

웹서버보안프로그래밍

5. 쇼핑몰 예제 프로젝트

중부대학교 정보보호학과

이병천 교수

sultan@joongbu.ac.kr

전체 목차

1. 강의 개요
2. 웹개발환경 구축
3. Next.js 소개
4. MongoDB CRUD
- 5. 쇼핑몰 예제 프로젝트**
6. PBL 팀프로젝트 발표

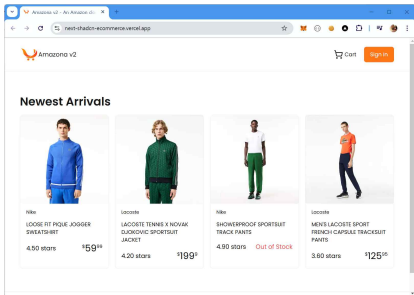
4. 쇼핑물 예제 프로젝트

1. 프로젝트 소개



1. 프로젝트 소개

- 쇼핑몰 예제 프로젝트
 - 동영상: <https://www.youtube.com/watch?v=M4DrCi8EuYE>
 - 소스코드: <https://github.com/basir/next-pg-shadcn-ecommerce>
 - Lessons 폴더에 상세 가이드 안내문 제공



일부 구현 코드

소스코드: <https://github.com/lbcsultan/next-shadcn-ecommerce>

웹서비스: <https://next-shadcn-ecommerce.vercel.app/>

사용 기술

- 이 프로젝트에서 사용하는 기술들
 - Next.js: 14.2.3 (or 15)
 - Tailwind CSS
 - Shadcn UI (컴포넌트 기반의 CSS)
 - PostgreSQL (vercel에서 제공하는 데이터베이스)
 - Next-auth: 5.0 beta
 - Bcrypt: 비밀번호 해시
 - Drizzle ORM: DB 관리자
 - Zod: 스키마 검증 라이브러리
 - Stripe, PayPal: 온라인 결제
 - 등....

```
"dependencies": {  
  "@auth/drizzle-adapter": "^1.5.0",  
  "@radix-ui/react-dropdown-menu": "^2.1.1",  
  "@radix-ui/react-label": "^2.1.0",  
  "@radix-ui/react-slot": "^1.1.0",  
  "@radix-ui/react-toast": "^1.2.1",  
  "@vercel/postgres": "^0.10.0",  
  "bcrypt-ts-edge": "^3.0.1",  
  "class-variance-authority": "^0.7.0",  
  "clsx": "^2.1.1",  
  "drizzle-orm": "^0.33.0",  
  "lucide-react": "^0.441.0",  
  "next": "14.2.12",  
  "next-auth": "5.0.0-beta.21",  
  "react": "^18",  
  "react-dom": "^18",  
  "tailwind-merge": "^2.5.2",  
  "tailwindcss-animate": "^1.0.7",  
  "zod": "^3.23.8"  
},
```

2. next.js 프로젝트 생성

- 프로젝트 폴더 생성
 - 예) daiso-shopping
- VSCode에서 위 프로젝트 폴더 열기
- Shadcn/ui를 이용하는 새로운 next.js 프로젝트 생성
 - <https://ui.shadcn.com/docs/installation/next> 참조

Shadcn/ui

- Shadcn/ui란?
 - Tailwind CSS 기반의 컴포넌트 라이브러리
 - 각종 컴포넌트 사용 가능: Button, Card, Badge, Chart, Input, Menu 등
 - Tailwind CSS에 컴포넌트 개념이 없다는 단점을 해결
 - 사용자가 디자인을 쉽게 수정할 수 있는 컴포넌트
 - <https://ui.shadcn.com/>
 - Next.js 프로젝트에서 사용하기
<https://ui.shadcn.com/docs/installation/next>

Gatsby

Manual

ComponentsSidebar **New**

Accordion

Alert

Alert Dialog

Aspect Ratio

Avatar

Badge

Breadcrumb

Button

Calendar

Card

Carousel

Chart

Checkbox

Collapsible

Combobox

Command

Context Menu

Data Table

Date Picker

Dialog

새로운 프로젝트 생성

- 프로젝트 폴더 생성
 - 예) daiso-shopping
- VSCode에서 위 프로젝트 폴더 열기
- Shadcn/ui를 이용하는 새로운 next.js 프로젝트 생성 (추천)
 - > **npx shadcn@latest init**

```
D:\WebDev\2024-2\nextjs\daiso-shopping>npx shadcn@latest init
✓ The path D:\WebDev\2024-2\nextjs\daiso-shopping does not contain
✓ What is your project named? ... .
✓ Creating a new Next.js project.
✓ Which style would you like to use? » Default
✓ Which color would you like to use as the base color? » Slate
✓ Would you like to use CSS variables for theming? ... no / yes
✓ Writing components.json.
✓ Checking registry.
✓ Updating tailwind.config.ts
✓ Updating app\globals.css
✓ Installing dependencies.
✓ Created 1 file:
  - lib\utils.ts

Success! Project initialization completed.
You may now add components.
```


새로운 프로젝트 생성

- 설치된 내용 확인하기
 - Package.json (이 프로젝트에 설치된 외부 패키지들)
 - Components.json (shadcn/ui의 환경 구성)
 - Tailwind.config.ts (각종 css 변수들 설정)
 - lib/utlis.ts (cn 함수 선언)

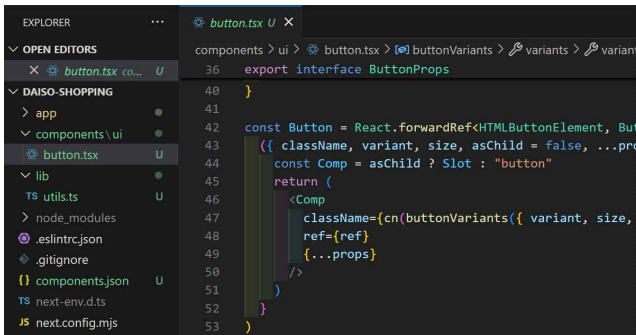
호환성 문제로 이전 버전으로 설치됨

- Next: 14.2.16
- React: 18

```
{
  "name": "daiso-shopping",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
  },
  "dependencies": {
    "class-variance-authority": "^0.7.0",
    "clsx": "^2.1.1",
    "lucide-react": "^0.461.0",
    "next": "14.2.16",
    "react": "^18",
    "react-dom": "^18",
    "tailwind-merge": "^2.5.5",
    "tailwindcss-animate": "^1.0.7"
  },
  "devDependencies": {
    "@types/node": "^20",
    "@types/react": "^18",
    "@types/react-dom": "^18",
    "eslint": "^8",
    "eslint-config-next": "14.2.16",
    "postcss": "^8",
    "tailwindcss": "^3.4.1",
    "typescript": "^5"
  }
}
```

버튼 컴포넌트 테스트

- Button 추가
 - > **npx shadcn@latest add button**
 - components/ui/button.tsx가 추가됨



The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays the project structure with the file `button.tsx` selected under the `components/ui` directory. The main editor area shows the content of `button.tsx`, which includes an exported `ButtonProps` interface and a `Button` component implemented using `React.forwardRef` and `cn` for class name merging.

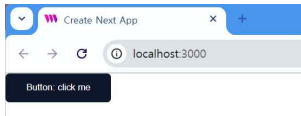
```
36 export interface ButtonProps
37 {
38   className?: string;
39   variant?: "default" | "destructive" | "outline" | "secondary" | "ghost";
40   size?: "default" | "sm" | "lg";
41 }
42 const Button = React.forwardRef<HTMLButtonElement, ButtonProps>((
43   { className, variant, size, asChild = false, ...props }, ref,
44   const Comp = asChild ? Slot : "button"
45   return (
46     <Comp
47       className={cn(buttonVariants({ variant, size,
48         ref={ref}
49       {...props}
50     />
51   )
52 }
53 )
```

버튼 컴포넌트 추가 테스트

- 버튼 사용해보기
- 메인페이지에 추가 /app/page.tsx

```
import { Button } from '@components/ui/button'
import Image from 'next/image'

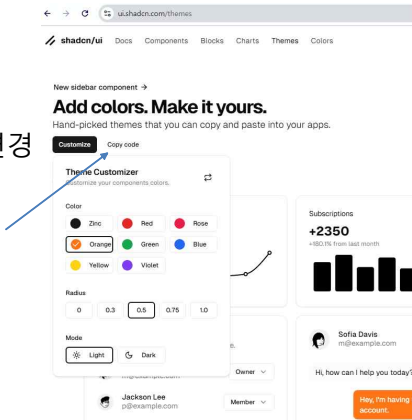
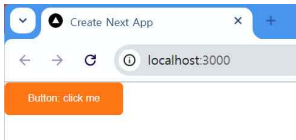
export default function Home() {
  return (
    <div>
      <Button variant={'default'} size={'lg'}>
        Button: click me
      </Button>
    </div>
  )
}
```



Button 컴포넌트를 사용. 속성 값 사용 가능
Variant: default, destructive, outline, secondary, ghost, link
Size: default, sm, lg, icon
중에서 선택 가능. 추가 가능

테마 바꾸기 가능

- <https://ui.shadcn.com/themes>
- 오렌지 테마로 변경
- Copy code 하여 globals.css 내용을 변경



Commit and push to github

- <https://github.com/lbcsultan/daiso-shopping>



Lessons

- 다음의 가이드를 따라서 진행
 - 가이드가 매우 잘 정리되어 있음
 - <https://github.com/basir/next-pg-shadcn-ecommerce/tree/main/lessons>
 - 03. create next app까지 이미 완료

next-pg-shadcn-ecommerce / lessons /

basir 37: add product quick view modal 1c61f42	
Name	Last commit message
..	
01. introduction.md	11. create new user
02. install tools.md	11. create new user
03. create next app.md	11. create new user
04. create website layout.md	04. create website layout
05. list products.md	05. list products
06. setup drizzle orm and postgres d...	06. setup drizzle orm and postgres data...
07. load products from database.md	07. load products from database
08. deploy on vercel.md	08. deploy on vercel
09. create product details page.md	09. create product details page
10. implement authentication.md	33. email order receipt by Resend - fix
11. create new user.md	11. create new user

04. Create website layout

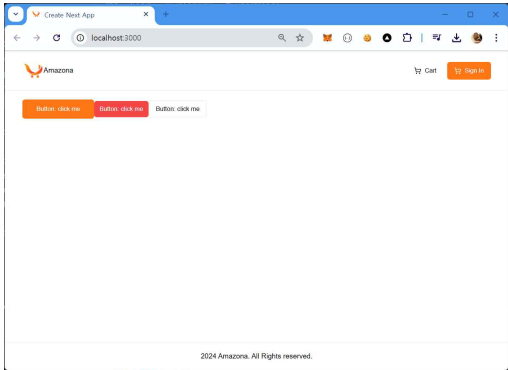
- 1. remove app/favicon.ico, public/vercel.svg and next.svg
- 2. add app/icon.svg and public/assets/icons/logo.svg
- 3. app/globals.css
- 4. lib/constants/index.ts
- 5. components/shared/header/index.tsx
- 6. components/shared/footer.tsx
- 7. app/(root)/layout.tsx
- 8. move app/page.tsx to app/(root)/page.tsx

- **Commit f6cd878 참조**

가이드를 따라 진행하고 내용을 이해하려고 노력할 것

04. Create website layout

- 완성
- Commit & push
 - Message: 04. create website layout

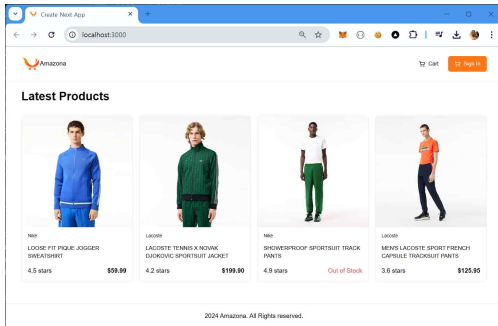


05. List products

- 1. .eslintrc.json
- 2. copy sample product images to public/assets/images folder
 - 이미지 파일 준비
- 3. lib/sample-data.ts
 - 초기화면용 샘플데이터 준비
- 4. pnpm dlx shadcn-ui@latest add card
(npx shadcn@latest add card 로 변경 적용)
- 5. components/shared/product/product-card.tsx
 - 제품정보를 카드 형식으로 표시. 위에서 추가한 Card 컴포넌트 사용
- 6. components/shared/product/product-list.tsx
 - 제품들을 그리드 형식으로 전시
- 7. app/(root)/page.tsx
 - 샘플데이터를 읽어와서 위의 ProductList 컴포넌트를 사용하여 화면에 표시.

05. List products

- 완성
- Build & deploy
 - > npm run build
 - 에러 수정 및 재전송



빌드 에러 수정

- Typescript 에러를 해결하기 위해
 - Product interface를 생성
 - any 를 Product으로 타입 지정

/types/index.ts

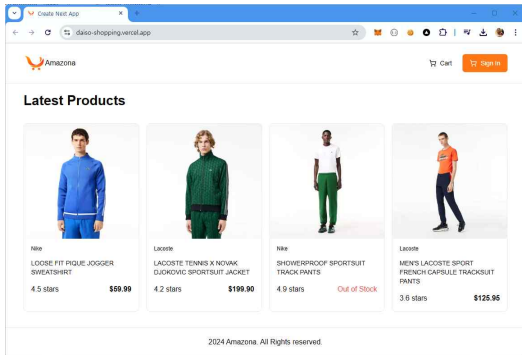
```
export interface Product {  
  name: string  
  slug: string  
  category: string  
  images: string[]  
  price: string  
  brand: string  
  rating: string  
  numReviews: number  
  stock: number  
  description: string  
  isFeatured?: boolean  
  banner?: string  
}
```

/app/(root)/page.tsx

```
import ProductList from '@components/shared/product/product-list'  
import sampleData from '@lib/sample-data'  
import { Product } from '@types'  
  
export default function Home() {  
  const validatedProducts: Product[] = sampleData.products.map((product) => ({  
    ...product,  
    isFeatured: product.isFeatured ?? false,  
  }))  
  
  return (  
    <div className="space-y-8">  
      <h2 className="h2-bold">Latest Products</h2>  
      <ProductList data={validatedProducts} />  
    </div>  
  )  
}
```

빌드 성공 후 push & deploy

- Commit & push
 - Message: 05. list products
- Vercel deploy
 - <https://daiso-shopping.vercel.app/>



06. setup drizzle orm and postgres database

- Vercel.com에서 database 생성하기
 - <https://vercel.com/docs/storage/vercel-postgres>
 - <https://vercel.com/lbcsultans-projects/~stores>
 - Create database > Postgres

Create Postgres Database



You have reached the limit of Postgres Databases on this plan.

Personal Accounts are limited to 1 active Postgres Database at a time. To create additional Postgres Databases, upgrade your plan to Pro or remove an existing Postgres Database.

[Learn more](#)

Go Back

Upgrade your plan

Browse Storage

Create databases and stores that you can connect to your projects.



Edge Config

Ultra-low latency reads



Blob Beta

Fast object storage



Postgres

Powered by  Neon



KV and Postgres are now available through the Marketplace.
[Learn more.](#)

Marketplace Database Providers [Learn more](#)



Neon Coming Soon

Serverless Postgres

[Learn more](#)




Upstash

Serverless Data Platform

Cancel

Continue

Vercel.com에서 database 생성하기

 nextjs-dashboard-postgres

Created

1y ago

[← All Databases](#)

Postgres Database

Beta

Status	Region	Endpoint	Storage Size	Compute Time	Data Transfer	Written Data
Available	Washington, D.C., USA	iad1 ep-square-firefly-07101495-poo...	30 MB/256 MB	0.18 hrs/60 hrs	74 kB	106 kB

Quickstart

```
psql  @vercel/postgres-kysely  drizzle-orm/vercel-postgres  @vercel/postgres  Prisma  pg  .env.local  Show secret  Copy Snippet
```

```
1 POSTGRES_URL="*****"
2 POSTGRES_PRISMA_URL="*****"
3 POSTGRES_URL_NO_SSL="*****"
4 POSTGRES_URL_NON_POOLING="*****"
5 POSTGRES_USER="*****"
6 POSTGRES_HOST="*****"
7 POSTGRES_PASSWORD="*****"
8 POSTGRES_DATABASE="*****"
```

.env.local에 붙여넣기

Vercel.com에서 database 생성하기

```
$ .env.local
1 POSTGRES_URL="postgres://default:711fxZsupne
aws.neon.tech:5432/verceldb?sslmode=require"
2 POSTGRES_PRISMA_URL="postgres://default:711f
us-east-1.aws.neon.tech:5432/verceldb?sslmod
3 POSTGRES_URL_NO_SSL="postgres://default:711f
us-east-1.aws.neon.tech:5432/verceldb"
4 POSTGRES_URL_NON_POOLING="postgres://default
us-east-1.aws.neon.tech:5432/verceldb?sslmod
5 POSTGRES_USER="default"
6 POSTGRES_HOST="ep-square-firefly-07101495-po
7 POSTGRES_PASSWORD="711fxZsupney"
8 POSTGRES_DATABASE="verceldb"
```

첫번째만 남기고 나머지는 지움
APP_NAME, APP_DESCRIPTION 추가

```
$ .env.local
1 NEXT_PUBLIC_APP_NAME=Amazona
2 NEXT_PUBLIC_APP_DESCRIPTION=An Amazon clone built w
3
4 POSTGRES_URL="postgres://default:711fxZsupney@ep-sq
aws.neon.tech:5432/verceldb?sslmode=require"
5
```

.env.local

ORM(Object-Relational Mapping)

- Prisma vs. Drizzle vs. Mongoose -

Prisma, Drizzle, Mongoose는 각각 고유한 특징을 가진 데이터베이스 ORM(Object-Relational Mapping) 도구입니다. 이들의 주요 차이점은 다음과 같습니다:

지원 데이터베이스

- Prisma: SQL 데이터베이스(PostgreSQL, MySQL, SQLite 등)와 MongoDB를 지원합니다 ③.
- Drizzle: SQL 데이터베이스만 지원하며, MongoDB는 지원하지 않습니다 ①.
- Mongoose: MongoDB 전용 ORM입니다 ②.

타입 안전성

- Prisma: 완전한 타입 안전성을 제공합니다 ③.
- Drizzle: 쿼리 결과에 대해서만 타입 정보를 제공합니다 ③.
- Mongoose: JavaScript 기반으로, 타입 안전성이 상대적으로 낮습니다.

쿼리 작성 방식

- Prisma: 직관적이고 추상화된 API를 제공합니다 ② ③.
- Drizzle: SQL과 유사한 문법을 사용합니다 ① ③.
- Mongoose: MongoDB 쿼리 문법과 유사한 방식을 사용합니다 ②.

성능

- Prisma: 최근 버전에서 성능 개선이 이루어졌습니다 ①.
- Drizzle: 경량화되어 있어 서버리스 환경에서 빠른 성능을 보입니다 ①.
- Mongoose: MongoDB에 최적화되어 있습니다.

스키마 정의

- Prisma: Prisma 스키마 언어를 사용합니다 ③.
- Drizzle: TypeScript 함수를 사용하여 테이블을 정의합니다 ③.
- Mongoose: JavaScript 객체를 사용하여 스키마를 정의합니다 ②.

사용 편의성

- Prisma: 높은 수준의 추상화로 사용이 쉽습니다 ③.
- Drizzle: SQL에 익숙한 개발자에게 친숙합니다 ①.
- Mongoose: MongoDB 사용자에게 직관적입니다 ②.

각 ORM은 프로젝트의 요구사항, 개발 팀의 경험, 사용하는 데이터베이스 등에 따라 선택할 수 있습니다.

Drizzle ORM 설치

- Drizzle ORM
 - <https://orm.drizzle.team/>
- 설치
 - <https://orm.drizzle.team/docs/get-started-postgresql#v>
 - <https://orm.drizzle.team/docs/get-started/vercel-new>
 - > npm i drizzle-orm @vercel/postgres dotenv
 - > npm i -D drizzle-kit tsx

Step 1 - Install required package

```
npm yarn pnpm bun  
  
npm i drizzle-orm @vercel/postgres dotenv  
npm i -D drizzle-kit tsx
```

Step 2 - Setup connection variables

Create a `.env` file in the root of your project and add your database connection variable:

```
POSTGRES_URL=
```

WARNING

It's important to name the variable `POSTGRES_URL` for Vercel Postgres.

In the Vercel Postgres storage tab, you can find the `.env.local` tab and copy the `POSTGRES_URL` variable

Step 3 - Connect Drizzle ORM to the database

Create a `index.ts` file in the `src/db` directory and initialize the connection:

Schema 생성

- /db/schema.ts 생성
 - 상품 product에 대한 명세 작성

/lib/sample-data.ts

```
products: [
  {
    name: 'LOOSE FIT PIQUE JOGGER SWEATSHIRT',
    slug: 'loose-fit-pique-jogger-sweatshirt',
    category: 'Men's Sweatshirts',
    images: ['/assets/images/p1-1.jpeg', '/assets/images/p1-2.jpeg'],
    price: '59.99',
    brand: 'Nike',
    rating: '4.5',
    numReviews: 10,
    stock: 5,
    description:
      'Lacoste sporting elegance with an urban twist. Fall in love with this loose, cozy sweatshirt in super-comfortable double-face piqué.',
    isFeatured: true,
    banner: '/assets/images/banner-1.jpeg',
  },
],
```

/db/schema.ts

```
import {
  boolean, integer, numeric, pgTable, text, timestamp, uniqueIndex, uuid,
} from 'drizzle-orm/pg-core'

// PRODUCTS
export const products = pgTable(
  'product',
  {
    id: uuid('id').defaultRandom().primaryKey().notNull(),
    name: text('name').notNull(),
    slug: text('slug').notNull(),
    category: text('category').notNull(),
    images: text('images').array().notNull(),
    brand: text('brand').notNull(),
    description: text('description').notNull(),
    stock: integer('stock').notNull(),
    price: numeric('price', { precision: 12, scale: 2 }).notNull().default('0'),
    rating: numeric('rating', { precision: 3, scale: 2 })
      .notNull()
      .default('0'),
    numReviews: integer('numReviews').notNull().default(0),
    isFeatured: boolean('isFeatured').default(false).notNull(),
    banner: text('banner'),
    createdAt: timestamp('createdAt').defaultNow().notNull(),
  },
  (table) => {
    return {
      productSlugIdx: uniqueIndex('product_slug_idx').on(table.slug),
    }
  }
)
```

Drizzle 환경 설정

- 패키지 설치
 - > npm i -D pg @types/pg @next/env
- /src/drizzle.config.ts 생성
 - schema.ts 지정

/src/drizzle.config.ts

```
import { cwd } from 'node:process'
import { loadEnvConfig } from '@next/env'

loadEnvConfig(cwd())

import { defineConfig } from 'drizzle-kit'
export default defineConfig({
  dialect: 'postgresql',
  schema: './db/schema.ts',
  out: './drizzle',
  dbCredentials: {
    url: process.env.POSTGRES_URL!,
  },
})
```

Schema를 vercel database에 적용

- 생성된 Schema를 vercel database로 push
 - > npx drizzle-kit push

```
D:\WebDev\2024-2\nextjs\daiso-shopping>npx drizzle-kit push
No config path provided, using default 'drizzle.config.ts'
Reading config file 'D:\WebDev\2024-2\nextjs\daiso-shopping\drizzle.config.ts'
Using 'pg' driver for database querying
[✓] Pulling schema from database...
Warning Found data-loss statements:
· You're about to delete user table with 4 items
· You're about to delete cart table with 3 items

THIS ACTION WILL CAUSE DATA LOSS AND CANNOT BE REVERTED

Do you still want to push changes?
[✓] Changes applied

D:\WebDev\2024-2\nextjs\daiso-shopping>
```

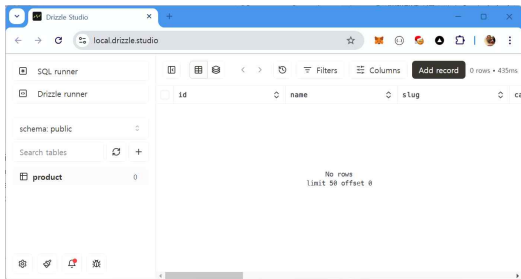
Drizzle-kit studio 실행

- Drizzle-kit studio 실행
 - > npx drizzle-kit studio
 - <https://local.drizzle.studio>

```
D:\WebDev\2024-2\nextjs\daiso-shopping>npx drizzle-kit studio
No config path provided, using default 'drizzle.config.ts'
Reading config file 'D:\WebDev\2024-2\nextjs\daiso-shopping\drizzle.config.ts'
Using 'pg' driver for database querying

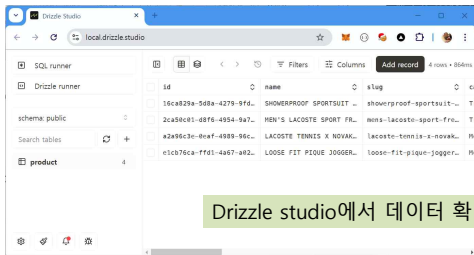
Warning Drizzle Studio is currently in Beta. If you find anything that is not working as expected or should be improved, feel free to create an issue on GitHub: https://github.com/drizzle-team/drizzle-kit-mirror/issues/new or write to us on Discord: https://discord.gg/WcRKz2FFxN

Drizzle Studio is up and running on https://local.drizzle.studio
```



샘플 데이터를 DB에 저장하기

- /db/seed.ts 생성
- > npx tsx ./db/seed 실행



Drizzle studio에서 데이터 확인

```
import { cwd } from 'node:process'
import { loadEnvConfig } from '@next/env'
```

/db/seed.ts

```
import { drizzle } from 'drizzle-orm/node-postgres'
import { Client } from 'pg'
```

```
import * as schema from './schema'
import sampleData from '@lib/sample-data'
```

```
loadEnvConfig(cwd())
```

```
const main = async () => {
  try {
    const client = new Client({
      connectionString: process.env.POSTGRES_URL,
    })
    await client.connect()
    const db = drizzle(client)
```

```
    await db.delete(schema.products)
```

```
    const resProducts = await db
      .insert(schema.products)
      .values(sampleData.products)
      .returning()
```

```
    console.log({ resProducts })
    await client.end()
  } catch (error) {
    console.error(error)
    throw new Error('Failed to seed database')
  }
}
```

```
main()
```

07. load products from database

- 1. db/drizzle.ts 생성
 - Drizzle을 통해 database에 접근 준비
- 2. /types/index.ts 생성
 - Schema로부터 Product type 생성

/db/drizzle.ts

```
import * as schema from './schema'

import { drizzle } from 'drizzle-orm/vercel-postgres'
import { sql } from '@vercel/postgres'
const db = drizzle(sql, {
  schema,
})
export default db
```

/types/index.ts

```
import { products } from '@db/schema'
import { InferSelectModel } from 'drizzle-orm'

// PRODUCTS
export type Product = InferSelectModel<typeof products>
```

DB로부터 데이터 읽어오기

- 3. /lib/actions/product.actions.ts 생성
 - Server action을 이용해 DB로부터 products를 가져옴
 - 생성시간 기준으로 정렬하고 데이터를 4개만 가져옴

```
'use server' /lib/actions/product.actions.ts

import { desc } from 'drizzle-orm'

import db from '@db/drizzle'
import { products } from '@db/schema'

export async function getLatestProducts() {
  const data = await db.query.products.findMany({
    orderBy: [desc(products.createdAt)],
    limit: 4,
  })
  return data
}
```

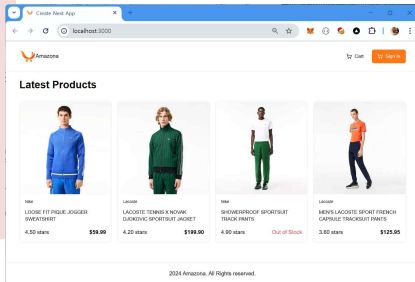

초기 화면에 표시

- 초기 화면 수정

/app/(root)/page.tsx

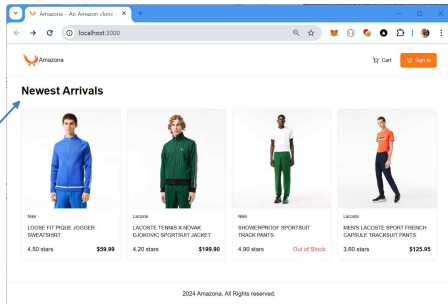
```
import ProductList from '@components/shared/product/product-list'
import { getLatestProducts } from '@lib/actions/product.actions'
```

```
export default async function Home() {
  const latestProducts = await getLatestProducts()
  return (
    <div className="space-y-8">
      <h2 className="h2-bold">Latest Products</h2>
      <ProductList data={latestProducts} />
    </div>
  )
}
```



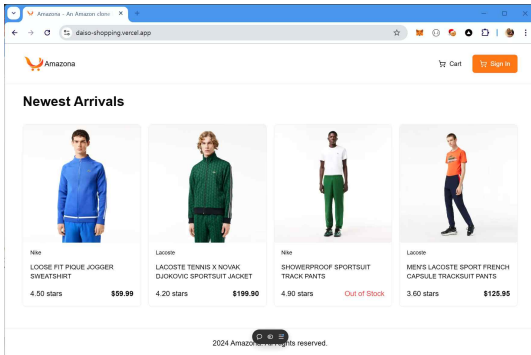
초기 화면 수정

- Title 변수 사용
 - /app/(root)/page.tsx 수정
 - /components/shared/product/product-list.tsx 수정
- Product type 사용
 - /components/shared/product/product-card.tsx
 - /components/shared/product/product-list.tsx



08. deploy on vercel

- 빌드 테스트
 - > npm run build
- Github
 - Push to github
- Vercel
 - import project from github
 - add env variables
 - deploy it (redeploy)



09. create product details page

- 1. /lib/actions/product.actions.ts 에 getProductBySlug() 함수 추가
 - slug값이 같은 product를 읽어옴
- 2. shadcn의 badge 추가
 - > npx shadcn@latest add badge

'use server'

/lib/actions/product.actions.ts

import { desc, eq } from 'drizzle-orm'

import db from '@db/drizzle'

import { products } from '@db/schema'

```
export async function getLatestProducts() {  
  const data = await db.query.products.findMany({  
    orderBy: [desc(products.createdAt)],  
    limit: 4,  
  })  
  return data  
}
```

```
export async function getProductBySlug(slug: string) {  
  return await db.query.products.findFirst({  
    where: eq(products.slug, slug),  
  })  
}
```

컴포넌트 생성, 수정

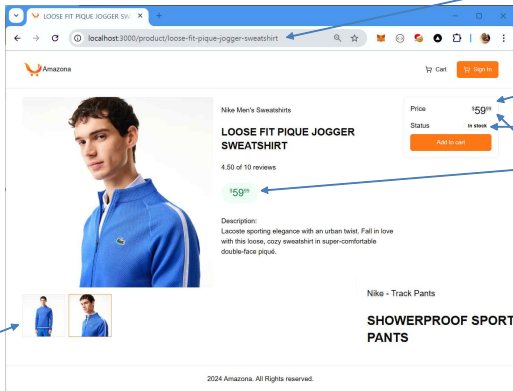
- 3. /components/shared/product/product-images.tsx 생성
 - 썸네일을 표시하도록 수정
- 4. /components/shared/product/product-price.tsx 생성
 - 가격을 소수점 기준으로 다르게 표시
- 5. /components/shared/product/product-card.tsx 수정
 - Price를 위의 ProductPrice 컴포넌트를 이용하여 표시



Product details 페이지 생성

- 6. /app/(root)/product/[slug]/page.tsx 생성

Slug로 dynamic routing



Card

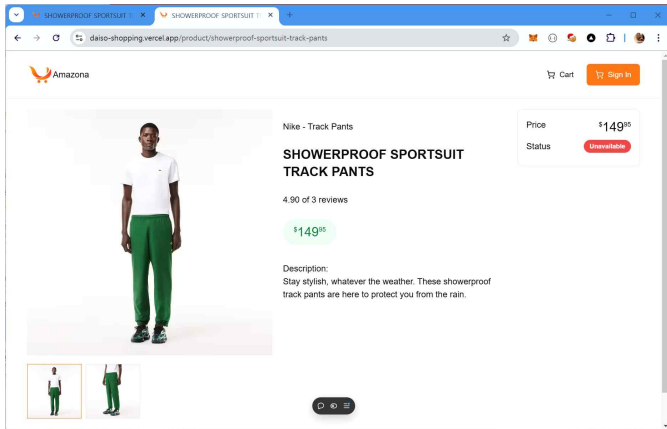
Badge

price 표시

Badge

썸네일

Deploy



10. implement authentication

- 1. install nextauth
 - <https://authjs.dev/getting-started/installation?framework=Next.js> 참조
 - > npm install next-auth@beta
 - > npx auth secret
 - /auth.ts 생성
 - /app/api/auth/[...nextauth]/route.ts 생성
 - /middleware.ts 생성
- 2. use drizzle adapter
- 3. /db/schema.ts

User 스키마, 테이블 생성

- 이메일, 패스워드를 이용한 사용자 등록, 인증 예정
 - Credentials provider 사용
- Drizzle ORM adapter
 - <https://authjs.dev/getting-started/adapters/drizzle> 참조
 - 패키지 설치
 - npm install drizzle-orm @auth/drizzle-adapter
 - npm install drizzle-kit --save-dev
 - Schemas > PostgreSQL 에서 users, accounts, sessions, verificationTokens 부분 추가
 - text('id') 대신 uuid('id'), uuid('userId')로 수정

```
import {
  boolean, integer, numeric, pgTable, primaryKey, text,
  timestamp, uniqueIndex, uuid,
} from 'drizzle-orm/pg-core'
import { AdapterAccountType } from 'next-auth/adapters'
```

```
// USERS
```

```
export const users = pgTable('user', {
  id: uuid('id').defaultRandom().primaryKey().notNull(),
  name: text('name'),
  email: text('email').unique(),
  emailVerified: timestamp('emailVerified', { mode: 'date' }),
  image: text('image'),
})

export const accounts = pgTable(
  'account',
  {
    userId: uuid('userId')
      .notNull()
      .references(() => users.id, { onDelete: 'cascade' }),
    type: text('type').$type<AdapterAccountType>().notNull(),
    provider: text('provider').notNull(),
    providerAccountId: text('providerAccountId').notNull(),
    refresh_token: text('refresh_token'),
    access_token: text('access_token'),
    expires_at: integer('expires_at'),
    token_type: text('token_type'),
    scope: text('scope'),
    id_token: text('id_token'),
    session_state: text('session_state'),
  },
  (account) => ({
    compoundKey: primaryKey({
      columns: [account.provider, account.providerAccountId],
    }),
  })
)
```

/db/schema.ts

```
export const sessions = pgTable('session', {
  sessionToken: text('sessionToken').primaryKey(),
  userId: uuid('userId')
    .notNull()
    .references(() => users.id, { onDelete: 'cascade' }),
  expires: timestamp('expires', { mode: 'date' }).notNull(),
})

export const verificationTokens = pgTable(
  'verificationToken',
  {
    identifier: text('identifier').notNull(),
    token: text('token').notNull(),
    expires: timestamp('expires', { mode: 'date' }).notNull(),
  },
  (verificationToken) => ({
    compositePk: primaryKey({
      columns: [verificationToken.identifier,
        verificationToken.token],
    }),
  })
)
```

```
// PRODUCTS
```

```
export const products = pgTable(
  'product',
  {
    id: uuid('id').defaultRandom().primaryKey().notNull(),
    name: text('name').notNull(),
    slug: text('slug').notNull(),
    category: text('category').notNull(),
    images: text('images').array().notNull(),
    brand: text('brand').notNull(),
    description: text('description').notNull(),
    stock: integer('stock').notNull(),
    price: numeric('price', { precision: 12, scale: 2 }).notNull().default('0'),
    rating: numeric('rating', { precision: 3, scale: 2 })
      .notNull()
      .default('0'),
    numReviews: integer('numReviews').notNull().default(0),
    isFeatured: boolean('isFeatured').default(false).notNull(),
    banner: text('banner'),
    createdAt: timestamp('createdAt').defaultNow().notNull(),
  },
  (table) => {
    return {
      productSlugIdx: uniqueIndex('product_slug_idx').on(table.slug),
    }
  }
)
```

- 4. add credentials provider
 - > npm install bcrypt-ts-edge (패스워드 해시)
 - auth.ts 수정
 - /db/schemas.ts 에서 users 수정 (role, password 추가)

```
export const users = pgTable('user', {  
  id: uuid('id').defaultRandom().primaryKey().notNull(),  
  name: text('name'),  
  email: text('email').unique(),  
  role: text('role').notNull().default('user'),  
  password: text('password'),  
  emailVerified: timestamp('emailVerified', { mode: 'date' }),  
  image: text('image'),  
})
```

```
import { DrizzleAdapter } from '@auth/drizzle-adapter'
import { compareSync } from 'bcrypt-ts-edge'
import { eq } from 'drizzle-orm'
import type { NextAuthConfig } from 'next-auth'
import NextAuth from 'next-auth'
import CredentialsProvider from 'next-auth/providers/credentials'
```

```
import db from './db/drizzle'
import { users } from './db/schema'
```

```
export const config = {
  pages: {
    signIn: '/sign-in',
    error: '/sign-in',
  },
  session: {
    strategy: 'jwt',
    maxAge: 30 * 24 * 60 * 60,
  },
  adapter: DrizzleAdapter(db),
  providers: [
    CredentialsProvider({
      //
    }
  )
],
  callbacks: {
    session: async ({ session, user, trigger, token }: any) => {
      session.user.id = token.sub
      if (trigger === 'update') {
        session.user.name = user.name
      }
      return session
    },
  },
} satisfies NextAuthConfig
export const { handlers, auth, signIn, signOut } = NextAuth(config)
```

상세 내용

```
providers: [
  CredentialsProvider({
    credentials: {
      email: {
        type: 'email',
      },
      password: { type: 'password' },
    },
    async authorize(credentials) {
      if (credentials == null) return null

      const user = await db.query.users.findFirst({
        where: eq(users.email, credentials.email as string),
      })
      if (user && user.password) {
        const isMatch = compareSync(
          credentials.password as string,
          user.password
        )
        if (isMatch) {
          return {
            id: user.id,
            name: user.name,
            email: user.email,
            role: user.role,
          }
        }
      }
      return null
    },
  })
],
```

• 7. middleware.ts 수정

```
export { auth as middleware } from '@auth'
export const config = {
  matcher: ['/(?!api|_next/static|_next/image|_next/assets|favicon.ico).*'],
}
```

이 config는 미들웨어가 적용될 URL 경로를 지정하는 matcher를 설정합니다 ① ③. 구체적으로:

1. 정규 표현식 ``/(?!api|_next/static|_next/image|_next/assets|favicon.ico).*``을 사용합니다.
2. 이 정규 표현식은 다음 경로를 제외한 모든 요청 경로에 미들웨어를 적용합니다 ③ ④:
 - ``/api``로 시작하는 경로 (API 라우트)
 - ``/_next/static``으로 시작하는 경로 (정적 파일)
 - ``/_next/image``로 시작하는 경로 (이미지 최적화 파일)
 - ``/_next/assets``로 시작하는 경로 (애셋 파일)
 - ``/favicon.ico`` (파비콘 파일)
3. 이렇게 설정함으로써, 위에 나열된 경로를 제외한 모든 요청에 대해 미들웨어가 실행됩니다 ⑥.

이 설정은 주로 정적 파일이나 API 라우트에는 미들웨어를 적용하지 않고, 일반적인 페이지 요청에만 미들웨어를 적용하고자 할 때 사용됩니다 ③ ④.

• 8. /types/next-auth.d.ts 생성

```
import { DefaultSession } from 'next-auth'
```

```
declare module 'next-auth' {  
  export interface Session {  
    user: {  
      role: string  
    } & DefaultSession['user']  
  }  
}
```

이 코드는 Next.js에서 사용되는 NextAuth 라이브러리의 Session 타입을 확장하는 TypeScript 선언입니다. 구체적으로:

1. `DefaultSession`에서 제공하는 기본 사용자 정보에 `role` 필드를 추가합니다.
2. `declare module 'next-auth'`를 사용하여 기존 `next-auth` 모듈의 타입 정의를 확장합니다.
3. `Session` 인터페이스를 재정의하여 `user` 객체에 `role` 속성을 추가합니다.
4. `& DefaultSession['user']`를 통해 기존 `DefaultSession`의 `user` 속성을 모두 포함하면서 `role`을 추가합니다.

이렇게 타입을 확장함으로써, NextAuth의 세션 객체에서 `user.role`에 접근할 수 있게 됩니다. 이는 사용자의 역할 기반 접근 제어(RBAC) 등을 구현할 때 유용합니다 5 6.

- 9. / lib/constants/index.ts 수정

```
export const APP_NAME = process.env.NEXT_PUBLIC_APP_NAME || 'Amazona'  
export const APP_DESCRIPTION =  
  process.env.NEXT_PUBLIC_APP_DESCRIPTION ||  
  'An Amazon clone built with Next.js, Postgres, Shadcn'  
  
export const signInDefaultValues = {  
  email: '',  
  password: '',  
}
```

- 10. /lib/sample-data.ts 수정
 - Default users 추가
- 11. shadcn에서 label input dropdown-menu 추가
 - > npx shadcn@latest add label input dropdown-menu
- 12. zod 추가
 - TypeScript를 위한 강력한 스키마 선언 및 유효성 검사 라이브러리
 - > npm install zod

/lib/sample-data.ts

```
import { hashSync } from 'bcrypt-ts-edge'
```

```
const sampleData = {  
  users: [  
    {  
      name: 'John',  
      email: 'admin@example.com',  
      password: hashSync('123456', 10),  
      role: 'admin',  
    },  
    {  
      name: 'Jane',  
      email: 'jane@example.com',  
      password: hashSync('123456', 10),  
      role: 'user',  
    },  
  ],  
  products: [  

```

계속...

- 13. /lib/validator.ts 생성

```
import * as z from 'zod'
```

```
// USER
```

```
export const signInFormSchema = z.object({  
  email: z.string().min(3, 'Email must be at least 3 characters'),  
  password: z.string().min(3, 'Password must be at least 3 characters'),  
})
```

- 14. /lib/actions/user.actions.ts 생성
- 15. /components/shared/header/user-button.tsx 생성
- 16. /components/shared/header/menu.tsx 생성

- 20. /app/(auth)/layout.tsx 생성 (lesson 파일에 예러)
- 21. /app/(auth)/sign-in/credentials-signin-form.tsx 생성
- 22. /app/(auth)/sign-in/page.tsx 생성

- 23. /db/seed.ts 생성
- 24. user 데이터를 DB에 저장
 - > npx drizzle-kit push (DB에 users table 생성)
 - > npx tsx ./db/seed (데이터 입력)

```
import { cwd } from 'node:process'
import { loadEnvConfig } from '@next/env'
import { drizzle } from 'drizzle-orm/node-postgres'
import { Client } from 'pg'
import * as schema from './schema'
import sampleData from '@lib/sample-data'
loadEnvConfig(cwd())
```

```
const main = async () => {
  try {
    const client = new Client({
      connectionString: process.env.POSTGRES_URL,
    })
    await client.connect()
    const db = drizzle(client)
    await db.delete(schema.accounts)
    await db.delete(schema.users)
    await db.delete(schema.products)
    const resUsers = await db
    .insert(schema.users)
    .values(sampleData.users)
    .returning()
    const resProducts = await db
      .insert(schema.products)
      .values(sampleData.products)
      .returning()
    console.log({ resUsers, resProducts })
    await client.end()
  } catch (error) {
    console.error(error)
    throw new Error('Failed to seed database')
  }
}
main()
```

Drizzle Studio

local.drizzle studio

SQL runner

Drizzle runner

schema: public

Search tables

- account 0
- product 4
- session 0
- user 2
- verificationToken 0

2 rows • 255ms

id	name	email	role
f3510ce7-3fa4-40c3-941...	Jane	jane@example.com	user
f677542d-b7cf-4c61-b49...	John	admin@example.com	admin

user, account, session, verificationToken 테이블이 생성됨
user 테이블에 2개의 사용자 정보가 등록됨
account, session, verificationToken 테이블은 비어 있음

- 25. 메뉴 수정
 - /components/shared/header/index.tsx

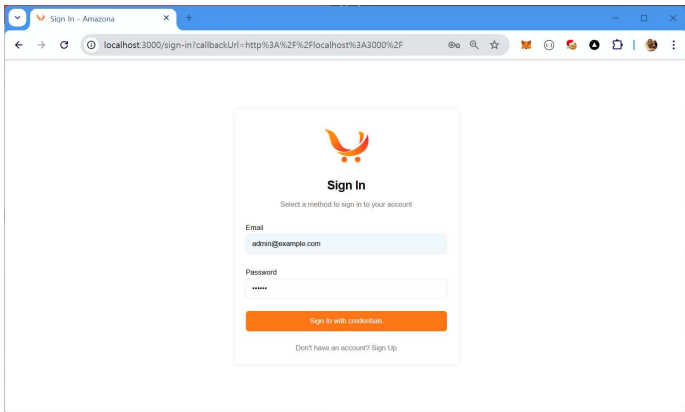
```
import Image from 'next/image'
import Link from 'next/link'


import { APP_NAME } from '@lib/constants'
import Menu from './menu'

const Header = async () => {
  return (
    <header className="w-full border-b">
      <div className="wrapper flex-between">
        <div className="flex-start">
          <Link href="/" className="flex-start">
            <Image
              src="/assets/icons/logo.svg"
              width={48}
              height={48}
              alt={` ${APP_NAME} logo`}
            />
            {APP_NAME}
          </Link>
        </div>
        <Menu />
      </div>
    </header>
  )
}

export default Header
```

로그인 테스트



 Cart John

John
admin@example.com
Sign Out

로그인 후 UserButton 나타남

Deploy

- 빌드 테스트
- 추가 환경변수 입력
- Redeploy

