설정(Configuration)이란?

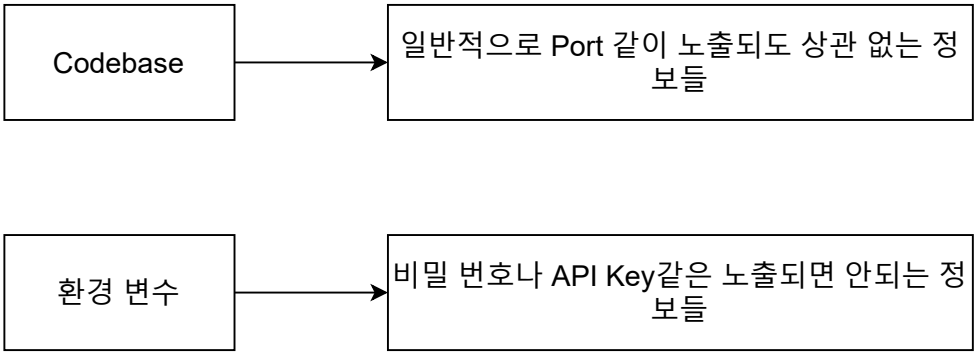
소스 코드안에서 어떠한 코드들은 개발 환경이나 운영 환경에 이러한 환경에 따라서 다르게 코드를 넣어줘야 할 때가 있으며, 남들에게 노출 되지 않아야 하는 코드들도 있습니다. 이러한 코드들을 위해서 설정 파일을 따로 만들어서 보관해주겠습니다.

설정 파일은...

runtime 도중에 바뀌는 것이 아닌 애플리케이션이 시작할 때 로드가 되어서 그 값들을 정의하여 줍니다. 그리고 설정 파일은 여러가지 파일 형식을 사용할 수 있습니다. 예로는 XML, JSON, YAML, Environmant Variables 같이 많은 형식을 이용할 수 있습

Codebase VS Enviroment Variables(환경 변수)

설정을 할 때 여러가지 형식으로 할 수 있다고 했습니다.
그곳에서 XML, JSON, YAML 같은 경우는 Codebase에 해당하며
그리고 다른 방법은 환경 변수로 할 수 있습니다. 주로 이 둘을 나눠서 하는 이유는 비밀 번호와 API Key 같은 남들에게 노출 되면 안되는 정보들을 주로 환경 변수를 이용해서 처리해줍니다



설정하기 위해서 필요한 모듈

윈도우에서는 win-node-env 를 설치해야 합니다.
(왜냐면 윈도우에서는 기본적으로 환경변수를 지원하지 않기 때문입니다.)

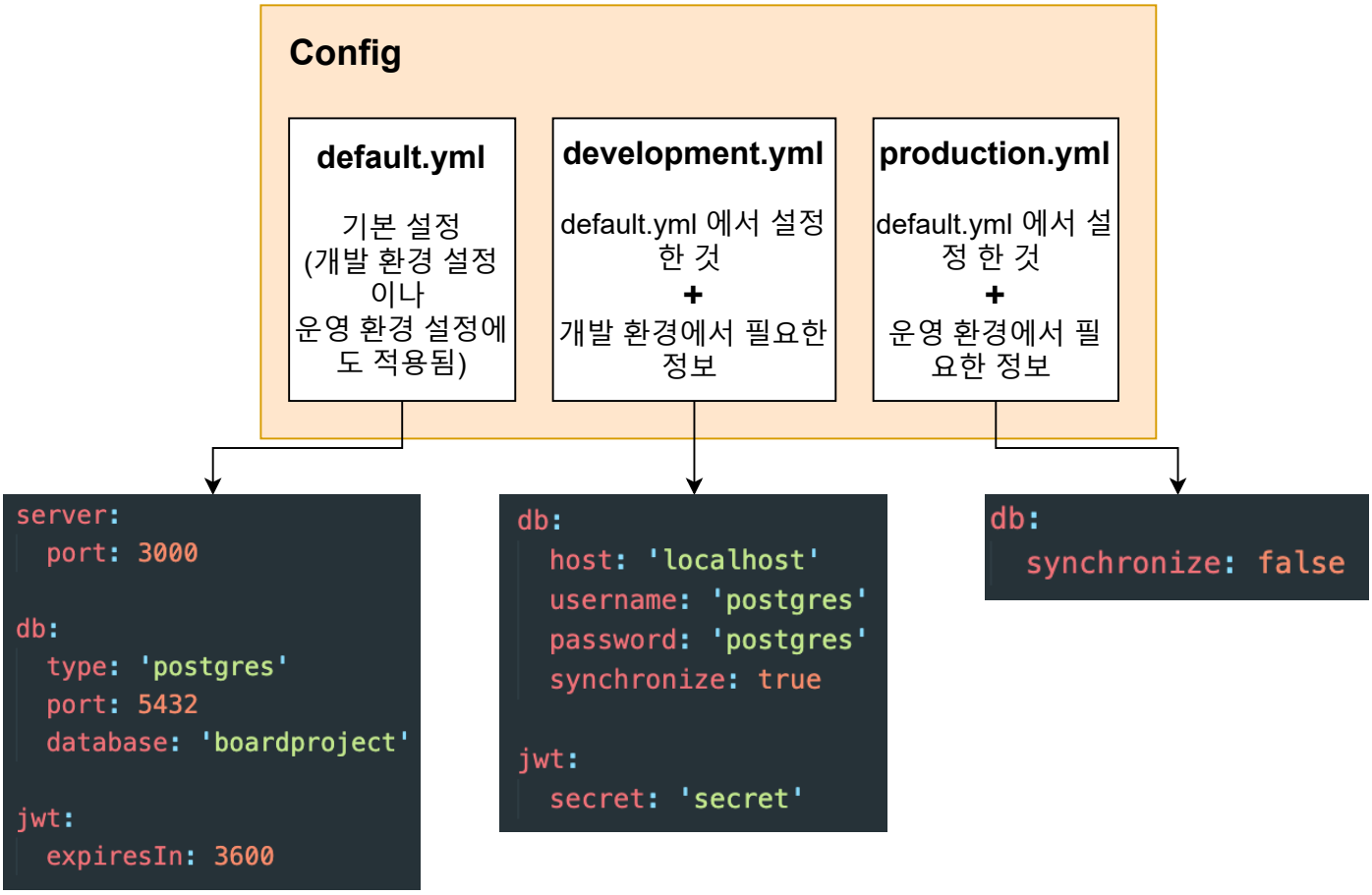
npm install -g win-node-env

그리고 윈도우와 맥 모두에서는 config라는 모듈을 설치받아야 합니다.

Config 모듈을 이용한 설정 파일 생성

1. 루트 디렉토리에 config 라는 폴더를 만든 후에 그 폴더 안에 JSON이나 YAML 형식의 파일을 생성합니다. config/default.yaml

2. config 폴더 안에 default.yml, development.yml, 그리고 production.yml 파일을 생성하겠습니다.



Config 폴더 안에 저장된 것들을 사용하는 방법

1. 어느 파일에서든지 config 모듈을 import 해서 사용하면 됩니다.

import * as config from 'config';

2. 그리고 config.get('server') => 이렇게 하면 { port: 3000 } 이렇게 나옵니다.

```
import { NestFactory } from '@nestjs/core';
import { Logger } from '@nestjs/common';
import { AppModule } from './app.module';
import * as config from 'config';

async function bootstrap() {
  const logger = new Logger();
  const app = await NestFactory.create(AppModule);

  const serverConfig = config.get('server');

  const port = serverConfig.port;
  await app.listen(port);
  logger.log(`Application running on port ${port}`)
}
bootstrap();
```

설정과 환경 변수 코드에 적용하기

설정 파일에 넣어준 값들을 실제 코드에 적용해 주고 환경 변수도 정의해서 소스 코드 안에 넣어주겠습니다.

typeorm.config.ts

```
import { TypeOrmModuleOptions } from "@nestjs/typeorm";
import * as config from 'config';

const dbConfig = config.get('db');

export const typeORMConfig : TypeOrmModuleOptions = {
    type: dbConfig.type,
    host: process.env.RDS_HOSTNAME || dbConfig.host,
    port: process.env.RDS_PORT || dbConfig.port,
    username: process.env.RDS_USERNAME || dbConfig.username,
    password: process.env.RDS_PASSWORD || dbConfig.password,
    database: process.env.RDS_DB_NAME || dbConfig.database,
    //Entities to be loaded for this connection
    entities: [__dirname + '/../**/*.entity.{js,ts}'],
    // Indicates if database schema should be auto created on
    // Be careful with this option and don't use this in produ
    // - otherwise you can lose production data.
    // This option is useful during debug and development.
    synchronize: dbConfig.synchronize
}
```

auth.module.ts

```
import * as config from 'config';

const jwtConfig = config.get('jwt');

@Module({
  imports: [
    PassportModule.register({ defaultStrategy: 'jwt' }),
    JwtModule.register({
      secret: jwtConfig.secret,
      signOptions: {
        expiresIn: process.env.JWR_SECRET || jwtConfig.expiresIn,
      }
```

```
    }),
    TypeOrmModule.forFeature([UserRepository])
  ],
```

jwt.strategy.ts

```
import * as config from 'config';
export class JwtStrategy extends PassportStrategy(Strategy) {
    constructor(
        @InjectRepository(UserRepository)
        private userRepository: UserRepository,
    ) {
        super({
            jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
            secretOrKey: process.env.JWT_SECRET || config.get('jwt.secret')
        });
    }
```