JalaUniversity

Programación IV

Lab Week5

Name: Aldo Ochoa Condori

Tutor: Luis Morales Ledezma

Topic: Principios Solidos en programación funcional y monadas

04/08/2024

Materia: Algorítmica I
Estudiante: Aldo Ochoa Condori

Identificación del Caso de Uso

Al revisar he notado que en algunas partes de mi proyecto capstone donde podría aplicar estos conceptos:

1. Validación y Procesamiento de Datos de Entrada:
    - Ejemplo: En mis funciones de registerCard, registerGame, y registerPlayer, estoy validando los datos de entrada. Aquí puedo ver que aplicamos un Functor para transformar los datos validados. O bueno los schemas ya que se asemejan más a la lógica de negocio e implementación

```
import z from 'zod'
import { extractValidationData } from '../errors/extractErrorData.js'

const cardSchema = z.object({
k to add a breakpoint ring({
        invalid_type_error:'color must be a string',
        required_error:'color required'
    }),
    value:z.string({
        invalid_type_error:'valuemust be a string',
        required_error:'value is required'
    }),
    gameId:z.number({
        invalid_type_error:'gameId must be a number',
        required_error:'gameId is required'
    })
})

export function validateCard(data){
    const result = cardSchema.safeParse(data);

    const{
        hasError,
        errorMessages,
        data:cardData,
    } = extractValidationData(result);
    return {
        hasError,
        errorMessages,
        cardData,
    }
}

export function validatePartialCard(data){
    const result = cardSchema.partial().safeParse(data);
    const{
        hasError,
        errorMessages,
        data:cardData,
    } = extractValidationData(result)
    return {
        hasError,
        errorMessages,
        cardData,
    }
```

Tutor: Luis Morales

Materia: Algorítmica I
Estudiante: Aldo Ochoa Condori

```
import z from 'zod';
import { extractValidationData } from '../errors/extractErrorData.js';


const gameSchema = z.object({
    title: z.string({
        invalid_type_error: 'title must be a string',
        required_error: 'title is required'
    }),
    status: z.enum(['active', 'inactive'], {
        invalid_type_error: 'status must be either "active" or "inactive"',
        required_error: 'status is required'
    }),
    maxPlayers: z.number({
        invalid_type_error: 'maxPlayers must be a number',
        required_error: 'maxPlayers is required'
    }),
    rules:z.string({
        invalid_type_error: 'rules must be a string',
        required_error: 'rules is required'
    }),
});


export function validateGame(data){

    const result = gameSchema.safeParse(data);
    const {
        hasError,
        errorMessages,
        data:gameData,
    } = extractValidationData(result);
    return {
        hasError,
        errorMessages,
        gameData,
    }
}

export function validatePartialGame(data){
    const result = gameSchema.partial().safeParse(data)
    const {
        hasError,
        errorMessages,
        data:gameData,
```

Tutor: Luis Morales

Materia: Algorítmica I
Estudiante: Aldo Ochoa Condori

```
import z from 'zod';
import { extractValidationData } from '../errors/extractErrorData.js';

const playerSchema = z.object({
    name:z.string({
        invalid_type_error: 'name must be a string',
        required_error: 'name is required'
    })
        .min(1,{message:'name is too short, minimum one character'})
        .max(50,{message:'name is too long, maximum 50 characteres'}),

    age:z.number({
        invalid_type_error:'age must be a integer',
        required_error:'age is required'
    }),

    email:z.string({
        invalid_type_error:'email must be a string',
        required_error:'email is required'
    }),
    username:z.string({
        invalid_type_error:'username must be a string',
        required_error:'email is required'
    }),
    password:z.string({
        invalid_type_error:'password must be a string',
        required_error:'email is required'
    })
})

export function validatePlayer(data){
    const result = playerSchema.safeParse(data)
    const{
        hasError,
        errorMessages,
        data:playerData,
    } = extractValidationData(result)
    return {
        hasError,
        errorMessages,
        playerData
    }
}
```

Tutor: Luis Morales

2. Operaciones Asíncronas Encadenadas:
    - Ejemplo: En startGame, manejo operaciones asincrónicas encadenadas. Aquí podría usar Monads para manejar el flujo de datos de manera más limpia.

Voya Reflexionar desde la experiencia en cuanto a clase y mi realización con el proyecto capstone para poder elegir entre Functor y Monad

En mi proyecto, he identificado dos casos de uso: la validación de datos y el manejo de operaciones asincrónicas. Para la validación, un Functor es más adecuado, ya que me permite transformar y encadenar datos validados de manera efectiva. Utilizaría un tipo como Either para manejar resultados exitosos o errores de validación. Por otro lado, para manejar operaciones asincrónicas encadenadas, una Monad es ideal. Las Promesas en JavaScript actúan como Monads, facilitando el encadenamiento de operaciones asíncronas de manera limpia y manejando los resultados de manera eficiente. Por lo tanto, elegiría Functors para validación y Monads para operaciones asincrónicas.

Materia: Algorítmica I
Estudiante: Aldo Ochoa Condori

Asegúrate de que tu implementación funcione correctamente dentro del proyecto.Utilizando Jest, prueba varios escenarios para verificar que el Functor o Monad maneja los datos como se espera.

```
describe('Card Controller', () => {
    beforeAll(async () => {
        await sequelize.sync({ force: true });
    });

    afterEach(() => {
        jest.clearAllMocks();
    });

    afterAll(async () => {
        await sequelize.close();
    });

    test('POST /cards/register should create a card', async () => {
        const cardData = { color: "blue", value: "10p", gameId: 3 };
        CardService.CreateCard.mockResolvedValue({ id: 1, ...cardData });

        const response = await request(app)
            .post('/cards/register')
            .send(cardData);

        expect(response.status).toBe(200);
        expect(response.body.color).toBe("blue");
        expect(response.body.value).toBe("10p");
    });

    test('GET /cards should return all cards', async () => {
        const cards = [{ id: 1, color: "blue", value: "10p", gameId: 3 }];
        CardService.findAllCards.mockResolvedValue(cards);

        const response = await request(app).get('/cards');

        expect(response.status).toBe(200);
        expect(response.body.length).toBe(1);
        expect(response.body[0].color).toBe("blue");
    });

    test('GET /cards/:id should return a card by id', async () => {
        const card = { id: 1, color: "blue", value: "10p", gameId: 3 };
        CardService.findOneCard.mockResolvedValue(card);

        const response = await request(app).get('/cards/1');
```

Tutor: Luis Morales

Materia: Algorítmica I
Estudiante: Aldo Ochoa Condori

```
test('GET /games should return all games', async () => {
    const games = [
        { id: 1, title: "New Game", status: "active", maxPlayers: 4, rules: "Some rules" }
    ];
    GameService.findAllGames.mockResolvedValue(games);

    const response = await request(app).get('/games');

    expect(response.status).toBe(200);
    expect(response.body.length).toBe(1);
    expect(response.body[0].title).toBe("New Game");
});

test('GET /games/:id should return a game by id', async () => {
    const game = { id: 1, title: "New Game", status: "active", maxPlayers: 4, rules: "Some rules" };
    GameService.findOneGame.mockResolvedValue(game);

    const response = await request(app).get('/games/1');

    expect(response.status).toBe(200);
    expect(response.body.title).toBe("New Game");
});

test('PATCH /games/:id should update a game', async () => {
    const gameUpdate = { title: "Updated Game" };
    const updatedGame = { id: 1, title: "Updated Game", status: "active", maxPlayers: 4, rules: "Some rules" };
    GameService.updateGame.mockResolvedValue(updatedGame);

    const response = await request(app)
        .patch('/games/1')
        .send(gameUpdate);

    expect(response.status).toBe(200);
    expect(response.body.title).toBe("Updated Game");
});

test('DELETE /games/:id should delete a game', async () => {
    GameService.deleteGame.mockResolvedValue({ id: 1 });
```

Tutor: Luis Morales