[사용자] 좋아요 기능 구현하기

구현 어려움 정도	⭐⭐⭐
구현 완료	☑
기간	@December 14, 2024
정리 완료	☑

💡 drizzle orm을 활용해 좋아요 기능 구현하기

1. 데이터베이스 모델 설정(Schema 만들기)

- `user 테이블`: 사용자 정보를 저장하는 테이블
  - 각 사용자는 고유한 id(uuid), name, password, 생성 및 업데이트 시간을 가집니다.
- `like 테이블`: 사용자가 좋아요한 코인에 대한 정보를 저장하는 테이블
  - 각 좋아요 항목은 고유 id와 사용자 id(userId), 코인 심볼, 생성시간을 가집니다.
- 관계 설정
  - user와 like는 `1:다` 관계입니다.
  - 하나의 사용자는 여러개의 좋아요(코인심볼)를 가질 수 있습니다.(one to many 관계)
  - https://orm.drizzle.team/docs/relations#one-to-many

```
import { relations } from "drizzle-orm";
import { pgTable, timestamp, uuid, text, varchar } from "drizzle-orm/pg-core";

// 사용자 테이블 정의
export const user = pgTable('users', {
  id: uuid('id').defaultRandom().notNull().primaryKey(),
  name: text('name').notNull(),
  email: text('email').notNull().unique(),
  password: text('password').notNull(),
  createdAt: timestamp("created_at").defaultNow().notNull(),
  updatedAt: timestamp("updated_at").defaultNow().notNull(),
});

export const userRelations = relations(user, ({ many }) => ({
  likes: many(like),
}));

// 좋아요 테이블 정의
export const like = pgTable("likes", {
  id: uuid('id').defaultRandom().notNull().primaryKey(),
  userId: uuid('userId').references(() => user.id, {onDelete: 'cascade'}).notNull(),
  coinSymbol: varchar("coinSymbol", { length: 50 }).notNull(),
  createdAt: timestamp("created_at").defaultNow().notNull(),
});

export const likeRelations = relations(like, ({ one, many }) => ({
  user: one(user, {
    fields: [like.userId],
    references: [user.id]
  }),
}));
```

2. 좋아요 추가, 삭제, 확인

1) addLike - 좋아요 추가

- 세션확인: verifySession()을 통해 현재 로그인한 사용자의 정보를 확인
- 중복 좋아요 체크: 이미 해당 코인에 대해 좋아요가 존재하는지 like 테이블에서 확인
- 중복된 좋아요를 방지하기 위해 userId와 coinSymbol을 기준으로 조회
- 좋아요 추가: 좋아요가 없으면 새로 좋아요를 추가합니다.

```
import { db } from "@/db";
import { verifySession } from "./sessions";
import { like } from "@/db/schema";
import { eq, and } from "drizzle-orm";
import { revalidatePath } from "next/cache";

// 좋아요 추가 함수
export const addLike = async (symbol: string) => {
  const session = await verifySession(); // 로그인 세션 확인
  const userId = session.id;

  // 이미 해당 코인에 대한 좋아요가 있는지 확인
  const existingLike = await db
    .select()
    .from(like)
    .where(and(eq(like.userId, userId), eq(like.coinSymbol, symbol)))
    .limit(1); // 이미 있으면 하나만 반환

  // 좋아요가 없다면 새로 추가
  if (existingLike.length === 0) {
    await db.insert(like).values({
      userId,
      coinSymbol: symbol,
    });
  } else {
    console.log("이미 좋아요가 존재합니다.");
  }
};
```

2) removeLike - 좋아요 삭제

- 사용자가 좋아요한 항목을 삭제합니다.
- userId와 coinSymbol을 기준으로 삭제할 항목을 찾아 삭제합니다.

```
export const removeLike = async (symbol: string) => {
  const session = await verifySession(); // 로그인 세션 확인
  const userId = session.id;

  // 해당 유저와 코인 심볼에 해당하는 좋아요 데이터 삭제
  await db
    .delete(like)
    .where(and(eq(like.userId, userId), eq(like.coinSymbol, symbol)));
};
```

3) checkLike - 좋아요 확인

- 사용자가 해당 코인에 대해 좋아요를 했는지 확인하는 함수

- 존재하면 true를 반환하고 존재하지 않으면 false를 반환

```
export const checkLike = async (symbol: string) => {
  const session = await verifySession();
  const userId = session.id;

  const existingLike = await db
    .select()
    .from(like)
    .where(and(eq(like.userId, userId), eq(like.coinSymbol, symbol)))
    .limit(1);

  return existingLike.length > 0; // 좋아요가 있으면 true, 없으면 false
};
```

3. LikeButton component

- 사용자가 코인에 좋아요 버튼을 클릭하면 `handleLikeToggle` 함수가 호출
- 서버에서 좋아요 상태를 확인하고 그에 따라 `addLike` 또는 `removeLike`를 호출하여 데이터베이스에 반영
- 데이터베이스 업데이트 후 UI 업데이트(하트 색깔 변함)

- `handleLikeToggle`: 좋아요 버튼을 클릭하면 현재 상태를 반전시키고 서버에 좋아요를 추가하거나 삭제
- `checkIfLiked`: debounce를 사용해 자주 서버에 요청보내지 않도록 하며 사용자가 클릭할 때마다 좋아요 상태를 확인
- 사용자가 좋아요를 클릭할 때마다 서버로 요청을 보내고 그 결과를 바탕으로 ui 상태(isLiked)를 업데이트
- 서버 요청이 진행 중일 때는 버튼을 비활성화해 사용자 인터페이스를 변하지 않도록 함

```
import { useState, useEffect } from 'react';
import { addLike, removeLike, checkLike } from "@/actions/likes"; // 서버 사이드 함수들
import HeartSVG from '@/components/svg/HeartSVG/HeartSVG';
import { debounce } from 'lodash';

interface LikeButtonProps {
  coinSymbol: string;
}

const LikeButton: React.FC<LikeButtonProps> = ({ coinSymbol }) => {
  const [isLiked, setIsLiked] = useState(false);
  const [isUpdating, setIsUpdating] = useState(false);

  // 서버에서 좋아요 상태 확인
  const checkIfLiked = debounce(async (symbol: string) => {
    const liked = await checkLike(symbol);
    setIsLiked(liked);
  }, 500);

  // 컴포넌트가 처음 렌더링 될 때와 coinSymbol이 변경될 때마다 실행
  useEffect(() => {
    if (coinSymbol) {
      checkIfLiked(coinSymbol);
    }
  }, [coinSymbol]);

  // 좋아요 토글 처리
  const handleLikeToggle = async () => {
    if (isUpdating) return; // 이미 업데이트 중이라면 클릭 방지
```

```
    setIsUpdating(true); // 업데이트 시작

    const newLikedState = !isLiked; // 새로운 좋아요 상태
    setIsLiked(newLikedState); // UI 즉시 반영

    try {
      // 좋아요 상태에 따라 addLike 또는 removeLike 호출
      if (newLikedState) {
        await addLike(coinSymbol);
      } else {
        await removeLike(coinSymbol);
      }
    } catch (error) {
      setIsLiked(isLiked); // 실패시 이전 상태로 되돌리기
      console.error(error, "LikeButton 업데이트에 실패했습니다.");
    } finally {
      setIsUpdating(false); // 업데이트 완료
    }
  };

  return (
    <button
      onClick={handleLikeToggle}
      disabled={isUpdating} // 버튼 비활성화
    >
      <HeartSVG color={isLiked ? '#EF4444' : '#ACB0B4'} /> {/* 좋아요 상태에 따라 색상 변경 */}
    </button>
  );
};

export default LikeButton;
```