Resuelto-Convo-CC-2425.pdf.pdf

CVV__

Diseño y Pruebas i

3º Grado en Ingeniería Informática - Ingeniería del Software

Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla

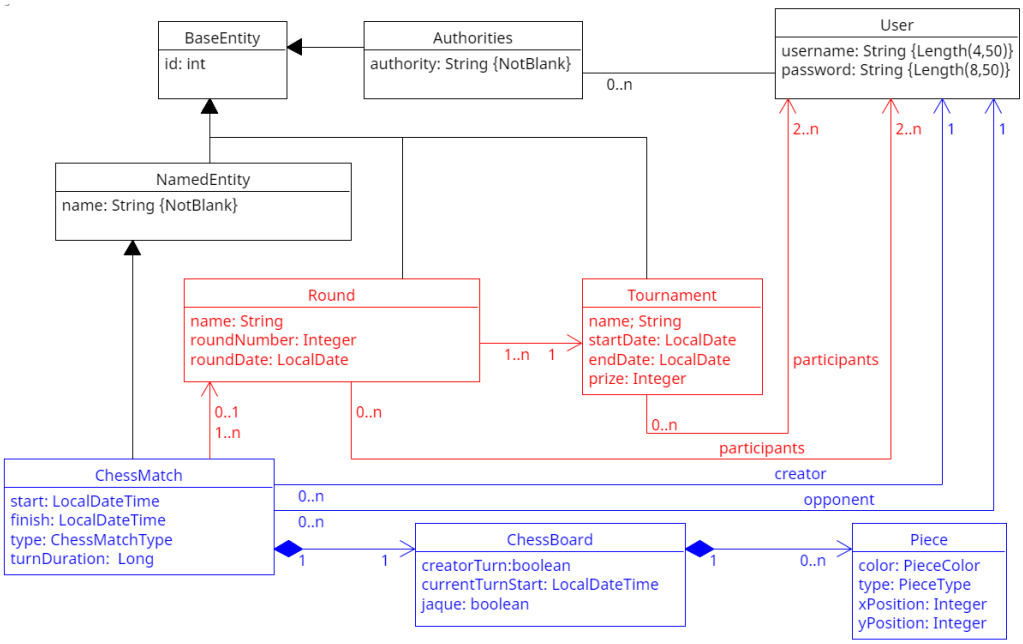
Accede al documento original

Control práctico de DP1 2024-20254(Primera Convocatoria Enero 2025)

Enunciado

En este ejercicio, añadiremos la funcionalidad de gestión de torneos para una implementación del juego del ajedrez. Concretamente, se proporciona una clase "ChessMatch" que representa las partidas que se juegan, y que tiene asociada una instancia de la clase "ChessBoard" que representa el estado del tablero para dicha partida, por lo que tendrá asociada un conjunto de instancias de la clase "Piece". Además, tendremos las clases "Tournament", que representa a un torneo y la clase "Round" que representa a cada una de las rondas de un torneo. Tanto el torneo como las rondas tienen un conjunto de usuarios que son los participantes, que se han expresado en el diagrama con sendas relaciones.

El diagrama UML que describe las clases y relaciones con las que vamos a trabajar es el siguiente:



Las clases para las que realizaremos el mapeo objeto-relacional como entidades JPA se han señalado en rojo. Las clases en azul son clases que se proporcionan ya mapeadas pero con las que se trabajará durante el control de laboratorio.

Realizaremos una serie de ejercicios basados en funcionalidades que implementaremos en el sistema, y validaremos mediante pruebas unitarias. Si desea ver el resultado que arrojarían las pruebas en backend, puede ejecutarlas (bien mediante su entorno de desarrollo favorito, bien mediante el comando "*mvnw test*" en la carpeta raíz del proyecto). Cada ejercicio correctamente resuelto valdrá un punto, el número de casos de prueba de cada ejercicio puede variar entre uno y otro y la nota se calculará en base al porcentaje de casos de prueba que pasan. Por ejemplo, si pasan la mitad (50%) de los casos de prueba de un ejercicio, en lugar de un punto usted obtendrá un 0.5.

Para comenzar el control debe aceptar la tarea de este control práctico a través del siguiente enlace:

https://classroom.github.com/a/mej8q4PH

Al aceptar dicha tarea, se creará un repositorio único individual para usted, debe usar dicho repositorio para realizar el control práctico. Debe entregar la actividad en EV asociada al control check proporcionando como texto la dirección url de su repositorio personal. Recuerde que además debe entregar su solución del control.

La entrega de su solución al control se realizará mediante un único comando "*git push*" a su repositorio individual. Recuerde que debe hacer push antes de cerrar sesión en la computadora y abandonar el aula, de lo contrario, su intento se evaluará como no presentado. Su primera tarea en este control será clonar (recuerde que si va a usar los equipos del aula para realizar el control necesitará usar un token de autenticación de GitHub como clave, tiene un documento de ayuda a la configuración en el propio repositorio del control). A continuación, deberá importar el proyecto en su entorno de desarrollo favorito y comenzar los ejercicios abajo listados. Al importar el proyecto, el mismo puede presentar errores de compilación. No se preocupe, si existen, dichos errores irán despareciendo conforme usted vaya implementando los distintos ejercicios del control.

Nota importante 1: No modifique los nombres de las clases ni la signatura (nombre, tipo de respuesta y parámetros) de los métodos proporcionados como material de base para el control. Las pruebas que se usan para la evaluación dependen de que las clases y los métodos tengan la estructura y nombres proporcionados. Si los modifica probablemente no pueda hacer que pasen las pruebas, y obtendrá una mala calificación.

Nota importante 2: No modifique las pruebas unitarias proporcionadas como parte del proyecto bajo ningún concepto. Aunque modifique las pruebas en su copia local del proyecto, éstas serán restituidas mediante un comando git previamente a la ejecución de las pruebas para la emisión de la nota final, por lo que sus modificaciones en las pruebas no serán tenidas en cuenta en ningún momento.

Nota importante 3: Mientras haya ejercicios no resueltos habrá tests que no funcionen y, por tanto, el comando "*mvnw install*" finalizará con error. Esto es normal debido a la forma en la que está planteado el control y no hay que preocuparse por ello. Si se quiere probar la aplicación se puede ejecutar de la forma habitual pese a que "mvnw install" finalice con error.

Nota importante 4: La descarga del material de la prueba usando git, y la entrega de su solución con git a través del repositorio GitHub creado a tal efecto forman parte de las competencias evaluadas durante el examen, por lo que no se aceptarán entregas que no hagan uso de este medio, y no se podrá solicitar ayuda a los profesores para realizar estas tareas.

Nota importante 5: No se aceptarán como soluciones válidas proyectos cuyo código fuente no compile correctamente o que provoquen fallos al arrancar la aplicación en la inicialización del contexto de Spring. Las soluciones cuyo código fuente no compile o incapaces de arrancar el contexto de Spring serán evaluadas con una nota de 0.

Test 1 – Creación de las entidades Tournament y Round y sus repositorios asociados

Modificar las clases "Tournament" y "Round" para que sean entidades. Estas clases están alojadas en el paquete "es.us.dp1.chess.tournament.round", y deben tener los siguientes atributos y restricciones:

Para ambas clases:

- El atributo de tipo entero (Integer) llamado "id" actuará como clave primaria en la tabla de la base de datos relacional asociada a la entidad.
- Un atributo de tipo cadena de caracteres (String) llamado "name" obligatorio (no puede ser nulo), que debe tener una longitud mínima de **5** caracteres y máxima de **60** y que no puede estar formada por caracteres vacíos (espacios, tabuladores, etc.).[1]

Para la clase Tournament:

- El atributo de tipo entero (Integer) llamado "*prize*", que representa el precio en metálico que se otorgará al ganador del torneo. Este atributo será *obligatorio* y tendrá un *valor mínimo de 1*.
- El atributo de tipo fecha (LocalDate) llamado "startDate", que representa fecha de comienzo del torneo y es obligatorio.
- El atributo de tipo fecha (LocalDate) llamado "finishDate", que representa fecha de finalización del torneo y es obligatorio.

Para la clase Round:

- El atributo de tipo entero (Integer) llamado "roundNumber" obligatorio que tendrá un valor mínimo de 1, un valor máximo de 8, y es obligatorio.
- Un atributo de tipo fecha (LocalDate) llamado "roundDate" que representa la fecha en la que se celebra la ronda y es obligatorio.

No modifique por ahora las anotaciones @Transient de las clases. Modificar las interfaces "TournamentRepository" y "RoundRepository" alojadas en el mismo paquete para que extiendan a CrudRepository. No olvide especificar sus parámetros de tipo.

Test 2 – Creación de relaciones entre las entidades

Elimine las anotaciones @Transient de los métodos y atributos que las tengan en las entidades creadas en el ejercicio anterior. Se pide crear las siguientes relaciones entre las entidades:
Cree una relación unidireccional desde "Round" hacia "Tournament" que exprese la que aparece en el diagrama UML (mostrado en la primera página de este enunciado) respetando sus cardinalidades, usando el atributo "tournament" de la clase "Round".

Además, se pide crear dos relaciones unidireccionales desde "Tournament" y "Round" hacia "User" que representen las que aparecen en el diagrama UML, tenga en cuenta la cardinalidad que tienen, usando como nombre de los atributos "participants". Debe asegurarse de que las relaciones expresan adecuadamente la cardinalidad que muestra el diagrama UML, por ejemplo, algunos atributos pueden ser

[1] Nótese que estas restricciones no coinciden con las de NamedEntity.

nulos puesto que la cardinalidad es 0..n pero otros no, porque su cardinalidad en el extremo navegable de la relación es 1..n.

Finalmente, se pide crear una relación unidireccional desde "ChessMatch" hacia "Round" que represente la que aparece en el diagrama, usando como nombre de atributo "round" en la clase "ChessMatch". Debe asegurarse de que las relaciones expresan adecuadamente la cardinalidad que muestra el diagrama UML, por ejemplo, en este caso el atributo si podría ser nulo, puesto que la cardinalidad es "0..1" en el extremo de "Round", pero si fuera "1" o "1..n" sería obligatorio.

Test 3 – Modificación del script de inicialización de la base de datos para incluir un torneo y dos rondas

Modificar el script de inicialización de la base de datos, para que se creen los siguientes torneos (Tournaments) y rondas (Rounds):

Tournament:

- id: 1
- name: "Los Palacios Chess Tournament"
- startDate: 2024-10-01
- endDate: 2024-10-15
- prize: 1000

Round 1:

- id: 1
- name: "SemiFinals"
- roundNumber: 2
- roundDate: 2024-10-07

Round 2:

- id: 2
- name: "Finals"
- roundNumber: 1
- roundDate: 2024-10-15

Test 4 – Modificación del script de inicialización de la base de datos para relacionar el torneo y las rondas

Modificar este script de inicialización de la base de datos para que:

- Las rondas cuyos id son 1 y 2 se asocien con el torneo cuyo id es 1
- El Torneo cuyo id es 1 tenga como participantes a los usuarios cuyos id son 4,5,6 y 7.
- La Ronda cuyo id es 1 tenga como participantes a los usuarios cuyos id son 4,5,6 y 7.
- La Ronda cuyo id es 2 tenga como participantes a los usuarios cuyos id son 4 y 7.

Tenga en cuenta que el orden en que aparecen los INSERT en el script de inicialización de la base de datos es relevante al definir las asociaciones.

Test 5 – Creación de servicios de gestión de los síntomas y los tratamientos

Modificar las clases "RoundService" y "TournamentService", para que sean un servicio Spring de lógica de negocio y proporcionen una implementación a los métodos que permitan:

1. Obtener todos los torneos/rondas (*Tournament / Round*) almacenados en la base de datos como una lista usando el repositorio (método *getAll* en cada uno de los servicios correspondientes).
2. Obtener un torneo/ronda (*Tournament / Round*) por Id (método *getById* en cada uno de los servicios correspondientes).
3. Grabar un torneo/ronda (*Round/ Tournament*) en la base de datos (método *save*).

Todos estos métodos **deben ser transaccionales**, pero las anotaciones asociadas deben realizarse a nivel de método, no a nivel de clase**.** No modifique por ahora la implementación del resto de métodos de los servicios.

Test 6 – Anotar el repositorio de partidas con una consulta compleja

Modificar la consulta personalizada que puede invocarse a través del método "comprisedMatches" del repositorio de partida "MatchRepository" (alojado en el paquete es.us.dp1.chess.tournament.match) que reciba como parámetro dos instantes determinados (que definen un rango donde el primero es anterior al segundo). El objetivo es que devuelva el número de partidas celebradas en ese rango de tiempo.

Test 7 – Creación del controlador para devolver las Partidas disponibles

Crear un método en el controlador "Chess*MatchController"* (alojada en el paquete es.us.dp1.chess.tournament.match) que permita devolver todas las partidas existentes. El método debe responder a peticiones tipo GET en la URL:

http://localhost:8080/api/v1/matches

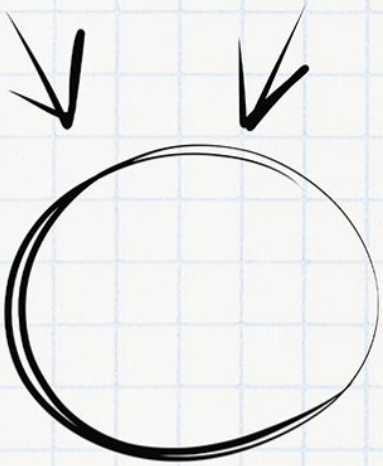
Así mismo debe crear un método para devolver una partida concreta, este método debe responder a peticiones en la url http://localhost:8080/api/v1/matches/X donde X es la Id de la partida a obtener, y devolverá los datos de la partida correspondiente. En caso de que no exista una partida con el id especificado el sistema debe devolver el código de estado 404 (NOT_FOUND).

Imagínate aprobando el examen

Necesitas tiempo y concentración

Planes	PLAN TURBO	PLAN PRO	PLAN PRO+
Descargas sin publi al mes	10	40	80
Elimina el video entre descargas	✓	✓	✓
Descarga carpetas	✗	✓	✓
Descarga archivos grandes	✗	✓	✓
Visualiza apuntes online sin publi	✗	✓	✓
Elimina toda la publi web	✗	✗	✓
Precios Anual	0,99 € / mes	3,99 € / mes	7,99 € / mes

Ahora que puedes conseguirlo, ¿Qué nota vas a sacar?



WUOLAH

Test 8 – Modificar el servicio de gestión de Partidas para que no se puedan tener partidas concurrentes

Modificar el método save del servicio de gestión de partidas (ChessMatchService) de manera que se lance la excepción (ConcurrentMatchExcepction) en caso de que se intente guardar una partida como empezada y no acabada (con un instante de inicio distinto de nulo e instante de fin nulo) cuando ya existe otra partida distinta para los usuarios de esa partida (ya sean como creador o como oponente) empezada pero no acabada (con un instante de inicio distinto de nulo e instante de fin nulo). El método debe ser transaccional y hacer rollback de la transacción en caso de que se lance dicha excepción. Para determinar las partidas del jugadoren cada caso debe usar los métodos findByCreator y findByOpponent de "MatchRepository" que porporcionarán las partidas en las que está involucrado el usuario que se pasa por parámetro.

Test 9 – Implementar una prueba para un algoritmo de validación movimientos para piezas de tipo caballo.

Con el objetivo de facilitar la especificación correcta de movimientos en las partidas, se ha decidido crear un conjunto algoritmo que permita validar los movimientos de las distintas piezas. En este ejercicio crearemos un conjunto de casos de prueba que permitan comprobar que el validador funciona correctamente.

La interfaz del algoritmo está especificada en la interfaz "HorseMoveValidator" que se encuentra en el paquete "es.us.dp1.chess.tournament.match". Y se proporcionan cuatro implementaciones del algoritmo. Concretamente el método que valida los movimientos tiene la siguiente signatura: boolean isValid(int originX, int originY, int destinationX, int destinationY). Si se le pasan al validador posiciones inválidas por definición (nótese que los valores deberían estar siempre entre 0 y 7 para representar posiciones válidas en el tablero), el validador deber rechazar el movimiento como inválido (incluso aunque el error esté en la posición de origen).

Modifique la clase de pruebas llamada "HorseMoveValidatorTest" que se encuentra en la carpeta "scr/test/java/es/us/dp1/tests/horsemovevalidator" y especifique tantos métodos con casos de prueba como considere necesarios para validar el correcto funcionamiento del algoritmo. Nótese que la clase tiene un atributo de tipo HorseMoveValidatorTest llamado algorithm, use dicho atributo como sujeto bajo prueba en todos sus métodos de prueba.

Su implementación del test **no debe usar mocks, ni anotaciones de pruebas de spring** (@DataJpaTest, @SpringBootTest,etc.), **ni tests parametrizados, y todos los métodos anotados con @Test deben ser sin parámetros**.

Recuerde que el movimiento del caballo en ajedrez tiene la siguiente estructura:

Test 10 – Creación de un componente React de listado de partidas

Modificar el componente React proporcionado en el fichero "frontend/src/match/index.js" para que muestre un listado de las partidas disponibles en el sistema. Para ello debe hacer uso de la API lanzando una petición tipo GET contra la URL api/v1/matches.

Este componente debe mostrar las partidas en una tabla (se recomienda usar el componente Table de reactstrap) incluyendo columnas para su nombre y sus participantes. En la columna de participantes, para cada partida, se mostrará una lista no ordenada con el nombre de los dos participantes.

Para poder lanzar esta prueba y comprobar su resultado puede colocarse en la carpeta de frontend y ejecutar el comando npm test y pulsar 'a' en el menú de comandos de jest. Nótese que previamente debe haber lanzado al menos una vez el comando npm install para que todas las librerías de node estén instaladas.

CONVO - CONTROL CHECK 24/25

Test 1 y 2

```
@Getter
@Setter
@Entity
public class Tournament extends BaseEntity {

    @NotNull @NotBlank @Size(min = 5, max=60)
    String name;

    @NotNull @Min(1)
    Integer prize;

    @NotNull @DateTimeFormat(pattern = "dd/MM/yyyy")
    LocalDate startDate;

    @NotNull@DateTimeFormat(pattern = "dd/MM/yyyy")
    LocalDate endDate;

    @ManyToMany
    @NotNull //puede que haya que añadir min 2
    List<User> participants;
}
```

```
@Getter
@Setter
@Entity
public class Round extends BaseEntity {
    @NotNull
    @NotBlank
    @Size(min = 5, max=60)
    String name;

    @NotNull @Min(1) @Max(8)
    Integer roundNumber;

    @NotNull @DateTimeFormat(pattern = "dd/MM/yyyy")
    LocalDate roundDate;

    @ManyToOne @NotNull
    Tournament tournament;

    @ManyToMany @NotNull //puede que haya que añadir min 2
    List<User> participants;
```

```
}
```

***Chessmatch**

```
@ManyToOne
Round round;
```

Test 3 y 4

```
INSERT INTO tournament(id, name, start_date, end_date, prize)
VALUES
(1, 'Los Palacios Chess Tournament', '2024-10-01',
'2024-10-15', 1000);

INSERT INTO round(id, name, round_number, round_date,
tournament_id) VALUES
(1, 'SemiFinals', 2, '2024-10-07', 1), (2,'Finals', 1,
'2024-10-15', 1);

INSERT INTO tournament_participants(tournament_id,
participants_id) VALUES
(1, 4), (1, 5), (1, 6), (1, 7);

INSERT INTO round_participants(round_id, participants_id)
VALUES
(1, 4), (1, 5), (1, 6), (1, 7), (2, 4), (2, 7);
```

Test 5

```
@Service
public class RoundService {
    RoundRepository roundRepository;

    public RoundService(RoundRepository roundRepository) {
        this.roundRepository = roundRepository;
    }
    @Transactional(readOnly = true)
    public List<Round> getAll() {
        return roundRepository.findAll();
    }
    @Transactional(readOnly = true)
    public Round getById(Integer id) {
        Optional<Round> round = roundRepository.findById(id);
        return round.isPresent()? round.get():null;
    }
```


```
    @Transactional()
    public void save(Round round) {
        roundRepository.save(round);
    }
}
```

```
@Service
public class TournamentService {
    TournamentRepository tournamentRepository;

    public TournamentService(TournamentRepository
tournamentRepository) {
        this.tournamentRepository = tournamentRepository;
    }
    @Transactional(readOnly = true)
    public List<Tournament> getAll() {
        return tournamentRepository.findAll();
    }

    @Transactional(readOnly = true)
    public Tournament getById(Integer id) {

        Optional<Tournament> tournament =
tournamentRepository.findById(id);
        return tournament.isPresent()? tournament.get():null;
    }
    @Transactional
    public void save(Tournament tournament) {
        tournamentRepository.save(tournament);
    }
}
```

Test 6

```
@Query("""
SELECT COUNT(m) FROM ChessMatch m WHERE m.start >= :start AND
m.finish <= :end
""")
Integer comprisedMatches(LocalDateTime start, LocalDateTime
end);
```

Test 7

```
@RestController
public class ChessMatchController {
```

```
    UserService userService;
    ChessMatchService matchService;

    @Autowired
    public ChessMatchController(ChessMatchService
service,UserService userService){
        this.matchService=service;
        this.userService=userService;
    }

    @GetMapping("/api/v1/matches")
    @ResponseStatus(HttpStatus.OK)
    public ResponseEntity<List<ChessMatch>> getAllMatches(){
        return new ResponseEntity<>(matchService.getMatches(),
HttpStatus.OK);
    }

    @GetMapping("/api/v1/matches/{matchId}")
    @ResponseStatus(HttpStatus.OK)
    public ResponseEntity<ChessMatch> getMatch(@PathVariable
Integer matchId){
        if (matchService.getMatchById(matchId).isEmpty())
            throw new ResourceNotFoundException("Match not
found");
        return new
ResponseEntity<>(matchService.getMatchById(matchId).get(),
HttpStatus.OK);
    }

}
```

Test 8

***Hay que modificar** **ConcurrentMatchExcepction**

```
public class ConcurrentMatchException extends RuntimeException
{ }
```

```
@Transactional(rollbackFor = ConcurrentMatchException.class)
public ChessMatch save(ChessMatch m) throws
ConcurrentMatchException {

    if (creatorHasOngoingMatch(m)) {
        throw new ConcurrentMatchException();
    }
    if (opponentHasOngoingMatch(m)) {
```

```
        throw new ConcurrentMatchException();
    }
    return repo.save(m);
}

public Boolean creatorHasOngoingMatch(ChessMatch m){
    return repo.findByCreator(m.getCreator()).stream()
        .anyMatch(match -> !match.getId().equals(m.getId()) &&
            match.getStart() != null &&
            match.getFinish() == null);

}
public Boolean opponentHasOngoingMatch(ChessMatch m){
    return repo.findByOpponent(m.getOpponent()).stream()
        .anyMatch(match -> !match.getId().equals(m.getId()) &&
            match.getStart() != null &&
            match.getFinish() == null);
}
```

Test 9

```
@Test
public void someTest(){
    Assertions.assertTrue(algorithm.isValid(3,2, 2,0));
    Assertions.assertTrue(algorithm.isValid(3,2, 1,1));
    Assertions.assertTrue(algorithm.isValid(3,2, 1,3));
    Assertions.assertTrue(algorithm.isValid(3,2, 2,4));

    Assertions.assertTrue(algorithm.isValid(3,2, 4,4));
    Assertions.assertTrue(algorithm.isValid(3,2, 5,3));
    Assertions.assertTrue(algorithm.isValid(3,2, 5,1));
    Assertions.assertTrue(algorithm.isValid(3,2, 4,0));

    Assertions.assertFalse(algorithm.isValid(3, 2, 0, 0));

    Assertions.assertFalse(algorithm.isValid(2, 0, 0, -1));

}
```

Test 10 (de los dos test que se verifican aqui, solo he conseguido que funcione el primero)

```
import { useState } from "react";
import useFetchState from "../util/useFetchState";
import { Table } from "reactstrap";
import tokenService from "../services/token.service";

const jwt = tokenService.getLocalAccessToken();



export default function MatchesListing() {
 const [message, setMessage] = useState(null);
 const [visible, setVisible] = useState(false);
 const [matches, setMatches] = useFetchState(
   [],
   `/api/v1/matches`,
   jwt,
   setMessage,
   setVisible
 );




const matchesList = matches.map((match) => {
   return (
     <tr key={match.id}>
       <td>{match.name}</td>
       <td>
         <ul>
           {match.rounds?.map((round) =>
             round.participants.map((user) => (
               <li key={user.username}>{user.username}</li>
             ))
           )}
         </ul>
       </td>
     </tr>
   );
 });
```

```
return (
   <>
     <div>
       <Table aria-label="matches" className="mt-4">
         <thead>
           <tr>
             <th width="10%">Name</th>
             <th width="10%">Participants</th>
           </tr>
         </thead>
         <tbody>{matchesList}</tbody>
       </Table>
     </div>
   </>
 );
}
```