



Bejjangi

# 베짱이 마켓

21101160 강민재, 21101199 양효정



—○—

## CONTENTS

개요 및 목표

---

프로젝트 진행 절차

---

프로젝트 설계

---

프로젝트 수행결과

---

설계 요소 평가

---

추후 프로젝트 발전 방향

—○—

# 01 개요 및 목표

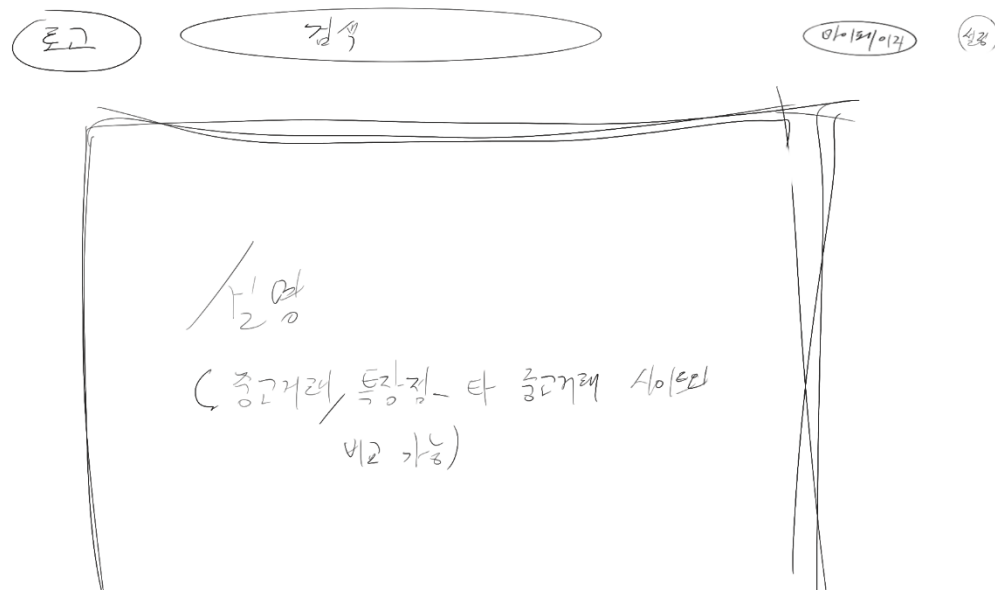
## 개발 목표

- 회원가입과 로그인 기능을 통해 유저를 관리한다.
- 검색어를 입력 받아 여러 중고 장터에 올라와 있는 검색어 관련 물품을 한 화면에서 볼 수 있도록 한다.
- 검색하여 나온 물품 리스트들은 장바구니에 넣을 수 있으며, 각 유저들은 장바구니에 들어가 자신이 저장해둔 물건들을 확인할 수 있다
- 각 회원들은 자신의 프로필 사진과 닉네임, 이메일 주소를 관리할 수 있다.
- 유저 정보와 장바구니 리스트는 데이터 베이스에서 저장한다.

## 02 진행 방법 및 절차

- 한 절차가 끝날때마다 깃허브에 업로드,  
다음 파트 역할 분담 후 작업 진행
- 프론트엔드와 백엔드를 동시에 진행할 때  
대면으로 진행해서 서로 파트 봐주기식 진행(pair programming)

# 03 Project 설계



메인 페이지

# 03 Project 설계

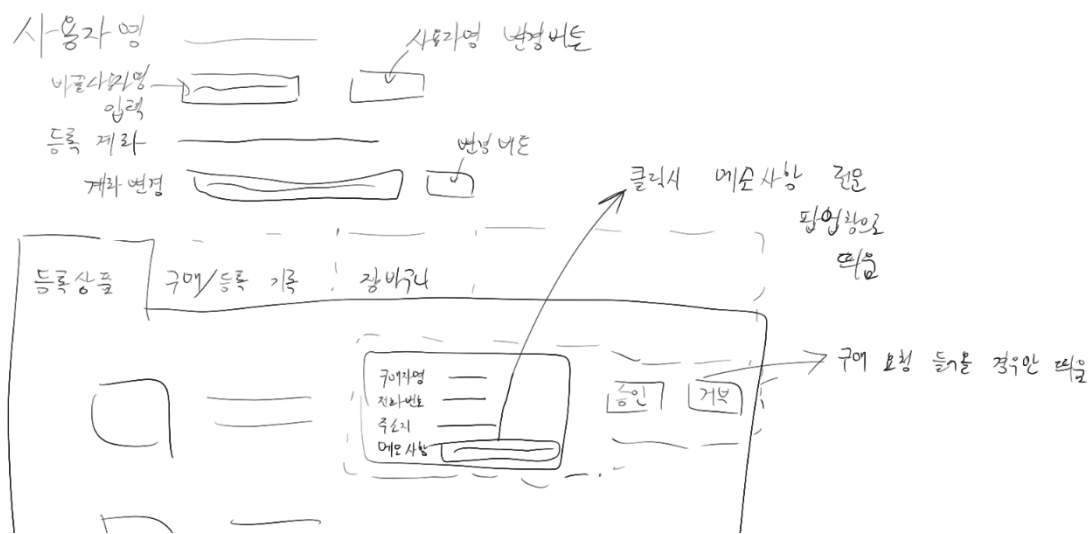
로그

검색

마이페이지

설정

my page



마이 페이지

# 03 Project 설계

로그      검색      마이페이지      설정

등록된 계좌번호

입금명 : ~~~

구매자명

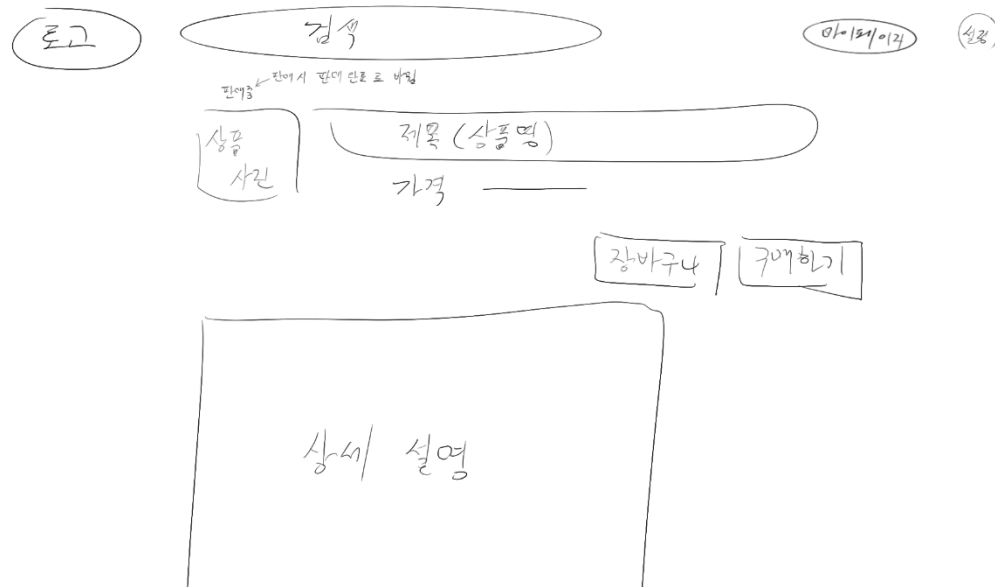
비밀지

전화번호

메모 사항

물품 구매 페이지

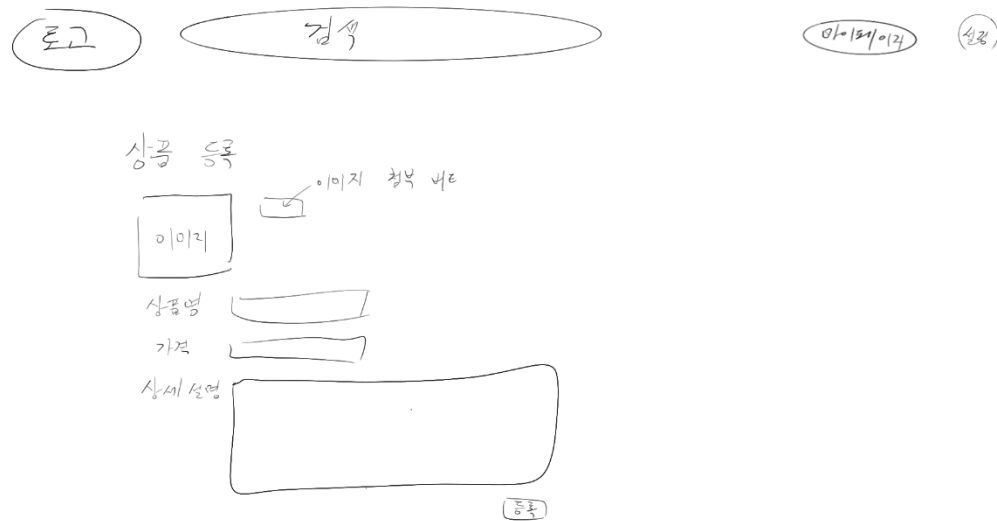
# 03 Project 설계



물품 상세 페이지

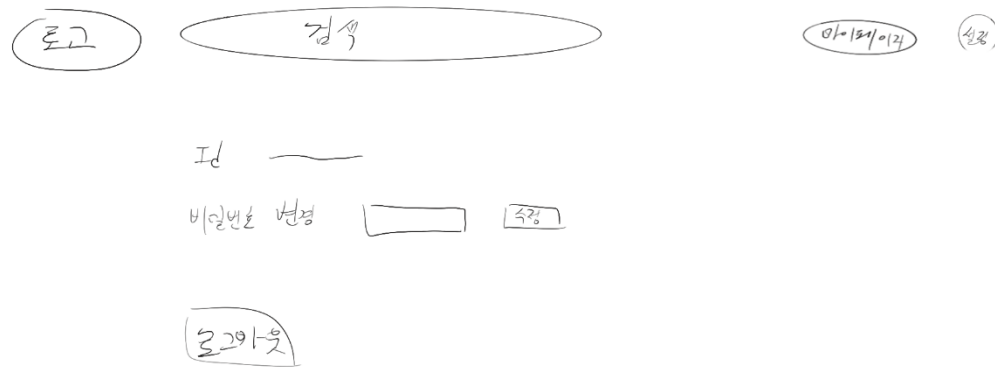


# 03 Project 설계



상품 등록 페이지

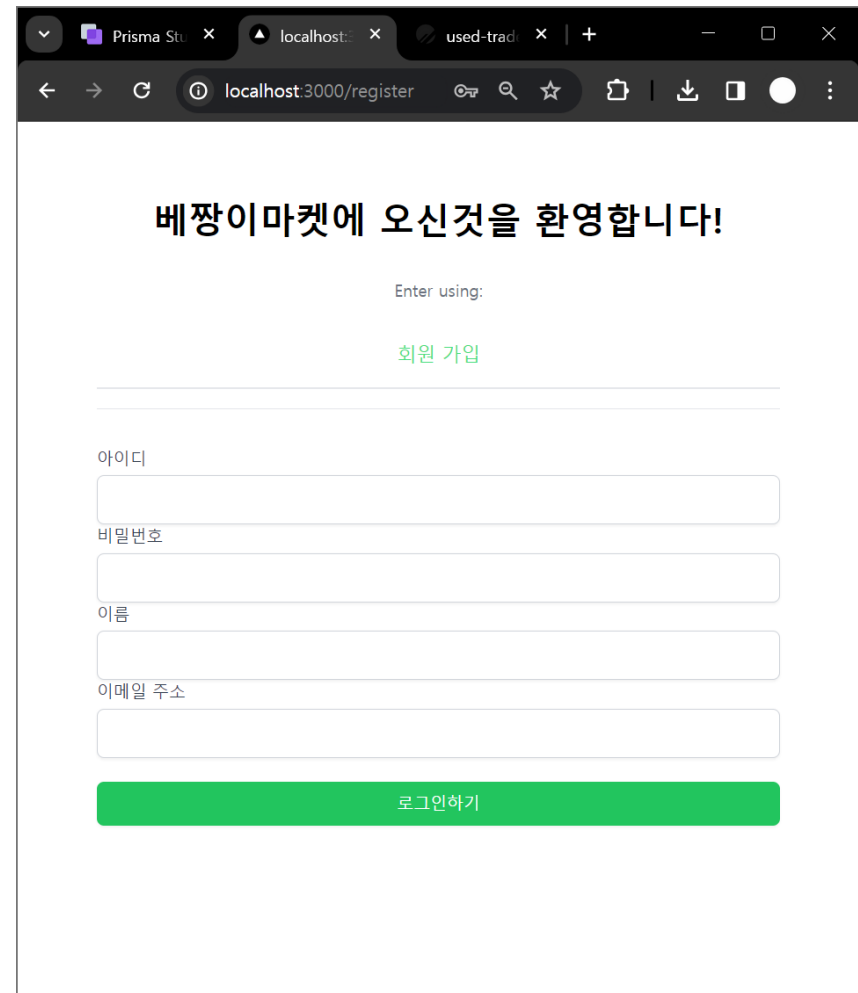
# 03 Project 설계



설정 페이지

## 04 Project 수행 결과

<http://localhost:3000/register> 접속시 유저 회원가입 페이지



베짱이마켓에 오신것을 환영합니다!

Enter using:

[회원 가입](#)

아이디

비밀번호

이름

이메일 주소

로그인하기

## 04 Project 수행 결과

src/pages/register.tsx: 유저 회원가입 페이지

src/pages/api/users/register.tsx: 유저 회원가입 페이지에서 데이터를 주고 받는 파일

## 04 Project 수행 결과

src/pages/register.tsx: 유저 회원가입 페이지

```
✓ interface RegisterForm {  
  name: string  
  user_id: string  
  email: string  
  password: string  
}  
  
✓ interface MutationResult {  
  ok: boolean  
  error?: string  
}
```

## 04 Project 수행 결과

src/pages/register.tsx: 유저 회원가입 페이지

```
const onValid = async (validForm: RegisterForm) => {
  if (loading) return

  try {
    const response = await enter(validForm) //api/users/register로 데이터 전송후 확인받기

    if (response.ok) {
      //유일한 id, 이메일이고 계정 잘 생성됨
      alert(validForm.name + '님, 회원가입이 완료되었습니다.')
      router.push('/enter')
    } else {
      // 로그인 실패
      console.error('회원가입 실패:', response.error) // 콘솔에 에러 출력
      setErrorMessage('이미 있는 아이디 혹은 이메일 주소입니다.')
    }
  } catch (error) {
    console.error('기타 에러 발생:', error)
  }
}
```

## 04 Project 수행 결과

src/pages/api/users/register.tsx: 유저 회원가입 페이지에서 데이터를 주고 받는 파일

```
export default async function handler(  
  req: NextApiRequest,  
  res: NextApiResponse,  
) {  
  const { user_id, password, name, email } = req.body  
  
  if (!user_id || !password || !name || !password) {  
    return res  
      .status(400)  
      .json({ ok: false, error: '모든 정보를 빠짐없이 적어주세요.' })  
  }  
  
  try {  
    // 이미 아이디가 존재하는지 확인  
    const isAccountExists = await checkIdExistence(user_id, email)  
  
    if (isAccountExists) {  
      return res  
        .status(400)  
        .json({ ok: false, error: '이미 존재하는 아이디 혹은 이메일입니다.' })  
    }  
  }  
}
```

## 04 Project 수행 결과

libs/utils.ts: 유저 id 혹은 이메일이 이미 존재하는지 확인하기

```
export async function checkIdExistence(user_id: string, email: string) {  
  try {  
    const userId = await prisma.user.findUnique({  
      where: {  
        user_id,  
      },  
    })  
    const userEmail = await prisma.user.findUnique({  
      where: {  
        email,  
      },  
    })  
    return !!userId || !!userEmail //아이디 이미 존재하는지 확인  
  } catch (error) {  
    console.error('util쪽에서 에러 발생:', error)  
    throw new Error('util쪽에서 에러 발생')  
  }  
}
```



## 04 Project 수행 결과

src/pages/api/users/register.tsx: 유저 회원가입 페이지에서 데이터를 주고 받는 파일

```
// 아이디&이메일이 존재하지 않으면 회원가입 진행
const user = await createUser(user_id, password, name, email)

if (user) {
  return res.json({ ok: true })
} else {
  return res
    .status(500)
    .json({ ok: false, error: '회원가입 중 오류가 발생했습니다.' })
}
} catch (error) {
  console.error('회원가입 중 에러 발생:', error)
  return res
    .status(500)
    .json({ ok: false, error: '서버 오류가 발생했습니다.' })
}
}
```

## 04 Project 수행 결과

libs/utiles.ts: 유저 회원가입 페이지에서 데이터를 주고 받는 파일

```
💡 새로운 사용자 생성
export async function createUser(
  user_id: string,
  password: string,
  name: string,
  email: string,
): Promise<any> {
  try {
    const user = await prisma.user.create({
      data: {
        user_id,
        password,
        name,
        email,
      },
    })
    return user
  } catch (error) {
    console.error('회원가입 중 에러 발생:', error)
  }
}
```

# 04 Project 수행 결과

schema.prisma 파일에 데이터 베이스 형식 지정.  
콘솔에서 npx prisma db push라고 치면 planescale에 올라감

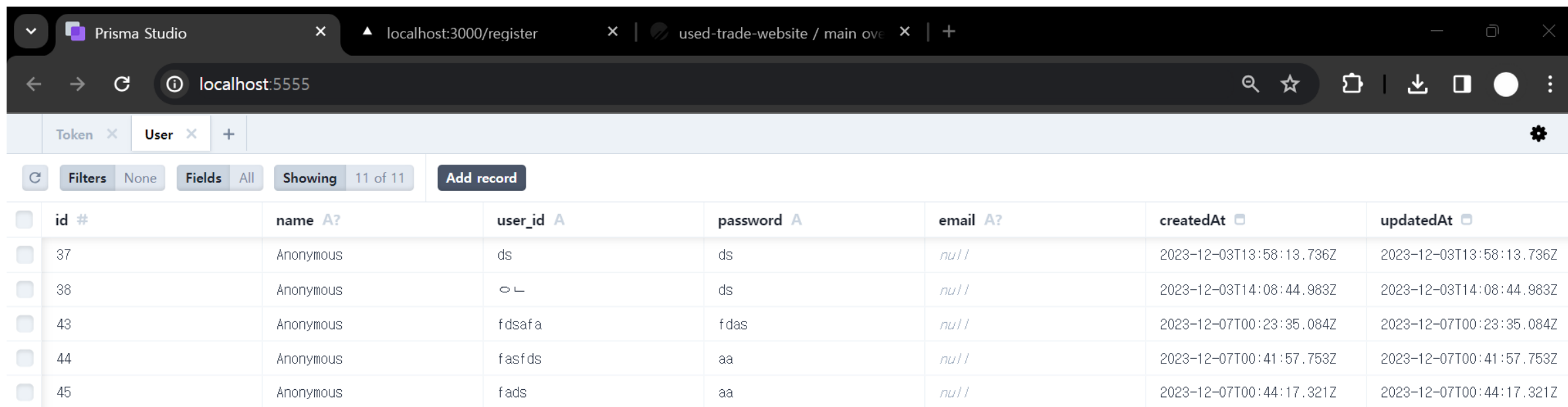
```
17 //autoincrement() : 자동으로 증가하는 숫자
18 model User {
19   id      Int      @id @default(autoincrement())
20   name    String?
21   user_id String    @unique
22   password String
23   email   String?  @unique
24   createdAt DateTime @default(now())
25   updatedAt DateTime @updatedAt
26   tokens  Token[]
27 }
```

User

```
CREATE TABLE `User` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(191),
  `user_id` varchar(191) NOT NULL,
  `password` varchar(191) NOT NULL,
  `email` varchar(191),
  `createdAt` datetime(3) NOT NULL DEFAULT current_timestamp(3),
  `updatedAt` datetime(3) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `User_user_id_key` (`user_id`),
  UNIQUE KEY `User_email_key` (`email`)
) ENGINE InnoDB,
  CHARSET utf8mb4,
  COLLATE utf8mb4_unicode_ci;
```

# 04 Project 수행 결과

콘솔에 `npx prisma studio` 입력하면 유저가 새로 생성 된 것을 확인할 수 있음.

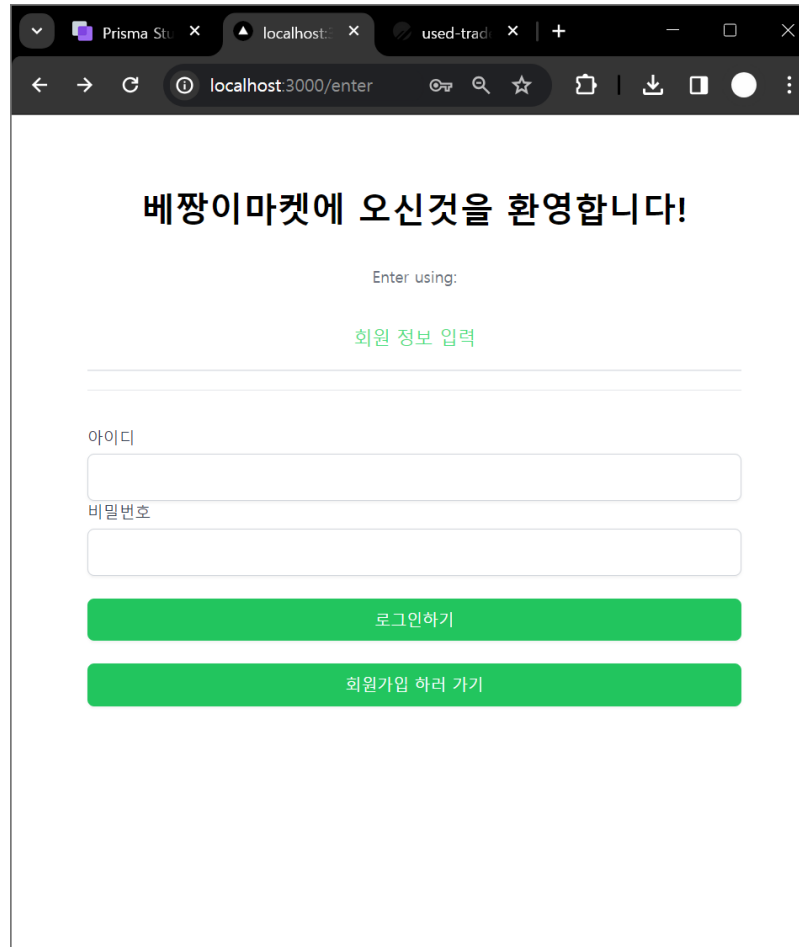


The screenshot shows the Prisma Studio web interface. The browser tabs include 'Prisma Studio', 'localhost:3000/register', and 'used-trade-website / main ove'. The address bar shows 'localhost:5555'. The interface has a top bar with 'Token' and 'User' tabs, and a settings gear icon. Below the tabs, there's a section with 'Filters' (None), 'Fields' (All), 'Showing 11 of 11', and an 'Add record' button. The main area displays a table with 8 columns: id #, name A?, user\_id A, password A, email A?, createdAt, and updatedAt. The table contains 5 rows of data, all with 'Anonymous' names and various timestamps.

<input type="checkbox"/>	id #	name A?	user_id A	password A	email A?	createdAt	updatedAt
<input type="checkbox"/>	37	Anonymous	ds	ds	null	2023-12-03T13:58:13.736Z	2023-12-03T13:58:13.736Z
<input type="checkbox"/>	38	Anonymous	oL	ds	null	2023-12-03T14:08:44.983Z	2023-12-03T14:08:44.983Z
<input type="checkbox"/>	43	Anonymous	fdsafa	fdas	null	2023-12-07T00:23:35.084Z	2023-12-07T00:23:35.084Z
<input type="checkbox"/>	44	Anonymous	fasfds	aa	null	2023-12-07T00:41:57.753Z	2023-12-07T00:41:57.753Z
<input type="checkbox"/>	45	Anonymous	fads	aa	null	2023-12-07T00:44:17.321Z	2023-12-07T00:44:17.321Z

## 04 Project 수행 결과

<http://localhost:3000/enter> 접속시 유저 로그인 페이지



베짱이마켓에 오신것을 환영합니다!

Enter using:

[회원 정보 입력](#)

아이디

비밀번호

로그인하기

회원가입 하러 가기

## 04 Project 수행 결과

src/pages/enter.tsx: 로그인 페이지

src/pages/api/enter2.tsx: 로그인 api 페이지 (로그인 정보 데이터베이스에서 확인하기)

```
interface EnterForm {  
  user_id: string  
  password: string  
}  
  
interface MutationResult {  
  ok: boolean  
  error?: string  
}  
  
interface TokenForm { //폐기  
  token: string  
}  
  
export default function Enter() {  
  const [enter, { loading, data, error }] =  
    useMutation<MutationResult>('/api/users/enter2')
```

## 04 Project 수행 결과

src/pages/enter.tsx: 로그인 페이지

src/pages/api/enter2.tsx: 로그인 api 페이지(로그인 정보 데이터베이스에서 확인하기)

```
const [submitting, setSubmitting] = useState(false)
const { register, watch, handleSubmit, reset } = useForm<EnterForm>()
```

```
const router = useRouter()

const onValid = async (validForm: EnterForm) => {
  if (loading) return

  try {
    const response = await enter(validForm) //api/users/enter2로 데이터 전송후 확인받기

    if (response.ok) {
      //아이디 비번 잘 맞춤!
      // 로그인 성공
      router.push('/')
    } else {
      // 로그인 실패
      console.error('로그인 실패:', response.error) // 콘솔에 에러 출력
      setErrorMessage('아이디나 비밀번호가 일치하지 않습니다.')
    }
  } catch (error) {
    console.error('기타 에러 발생:', error)
  }
}
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

## 04 Project 수행 결과

src/pages/api/enter2.tsx: 로그인 api 페이지(로그인 정보 데이터베이스에서 확인하기)

```
const handler_2: NextApiHandler = async (req, res) => {  
  if (req.method === 'POST') {  
    const { user_id, password } = req.body //아이디 비번 받아와서  
  
    try {  
      const isValid = await checkAccountExistence(user_id, password)  
      //utils에 있는 checkAccountExistence 함수로 아이디 비번 확인
```



## 04 Project 수행 결과

src/pages/api/enter2.tsx: 로그인 api 페이지(로그인 정보 데이터베이스에서 확인하기)

```
es > api > users > TS enter2.tsx > ...  
  
if (isAccountValid) {  
  //있는 아이디 비번이다!  
  const user = await client.user.findUnique({  
    where: { user_id: user_id },  
  }) // 실제 사용자를 아이디로 조회  
  if (user) {  
    req.session.user = { id: user.id } //쿠키 보내기 스타트  
    await req.session.save()  
  }  
  return res.json({ ok: true })  
} else {  
  //없는 아이디 비번이다!  
  return res  
    .status(401)  
    .json({ ok: false, error: '아이디나 비밀번호가 일치하지 않습니다.' })  
}
```

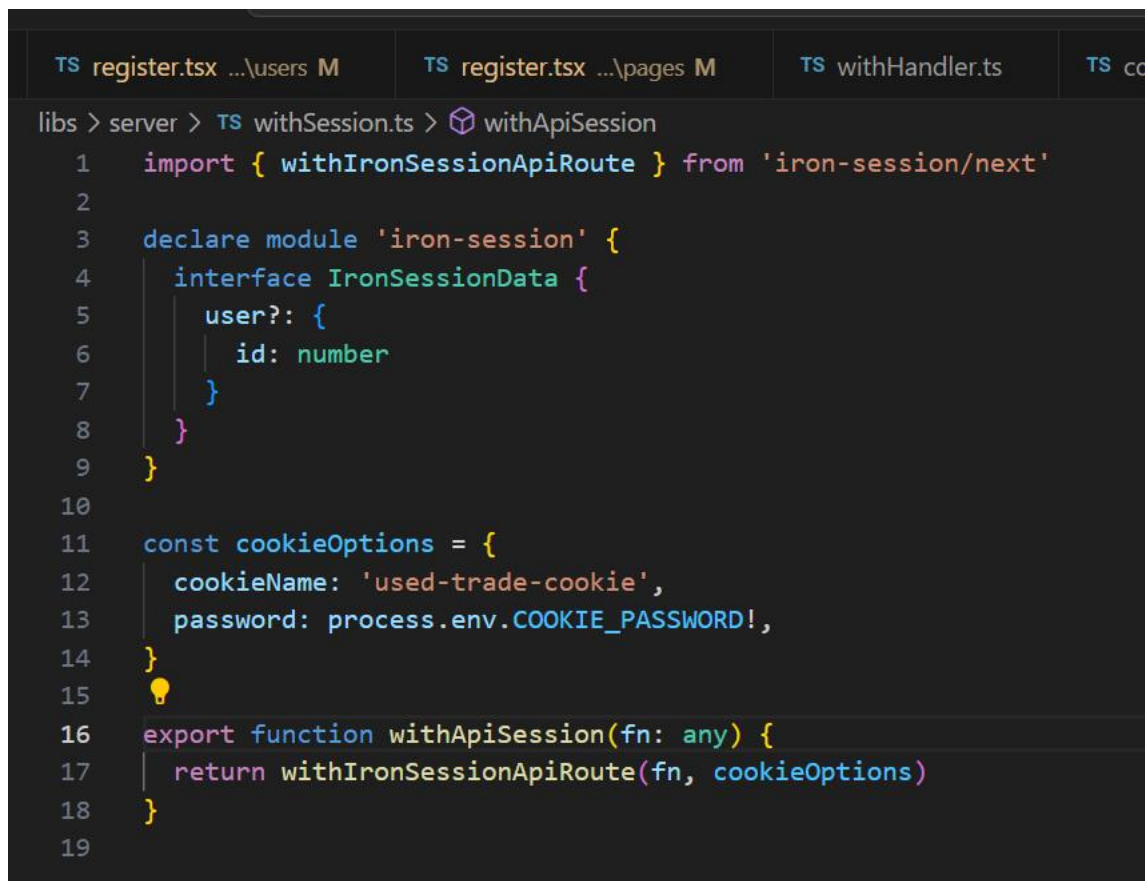
## 04 Project 수행 결과

src/pages/api/enter2.tsx: 로그인 api 페이지 (로그인 정보 데이터베이스에서 확인하기)

```
export default withApiSession(  
  withHandler(  
    method: 'POST',  
    handler: handler_2,  
    isPrivate: false,  
  )),  
)
```

## 04 Project 수행 결과

libs/server/withSession.ts



```
TS register.tsx ...\users M    TS register.tsx ...\pages M    TS withHandler.ts    TS co
libs > server > TS withSession.ts > withApiSession
1  import { withIronSessionApiRoute } from 'iron-session/next'
2
3  declare module 'iron-session' {
4    interface IronSessionData {
5      user?: {
6        id: number
7      }
8    }
9  }
10
11  const cookieOptions = {
12    cookieName: 'used-trade-cookie',
13    password: process.env.COOKIE_PASSWORD!,
14  }
15
16  export function withApiSession(fn: any) {
17    return withIronSessionApiRoute(fn, cookieOptions)
18  }
19
```

## 04 Project 수행 결과

libs/server/withSession.ts: iron-session으로 암호화된 쿠키 생성

```
TS register.tsx ...\users M  TS register.tsx ...\pages M  TS withHandler.ts  TS co
libs > server > TS withSession.ts > withApiSession
1  import { withIronSessionApiRoute } from 'iron-session/next'
2
3  declare module 'iron-session' {
4    interface IronSessionData {
5      user?: {
6        id: number
7      }
8    }
9  }
10
11  const cookieOptions = {
12    cookieName: 'used-trade-cookie',
13    password: process.env.COOKIE_PASSWORD!,
14  }
15
16  export function withApiSession(fn: any) {
17    return withIronSessionApiRoute(fn, cookieOptions)
18  }
19
```

## 04 Project 수행 결과

src/pages/api/users/me.tsx: 유저 id로 전체 데이터를 가져옴.

```
async function handler(  
  req: NextApiRequest,  
  res: NextApiResponse<ResponseType>,  
) {  
  console.log(req.session.user)  
  const profile = await client.user.findUnique({  
    where: { id: req.session.user?.id },  
  })  
  res.json({  
    ok: true,  
    profile,  
  })  
  res.status(200).end()  
}
```

## 04 Project 수행 결과

## 로그인 직후의 쿠키 확인하기

DevTools is now available in Korean! [Always match Chrome's language](#) [Switch DevTools to Korean](#) [Don't show again](#)

Elements Console Sources Network Performance Memory **Application** Security Lighthouse Recorder Performance insights

**Application**

- Manifest
- Service workers
- Storage

**Storage**

- Local storage
- Session storage
- IndexedDB
- Web SQL
- Cookies
- http://localhost:3000
- Private state tokens

Filter

☒ Only show cookies with an issue

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Partitioned	Priority
used-trade-cookie	Fe26.2*1*6e4ae...	localhost	/	2023-12-30T12:26:21.469Z	2...	✓	✓	Lax		Medium

# 04 Project 수행 결과

## 홈 화면 진입 모습

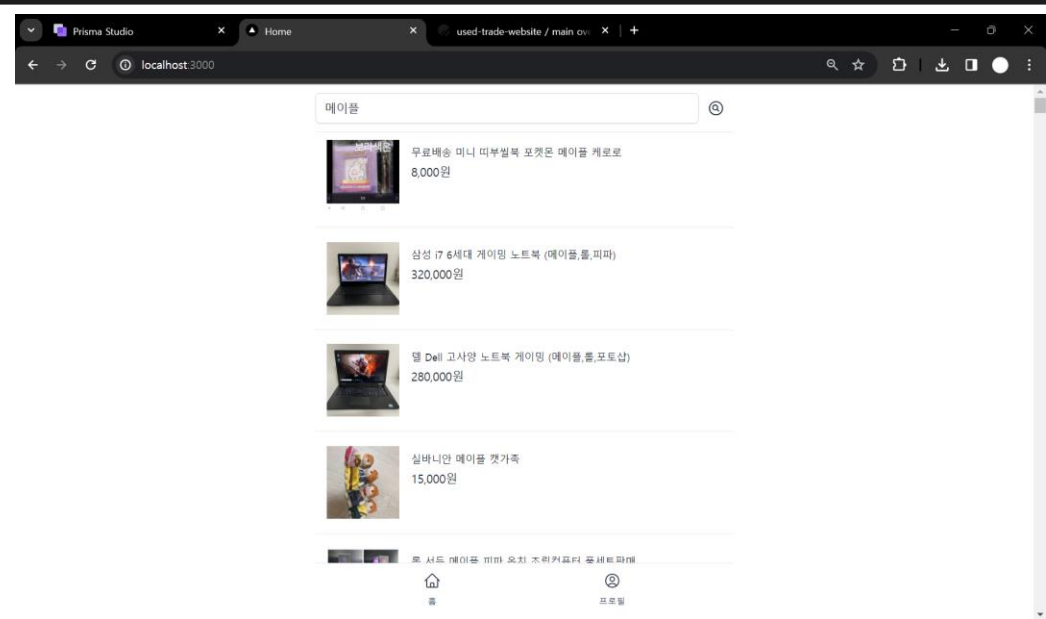
- 상단에 검색창이 위치한다.

\*유저는 검색창에 자신이 찾는 물건을 입력하면 그에 맞는 중고 물품들이 출력된다.

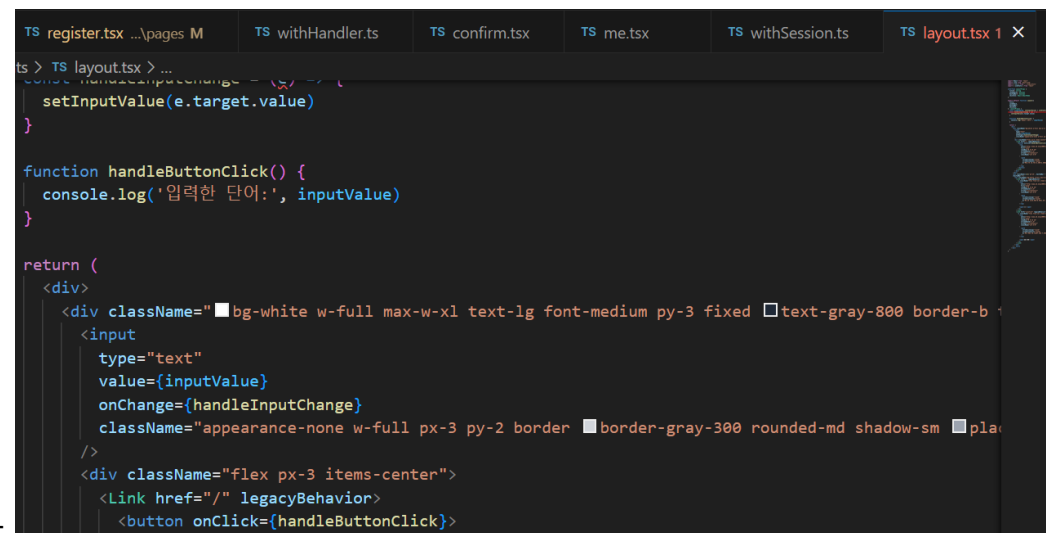
- 물품 리스트는 일렬로 스크롤 형태로 배치되며, 각 물품은 사진, 가격, 제목 정보를 보여준다.

\*이때, 제목을 클릭하면 해당 물품이 있는 url로 이동한다.

- 하단에는 홈과 프로필을 오고 갈 수 있는 배너가 존재한다.



검색어를 받아 변수 값을 설정하는 코드



# 04 Project 수행 결과

## 홈 화면 내부 동작

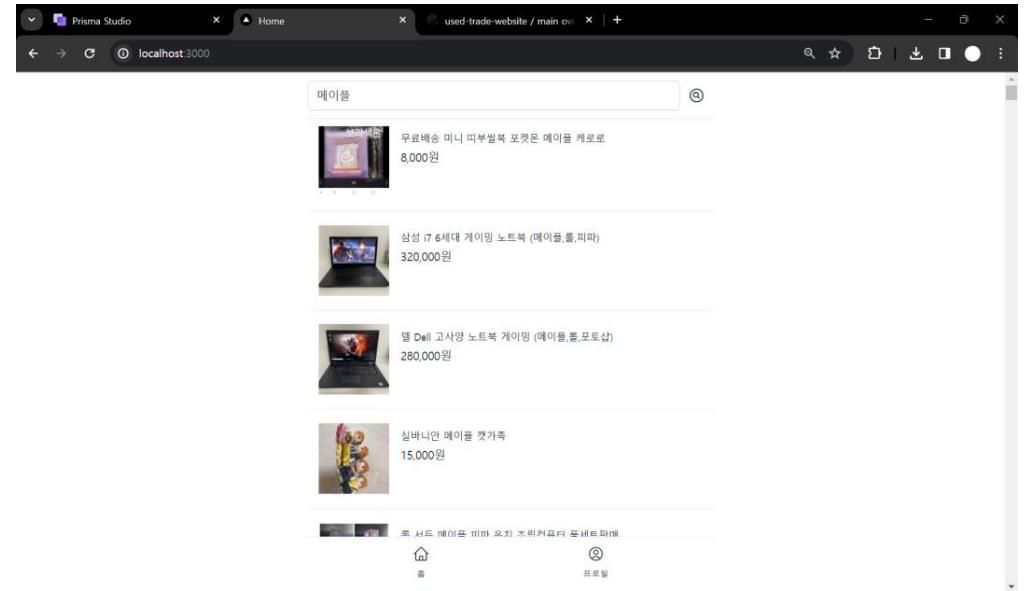
1> 유저가 검색어를 입력하면, 각 사이트를 크롤링 하는 API로 이동

2> API내 url 변수값을 수정하여 찾고자 하는 사이트에 접근해 데이터를 크롤링 한다. //test.tsx, test2.tsx

3> 긁어온 데이터에서 물품 리스트에 필요한 정보를 필터링 한 후 각 정보 변수 값에 저장한다.

\* 이 정보 변수 값들은 각 사이트의 물품 리스트에 저장된다.

4> 각 중고 사이트의 물품 리스트는 홈 화면에 순차적으로 출력된다.



```
const [data, setData] = useState<Product_Nara[] | null>(null)
const [data2, setData2] = useState<Product_Thunder[] | null>(null)

useEffect(() => {
  axios.get('/api/users/test').then((response) => {
    setData(response.data)
  })
}, [])

useEffect(() => {
  fetch('/api/users/test2')
    .then((res) => res.json())
    .then((data2: Product_Thunder[]) => {
      //console.log(data) // 콘솔에 데이터를 출력
      setData2(data2)
    })
}, [])
```

```
<Layout hasTabBar>
  <Head>
    <title>Home</title>
  </Head>
  <div className="flex flex-col space-y-5 divide-y">
    {data.map((item: Product_Nara, index: number) => (
      <div
        key={index}
        className="flex px-4 pt-5 cursor-pointer justify-between"
      >
        <div className="flex space-x-4">
          <img
            src={item.productImage}
            alt=""
            style={{ width: '100px', height: '100px' }}
          />
          <div className="pt-2 flex flex-col">
            <a
              href={item.productLink}
            >
```

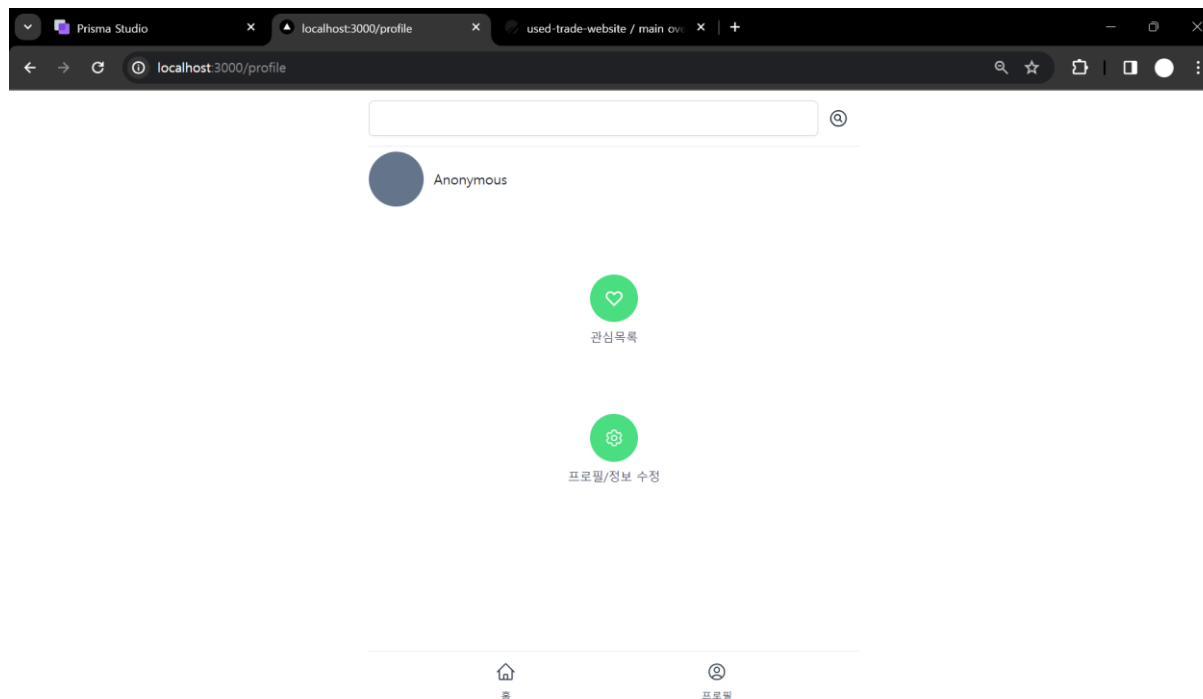
크롤링 값 가져와서 필터링, 이후 map 하여 item과 index 분리



## 04 Project 수행 결과

### 프로필 화면 진입 모습 // profile/index.tsx

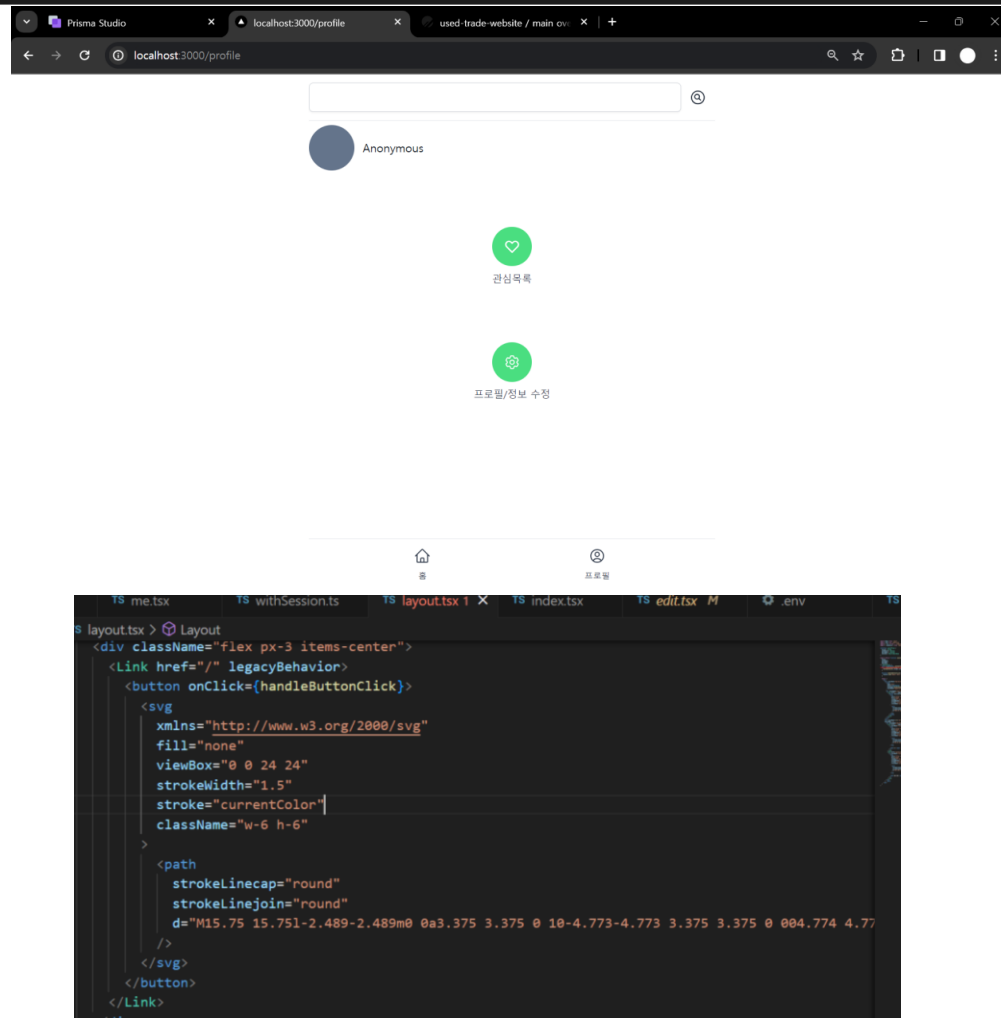
- 상단에는 검색창이 존재한다. //layout.tsx  
\*유저가 검색창에 검색어를 넣고 버튼을 누르면, 홈 화면으로 강제로 이동하며 검색의 결과를 보여준다
- 프로필 요소로는 프로필 사진(왼쪽 상단 원형), 닉네임이 있다.
- 유저는 프로필 화면에서 지금까지 추가한 관심목록의 리스트를 볼 수 있다.
- 유저는 프로필 화면에서 프로필의 정보를 수정하는 설정 창으로 이동할 수 있다.



## 04 Project 수행 결과

## 프로필 화면 내부 동작

- 유저가 검색 버튼을 누르면 강제로 홈 화면으로 이동시킨다. `//layout.tsx`
- 프로필 정보를 가지고 있는 변수들을 이용해 사진과 닉네임을 화면에 표시한다
- 프로필 내부에는 두가지 버튼이 있으며, 각각 장바구니와 설정창으로 이동시키는 역할을 한다
- 하단에는 배너가 지속해서 출력된다. `//layout.tsx`

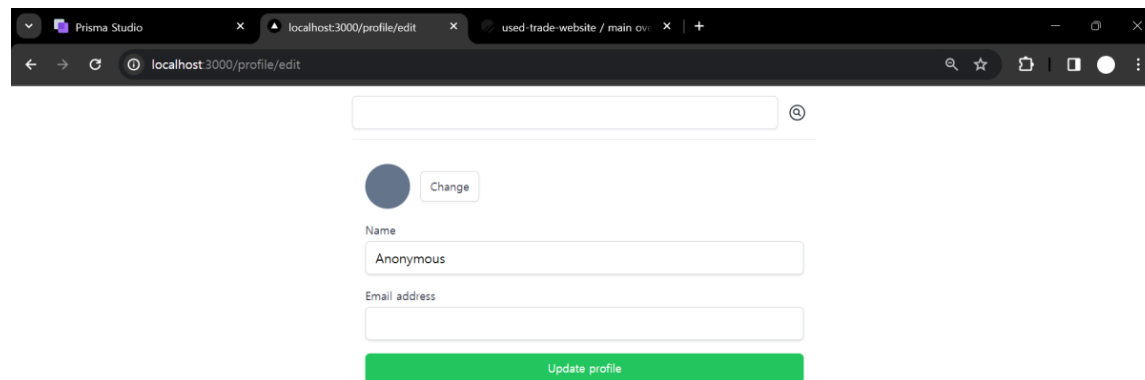


버튼을 실행 시 홈으로 이동하는 link로  
감아 작동 시 홈으로 돌아가도록 제작

# 04 Project 수행 결과

## 프로필 수정 진입 모습

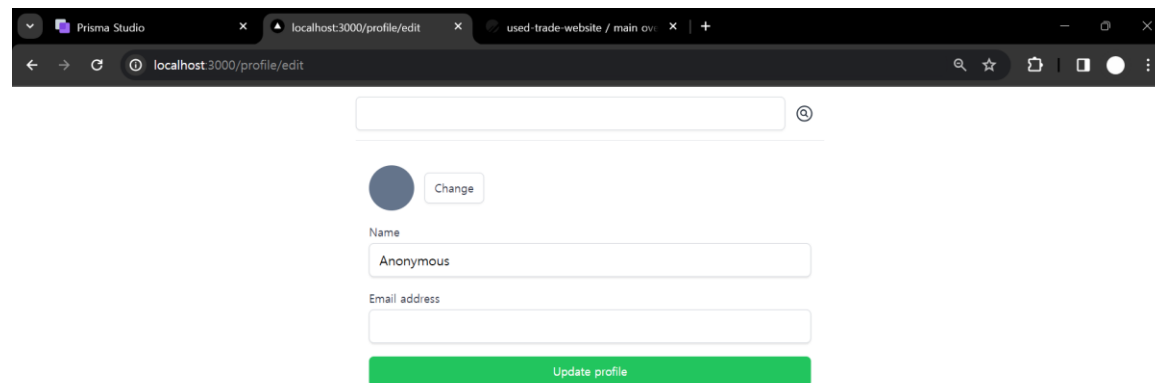
- 상단에는 검색창이 존재한다. // layout.tsx
  - \*유저가 검색창에 검색어를 넣고 버튼을 누르면, 홈 화면으로 강제로 이동하며 검색의 결과를 보여준다
- 프로필 이미지 변경 버튼, 닉네임 변경 칸, 이메일 변경 칸이 존재한다. //profile/index.tsx
- 변경 요소들 밑에는 저장 버튼이 위치해있다.



## 04 Project 수행 결과

### 프로필 수정 내부 동작

- 유저가 검색 버튼을 누르면 강제로 홈 화면으로 이동시킨다. // layout.tsx
- Change버튼을 누르면 업로드 창이 뜨며, 그 중 이미지를 선택하여 프로필 사진을 바꿀 수 있다.
- 이름과 메일 주소 칸이 존재하며, 해당 부분에 값을 바꾸고 초록색 저장 버튼을 누르면 값이 저장된다. //edit.tsx



## 05 설계 요소 평가

### > 성능

데이터 크롤링 시간이 상당히 길며, 매 검색마다 데이터 크롤링을 진행한다는 점에서 사이트의 동작 속도가 원할하지 못하다.

### > 안정성

데이터 베이스에 유저 정보를 저장하므로, 데이터 베이스가 망가지지 않는한 유저 정보는 잘 유지된다.

### > 보안성

유저가 로그인시 서버에 쿠키를 넘겨줄 때 iron-session/next를 씌으로써 쿠키를 암호화하여 보안성을 강화하였다. 그리고 쿠키가 일정 시간이 지나면 만료되도록 설정하였다. 이 쿠키는 사용자의 세션 정보를 안전하게 유지하고, 무단 액세스를 방지하기 위해 쓴다.

# 05 설계 요소 평가

## > 구현 기간

UI제작 - 2주 / 백엔드 제작 - 3주 / 데이터 크롤링 - 1주 / UI <-> 백엔드 연결 - 1주

## 06 추후 Project 발전 방향

### 추가 가능한 요소

- 1) 데이터 크롤링 시간 단축하는 방법 고안
- 2) 더 많은 중고 사이트의 데이터를 크롤링
- 3) 이름순, 최저가순, 최고가순 등 특정 기준으로 정렬하는 기능 추가
- 4) 타 사이트 정보만을 가져오는 것만이 아닌, 사이트 내에서 직접 업로드
- 5) 4)에 따른 유저간 소통 요소 추가 ex. 채팅, 유저 상호평가, 후기 등







THANK YOU

