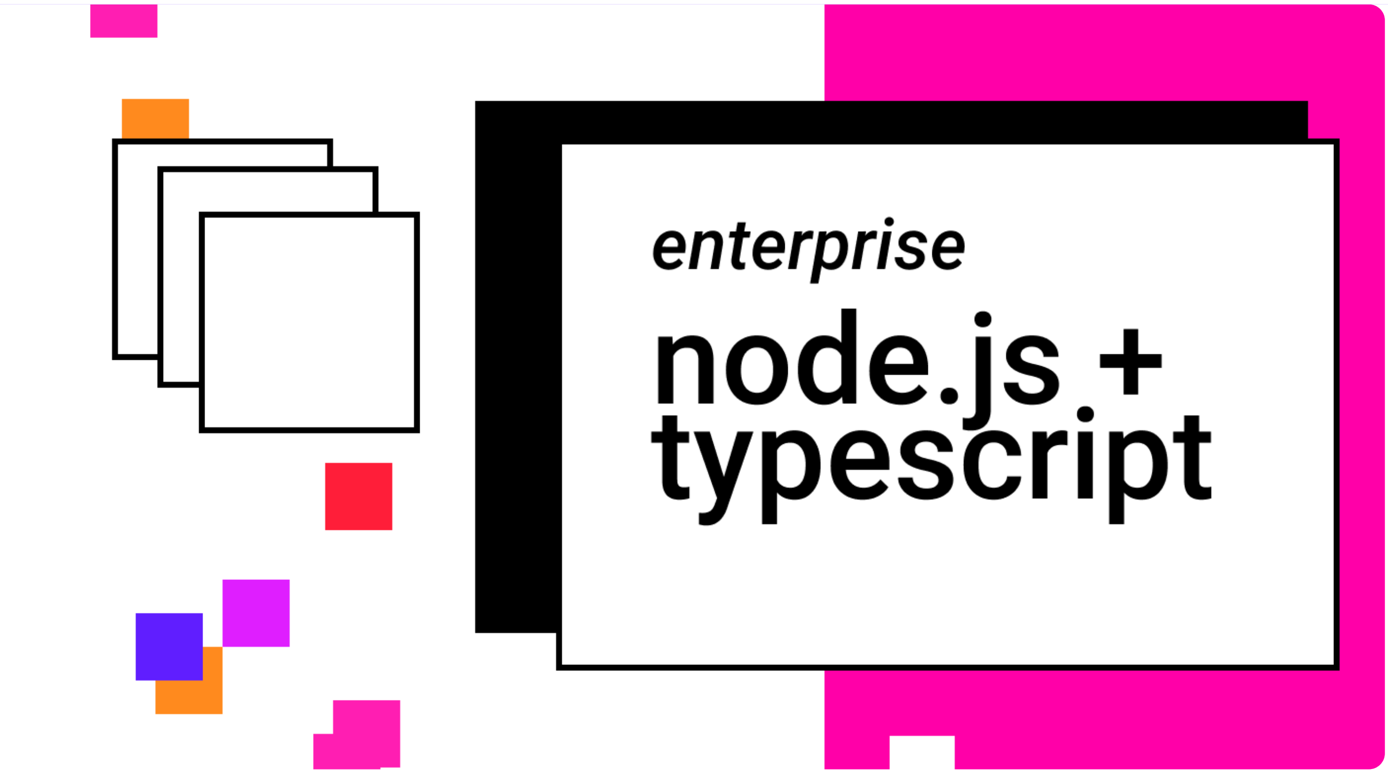
How to Mock without Providing an Implementation in TypeScript

Test-Driven Development

typescript test-driven development mocking jest ts-auto-mock

Having to provide an implementation everytime you create a test double leads to brittle tests. In this post, we learn how to creat...

khalilstemmler.com

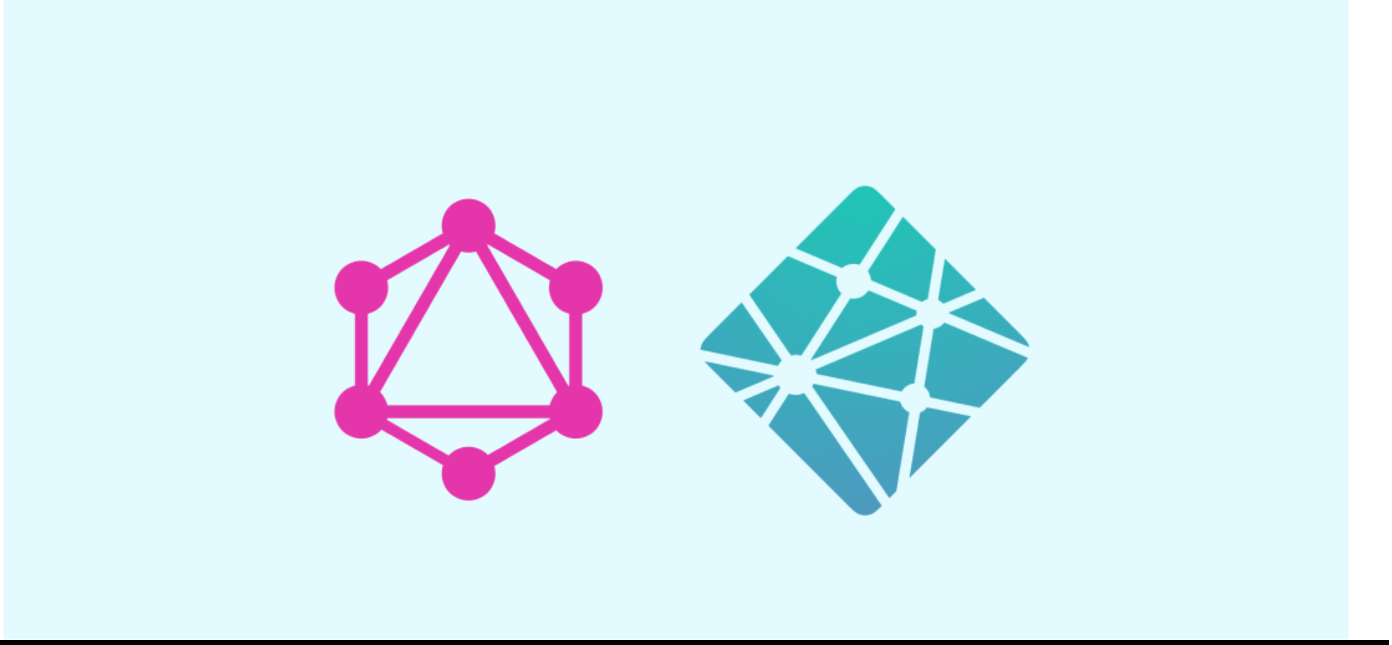


Use DTOs to Enforce a Layer of Indirection | Node.js w/ TypeScript

Enterprise Node + TypeScript

node.js typescript express.js enterprise software dto

DTOs help you create a more stable RESTful API; they protect your API clients from changes made on the server.



khalilstemmler.com

How to Deploy a Serverless GraphQL API on Netlify [Starters]

GraphQL

tutorial netlify graphql serverless starters

Exploring two starter projects you can use to deploy a serverless GraphQL API on Netlify with or without TypeScript.

I'm Khalil. I teach advanced TypeScript & Node.js best practices for large-scale applications. Learn to write testable, flexible, and maintainable software.

Menu

- About
- Articles
- Blog
- Courses
- Books
- Newsletter
- Wiki

Contact

- Email
- @stemmlerjs

khalilstemmler.com

GitHub

Twitter

Instagram

LinkedIn