

**RugbyApp**

Rui Garcia Jo˜ao Ferreira

Orientador: Jorge Martins

Relato´rio de progresso realizado no aˆmbito de Projeto e Semin´ario Licenciatura em Engenharia Informa´tica e de Computadores

Maio de 2020

Instituto Superior de Engenharia de Lisboa

**RugbyApp**

40539 Rui Miguel Marques Garcia

40913 Joa˜o Carlos M´aximo Ferreira

Orientador Jorge Manuel Rodrigues Martins Pia˜o

Relat´orio de progresso realizado no ˆambito de Projeto e Semina´rio Licenciatura em Engenharia Inform´atica e de Computadores

Maio de 2020

**Resumo**

O Rugby ´e um desporto que representa uma grande presenc¸a no quotidiano dos membros do nosso grupo, por sermos ou conhecermos praticantes ativos, e experienciarmos a maioria das vertentes deste desporto `há um longo per´ıodo de tempo. A falta de presença deste desporto no conceito geral da nossa cultura n˜ao o expo˜e a tanto apoio e auxilio como noutros desportos de grande renome, o que se constata de forma clara, ´ sem a necessidade de uma busca intensiva.

A ideia deste projeto nasceu da nossa pro´pria necessidade de criar uma aplica¸c˜ao que preencha essa lacuna. Com o foco prima´rio em trazer `as equipas deste desporto uma aplica¸cão virada para a organiza¸c˜ao e gest˜ao de informac¸a˜o dentro duma equipa, esperamos no fim apresentar uma ferramenta que consiga aglomerar os aspectos principais de uma equipa num u´nico sitio, e que provenha um apoio extra `a maioria das suas necessidades funcionais.

**Agradecimentos**

Agradecemos `as equipas t´ecnicas do Belas Rugby Clube e do Sporting Clube de Portugal pela disposi¸c˜ao e partilha de ideias e coment´arios, a fim de atenuar a ideia principal em que se baseia este projeto. Agradecemos tamb´em ao engenheiro Jorge Martins por se disponibilizar para ser o nosso orientador do projeto.

**´Indice**

1. Introduc¸˜ao 1

[1.1 Enquadramento . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1](#_TOC_250019)

[1.2 Objetivos e Funcionalidades . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1](#_TOC_250018)

[1.3 Organizac¸a˜o do documento . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2](#_TOC_250017)

1. Formulac¸˜ao do Problema 3

[2.1 Formulac¸a˜o . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3](#_TOC_250016)

[2.2 Propriedades B´asicas . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3](#_TOC_250015)

[2.2.1 Propriedades Principais . . . . . . . . . . . . . . . . . . . . . . . . . . 4](#_TOC_250014)

[2.2.2 Propriedades Secunda´rias . . . . . . . . . . . . . . . . . . . . . . . . . 5](#_TOC_250013)

[2.3 Especificac¸a˜o . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5](#_TOC_250012)

1. Aplicac¸˜ao Servidor 7
   1. [Introdu¸c˜ao e Estrutura da Aplica¸c˜ao Servidor . . . . . . . . . . . . . . . . . . 7](#_TOC_250011)

[3.1.1 *Model* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7](#_TOC_250010)

[3.1.2 *Repository* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8](#_TOC_250009)

[3.1.3 *Business* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9](#_TOC_250008)

[3.1.4 Controller 10](#_TOC_250007)

1. Aplicac¸˜ao Cliente 13
   1. [Introdu¸c˜ao e Estrutura da Aplica¸c˜ao Cliente 13](#_TOC_250006)
2. Testes 15
   1. [Aplica¸c˜ao Servidor 15](#_TOC_250005)
      1. [Testes de GET 16](#_TOC_250004)
      2. [Testes de POST 19](#_TOC_250003)
      3. [Testes de UPDATE 21](#_TOC_250002)
      4. [Testes de DELETE 24](#_TOC_250001)
3. Conclus˜ao 27
   1. [Recapitula¸c˜ao 27](#_TOC_250000)

**A Apˆendice A** **29**

**Cap´ıtulo 1**

**Introdu¸c˜ao**

Este projeto foi desenvolvido no ˆambito de Projeto e Semina´rio, no semestre de Ver˜ao de 2020 da Licenciatura em Engenharia Inform´atica e de Computadores. Este cap´tulo esta´ organizado em trˆes sec¸c˜oes que descrevem o enquadramento e objectivo do projeto, assim como a organiza¸c˜ao do documento.

# Enquadramento

Este projeto tem como tema´tica principal o desporto Rugby. Sendo uma atividade desportiva pouco reconhecida ou relevante no contexto da nossa cultura, ´e noto´ria a falta de ferramentas que providenciem suporte `a pratica desta atividade. Este problema foi apresentado aos diversos elementos do grupo devido a ligac¸o˜es interpessoais entre estes e o desporto, experienciando ativamente e lidando com este problema no pr´oprio quotidiano.

Apesar de existir um grupo de ferramentas que proporcionem uma melhor qualidade na organiza¸c˜ao e pra´tica desta modalidade, este foi considerado pelos estudantes como um grupo escasso e dispendioso, pelo que foi optado como objectivo deste projeto criar uma aplica¸cão que assente na ideia de ajudar ativamente equipas t´ecnicas desta modalidade desportiva em v´arios temas relevantes `a optimiza¸c˜ao e melhoria do desempenho da equipa ao longo da ´epoca desportiva.

# Objetivos e Funcionalidades

Esta secc¸˜ao aborda os objetivos e funcionalidades principais e secunda´rios.

E´ de notar que

entre a proposta de projeto inicial e o corrente relato´rio ocorreram diversas reunio˜es com as equipas t´ecnicas do Belas Rugby Clube e Sporting Clube de Portugal, pelo que ´e nota´vel alguma diversidade de objetivos entre ambos os documentos.

A ideia chave deste projeto ´e criar uma aplica¸c˜ao que seja capaz de recolher e analisar estatisticamente dados sobre o desempenho dos jogadores de uma equipa de Rugby, afim de monitorizar aspectos cr´ıticos que avaliem n˜ao s´o o estado individual de cada jogador, assim

como o estado atual de toda a equipa. Pressupo˜e-se que toda a informac¸a˜o recolhida consiga facilitar aspectos chaves do funcionamento de uma equipa desportiva, auxiliando desde a face t´actica do desporto (aspectos como a constituic¸a˜o da equipa, o plano t´actico, a optimiza¸cao de tema´ticas de treino), assim como a face menos t´ecnica de uma equipa (aspectos como a organiza¸cao de treinos e jogos, a facilidade de acesso a informac¸˜ao pertinente, entre outros). Pretende-se aglomerar todos estes aspectos numa u´nica aplica¸c˜ao, que ofere¸ca aos utiliza- dores, sendo eles atletas ou equipa t´ecnica, uma plataforma onde possam observar os dados aglomerados referentes a cada jogador em particular, os dados referentes a jogos concretos ao longo da ´epoca, aceder a planos de treinos f´ısicos propostos pela equipa t´ecnica, e obser- var uma linha temporal sobre todos os eventos futuros contextuais com a equipa. No que toca `a equipa t´ecnica, esta tamb´em tera´ a funcionalidade de gerar jogos, manusear jogadores em contexto destes jogos (conceitos como lista de convocados, jogadores titulares, jogado- res suplentes), assim como ter acesso a uma interface gr´afica onde seja poss´ıvel adicionar estat´ısticas aos atletas, oferecendo uma percepc¸a˜o detalhada do desempenho desse atleta no

jogo.

As funcionalidades s˜ao explicadas em mais detalhe na secc¸a˜o 2.2.

# Organizac¸˜ao do documento

O restante relato´rio encontra-se organizado da seguinte forma.

Cap´ıtulo 2 **Formulac¸˜ao do Problema**

Formulac¸a˜o e Contextualizac¸a˜o do Problema, Propriedades B´asicas e Especificac¸a˜o

Cap´ıtulo 3 **Aplicac¸˜ao Servidor**

Aspectos relacionados com a aplica¸c˜ao servidora, como abordagens, metodologias e detalhes de implementação¸.

Cap´ıtulo 4 **Aplicac¸˜ao Cliente**

Aspectos relacionados com a aplica¸c˜ao cliente, como abordagens, metodologias e detalhes de implementação¸.

Cap´ıtulo 5 **Testes**

Testes executados sobre as diversas vertentes do projeto.

Cap´ıtulo 6 **Concluso˜es**

Recapitulação das observa¸c˜oes e conclus˜oes importantes.

**Cap´ıtulo 2**

**Formula¸c˜ao do Problema**

Este cap´ıtulo esta´ organizado em trˆes secc¸o˜es, onde se descreve a formulac¸a˜o do problema e as suas propriedades, assim como as especificac¸o˜es.

# Formula¸c˜ao

Esta sec¸c˜ao aborda todos os aspetos referentes `a Formulac¸a˜o do Problema.

Tema: Aplica¸c˜ao de Suporte a Equipas de Rugby

Problema v1: As equipas t´ecnicas de Rugby tˆem ferramentas de suporte `a sua organiza¸c˜ao?

Que aspectos s˜ao necessa´rios implementar numa aplica¸c˜ao para garantir a sua utilidade?

Problema v2: Que aspectos s˜ao necessa´rios implementar numa aplica¸c˜ao para garantir

a cobertura da falta de ferramentas de apoio `as equipas t´ecnicas de Rugby?

A hipo´tese de resposta a esta pergunta foi adquirida da no¸c˜ao pessoal dos estudantes, como indiv´ıduos com ligac¸o˜es interpessoais com o desporto, e das ideias que resultaram do diálogo com as duas equipas referidas neste documento. Ap´os diversas reunio˜es com foco na recolha de ideias, foi atingida a hipo´tese de resposta cujos objetivos e funcionalidades esta˜o descritos na sec¸c˜ao 1.2. Apesar do problema apresentar algum teor subjectivo (equipas dis-tintas operam e organizam-se de formas distintas, e sentem necessidades distintas em factores distintos), foi poss´ıvel alcan¸car uma soluc¸a˜o que aglomera os fatores mais importantes para garantir a utilidade e a cobertura necessa´rias no contexto desta aplica¸c˜ao.

# Propriedades B´asicas

Esta sec¸c˜ao enumera as propriedades b´asicas da nossa soluc¸a˜o, separando-as em propriedades principais e propriedades secunda´rias.

## Propriedades Principais

A primeira sub-secc¸a˜o desta sec¸c˜ao lista todos os conceitos chave que se pretendem desenvol- ver como propriedades principais:

* + - 1. Conceito de Perfil de Atleta;
      2. Conceito de Perfil de Equipa T´ecnica;
      3. Conceito de Jogo;
      4. Conceito de Estat´ısticas de Jogo;
      5. Conceito de Treino;
      6. Conceito de Planos de Treinos F´ısicos;
      7. Conceito de Calenda´rio de Eventos;
      8. Conceito de Torneio;
      9. Conceito de Evento;

Estes conceitos refletem a estrutura da nossa aplicac¸a˜o.

A nossa aplica¸c˜ao ira´ implementar um perfil de Atleta, onde se pode observar a informac¸a˜o correspondente do atleta, como a idade, peso, altura, posi¸c˜oes, assim como as suas estat´ısticas ao longo da ´epoca. Tamb´em sera´ poss´ıvel observar uma lista dos jogos onde foi convocado, ligac¸o˜es para as suas estat´ısticas nos mesmos, e uma lista de treinos e eventos a que com- pareceu. A aplica¸c˜ao ira´ tamb´em implementar perfis dedicados aos integrantes das equipas t´ecnicas, para adicionar alguma coes˜ao sobre a informac¸a˜o global da equipa.

A nossa aplica¸c˜ao ira´ implementar um menu de jogo, com as estat´ısticas da equipa no contexto desse jogo, os jogadores convocados e titulares, o oponente, e coment´arios adicionais. A nossa aplica¸c˜ao ira´ implementar um menu de treinos, com as datas e locais de treinos,

a lista de comparecentes, e coment´arios adicionais. A lista de comparecentes ira´ conseguir diferenciar os atletas que compareceram como treinantes, os que compareceram sem participar no treino ou os que compareceram para outra atividade ligada ao treino, como treinos f´ısicos e de recupera¸c˜ao de leso˜es.

A nossa aplica¸c˜ao ira´ implementar um menu de planos de treino, onde a equipa t´ecnica podera´ fazer *upload* de planos de treino f´ısicos ou de gin´asio, e indicar as datas onde estes planos se devem concretizar e os atletas a que os planos se dirigem.

A nossa aplica¸c˜ao ira´ implementar um calenda´rio, onde ira˜o estar demonstrados todos os jogos, treinos, torneios ou outros eventos adicionados pela equipa t´ecnica e a sua respectiva data de concretizac¸a˜o.

## Propriedades Secund´arias

Esta ´e a segunda sub-secc¸a˜o desta sec¸c˜ao, que aponta os conceitos de algumas propriedades secunda´rias. Conforme a disponibilidade, s˜ao propriedades que podera˜o ser inseridas no contexto da aplica¸c˜ao, nomeadamente:

* + - 1. Conceito de Fisioterapeuta;
      2. Conceito de Les˜ao;
      3. Conceito de Campeonato;
      4. Conceito de Estat´ısticas Gra´ficas;
      5. Conceito de Exporta¸c˜ao de Dados;

Estes conceitos refletem a possibilidade de monitorizar e documentar leso˜es, e apresenta- las de forma organizada numa interface pro´pria para uso pelos fisioterapeutas, assim como de organizar os diversos jogos da ´epoca num campeonato com respectivas classificac¸o˜es. Tamb´em propo˜e a possibilidade de exportar dados em formatos de texto para serem consumidos por outros meios, assim como de representar visualmente as estat´ısticas dos jogos com auxilio gr´afico.

# Especificac¸˜ao

Esta sec¸c˜ao explicita as especificac¸o˜es da nossa aplica¸c˜ao.

A nossa aplica¸c˜ao ira´ ser dividida entre aplica¸c˜ao servidor e aplica¸c˜ao cliente.

A aplica¸c˜ao servidora ira´ ser programada em *Java* com uso da *Spring Boot framework*. A base de dados ira´ ser programada em *MySQL* com a ligac¸a˜o entre estes componentes feitos com o auxilio de *JPA - Java Persistence API*.

A aplica¸c˜ao cliente ira´ ser dividida em aplica¸c˜ao *Web* e aplica¸c˜ao *mobile*, ambas progra- madas em *TypeScript*. A vertente *Web* ira´ ser programada com o uso de *Angular framework*, e a vertente *mobile* com o uso de *IONIC framework*.

Entende-se que a maioria destes dom´ınios sejam familiares ao leitor.

**Cap´ıtulo 3**

**Aplicac¸˜ao Servidor**

Este cap´ıtulo vai apresentar a nossa soluc¸a˜o para o lado da aplica¸c˜ao servidor.

# Introdu¸c˜ao e Estrutura da Aplicac¸˜ao Servidor

A aplica¸co servidor ´e uma das duas partic¸o˜es da nossa aplica¸c˜ao.

E´ a partic¸a˜o onde se

encontra a base de dados, o modelo de dados, e todo o comportamento que os interliga um com o outro, assim como com a aplica¸c˜ao cliente, sejam estes leituras e escritas, algoritmos de pesquisa ou *routing*.

A partir das propriedades principais discutidas na sub-secc¸a˜o 2.2.1, foi poss´ıvel desenvolver a estrutura do nosso modelo de dados, de modo a ser mais percept´ıvel a maneira como os dados iriam ser guardados , as suas . No Apˆendice A pode-se observar a descric¸a˜o do modelo de dados.

Como descrito na secc¸˜ao 2.3, a aplicac¸a˜o servidor ira´ ser desenvolvida com uso da *Spring Boot framework* e de *JPA - Java Persistence API*. Dadas as funcionalidades acrescentadas destas *framework* e *API*, a nossa soluc¸a˜o separa a aplica¸c˜ao servidor em quatro camadas distintas:

* + 1. *Model*
    2. *Repository*
    3. *Business*
    4. *Controller*

As seguintes sub-secc¸o˜es explicam como cada camada funciona e como ´e que estas interagem entre si.

* + 1. ***Model***

A camada *Model*, referida nesta sub-secc¸a˜o como camada do Modelo, representa o modelo de dados. E´ aqui que encontramos todas as Entidades que estruturam o modelo de dados, assim como as relac¸o˜es entre elas.

Uma das caracter´ısticas chave do JPA ´e a possibilidade de criar classes com a anota¸cao

*@Entity* onde, mapeando os campos das entidades do modelo de dados diretamente, consegue- se gerar automaticamente a base de dados, e todas as conexo˜es necessa´rias entre esta e o modelo de dados.

Podemos observar no troc¸o de c´odigo seguinte, como exemplo, a classe *Event* inserida na camada do Modelo, que representa um Evento.

@Entity

@Table(name = "event")

@Data

public class Event {

@Id

@GeneratedValue(strategy = GenerationType.AUTO) private Long id;

@Column

private String name;

@Column

private String description;

@Column

private Date date;

@Column

private String local;

@Column

@OneToMany(mappedBy = "events") private List<Profile> profiles;

}

Replicando este conceito para todas as outras entidades, obtemos uma camada de Modelo onde esta˜o geradas todas as tabelas da base de dados, que constituem a camada do Modelo.

* + 1. ***Repository***

A camada *Repository*, referida nesta sub-secc¸a˜o como camada de Reposit´orio, representa os reposit´orios para cada entidade. Outra das caracter´ısticas chave do JPA ´e a habilidade de

criac¸a˜o de interfaces de reposit´orios associadas `as entidades do modelo, permitindo gerar persistˆencia na base de dados. Quando a aplica¸c˜ao interage com os reposit´orios (atrav´es da chamada a m´etodos de *queries*), o JPA gera as ligac¸o˜es `a base de dados e garante a comunicac¸a˜o entre o modelo f´ısico e o modelo de dados.

Podemos observar no troc¸o de c´odigo seguinte, como exemplo, a interface *EventRepository*

inserida na camada do Reposit´orio.

public interface EventRepository extends CrudRepository<Event, Long> { List<Event> findByDate(Date date);

}

Ao estender da interface *CrudRepository*, j´a implementada na biblioteca do JPA, as nossas classes de reposit´orio herdam m´etodos para trabalhar com a persistˆencia dos nossos objetos (neste caso, do *Event* ), atrav´es de opera¸c˜oes *Create* , *Read* , *Update* e *Delete*. Conforme a necessidade e o contexto, ´e poss´ıvel adicionar outras *querries* (*findByDate*) diretamente a estas interfaces, sem a necessidade de as implementar.

O agrupamento de todos os reposit´orios de todas as nossas entidades constituem a nossa camada de Reposit´orio.

* + 1. ***Business***

A camada *Business*, referida nesta sub-secc¸a˜o como camada de Neg´ocio, representa todos os

comportamentos referentes ao nosso modelo de nego´cios.

E´ nesta camada que se encontra

toda a algoritmia dedicada aos comportamentos da aplica¸c˜ao. Foram geradas classes *Business* para cada entidade, que cont´em comportamentos relacionados com procura e verificac¸a˜o de recursos. Esta camada liga diretamente aos reposit´orios.

Podemos observar no troc¸o de c´odigo seguinte, como exemplo, a classe *EventBusiness*

inserida na camada de Neg´ocio.

@Component

public class EventBusiness {

@Autowired

EventRepository eventRepository;

public Iterable<Event> findAllEvents(){ return eventRepository.findAll();

}

public Long postEvent(Event event){

return eventRepository.save(event).getId();

}

public Event findEventById(Long id){ return eventRepository.findById(id)

.orElseThrow(()-> new ResourceNotFoundException("Event", "Id", id));

}

public Long updateEvent(Event event) eventRepository.findById(event.getId())

.orElseThrow(()-> new ResourceNotFoundException("Event", "Id", event.getId())); return eventRepository.save(event).getId();

}

public void deleteEvent(Event event){ eventRepository.findById(event.getId())

.orElseThrow(() -> new ResourceNotFoundException("Event", "Id", event.getId())); eventRepository.delete(event);

}

}

A anota¸c˜ao *@AutoWired* garante a injec¸a˜o do *EventRepository* quando a classe *Event- Business* ´e criada. A anota¸c˜ao *@Component* permite que as classes sejam injetadas com

*@AutoWired*. Cada um destes m´etodos tem uma interac¸a˜o diferente com o reposit´orio, e nos casos justificados, faz a verificac¸a˜o da existˆencia do objeto no reposit´orio antes de o alterar/remover/retornar.

Este modelo de nego´cios garante a comunicac¸a˜o entre as camadas de controlo e reposit´orio, servindo de camada interm´edia onde ´e feita a verificac¸a˜o dos objetos antes de serem feitas altera¸c˜oes persistentes `a base de dados, e contem um comportamento que sera´ incremental ao longo da realizac¸˜ao do projeto.

### Controller

A camada *Controller*, referida nesta sub-secc¸a˜o como camada de Controlo, representa todo

o *rounting* do exterior para a aplica¸c˜ao servidor.

E´ a camada que gera todos os *endpoints*,

assim como os m´etodos associados a estes *endpoints*.

Podemos observar no troc¸o de c´odigo seguinte, como exemplo, a classe *EventController*

inserida na camada de Controlo.

@RestController()

@RequestMapping("/event")

public class EventController {

private static final Logger logger = LoggerFactory.getLogger(RugbyApplication.class);

@Autowired

EventBusiness eventBusiness;

@RequestMapping("event/all")

public Iterable<Event> findAllEvents(){ logger.info("On method GET event/all"); return eventBusiness.findAllEvents();

}

@GetMapping("/findById/{id}")

public Event findEventById(@PathVariable Long id){ logger.info("On method GET event/findById/{id} with id: "+ id); return eventBusiness.findEventById(id);

}

@PostMapping("/post")

public Long postEvent(@RequestBody Event event){ logger.info("On method POST event/post");

return eventBusiness.postEvent(event);

}

@PutMapping("/update")

public Long putEvent(@RequestBody Event event){ logger.info("On method PUT event/update"); return eventBusiness.updateEvent(event);

}

@DeleteMapping("/delete")

public ResponseEntity<?> deleteStats(@RequestBody Event event){ logger.info("On method GET event/all"); eventBusiness.deleteEvent(event);

return ResponseEntity.ok().build();

}

}

A anota¸c˜ao *RestController* serve para implementar classes de controlo, que cont´em m´etodos capazes de processar pedidos HTTP, ao mesmo tempo que serializa os objetos de retorno des- tes m´etodos para *HttpResponse*. Ou seja, todos os m´etodos desta classe s˜ao mapeados para um *endpoint* diferente e processam os pedidos para esse *endpoint*. A anota¸c˜ao *@RequestMap- ping* recebe os paraˆmetros de mapeamento, podendo especificar-se o *endpoint* que o m´etodo ou classe de controlo v˜ao processar, assim como outros paraˆmetros.

E´ de salientar que

*@GetMapping* corresponde a *@RequestMapping(method = RequestMethod.GET)*

*@PostMapping* corresponde a *@RequestMapping(method = RequestMethod.POST)*

*@PutMapping* corresponde a *@RequestMapping(method = RequestMethod.PUT)*

*@DeleteMapping* corresponde a *@RequestMapping(method = RequestMethod.DELETE)*

Podemos ent˜ao observar que todos os pedidos para o caminho sera˜o por esta classe, onde cada m´etodo de cada pedido ´e processado num m´etodo da classe.

Ap´os replicar este comportamento para as classes das diversas entidades, obtemos um *Router* completo para todos os *endpoints* da nossa aplica¸c˜ao. Tamb´em esta camada tem comportamento que sera´ incremental ao longo do desenvolvimento do projeto.

**Cap´ıtulo 4**

**Aplicac¸˜ao Cliente**

Este cap´ıtulo vai apresentar a nossa soluc¸a˜o para o lado da aplica¸c˜ao cliente.

# 4.1 Introdu¸c˜ao e Estrutura da Aplicac¸˜ao Cliente

A aplica¸c˜ao cliente ´e a segunda partic¸a˜o da nossa aplica¸c˜ao. E´ a partic¸a˜o onde se encontra a interface de utilizador, pain´eis de controlo e alguma l´ogica de nego´cio adicional.

No estado atual do nosso projeto, a aplica¸c˜ao cliente ainda n˜ao apresenta grande desen- volvimento. Para al´em do sub-projeto gerado em *Angular*, foram geradas as classes corres- pondentes `as Entidades da aplica¸c˜ao servidor em *TypeScript*.

Podemos observar no troc¸o de c´odigo seguinte, como exemplo, a classe *Event.ts* inserida no *package* de classes da nossa Aplica¸c˜ao Cliente.

import {Profile} from ’./profile’;

export class Event { constructor(

private id?: number, private name?: string,

private description?: string, private date?: Date,

private local?: string, private profiles?: Profile[]

) {

this.id = id ? id : 0;

this.description = description ? description : ’’; this.date = date ? date : new Date(0);

this.local = local ? local : ’’; this.name = name ? name : ’’;

this.profiles = profiles ? profiles : [];}}

Um dos aspetos principais a salientar na implementac¸a˜o dos construtores das Entidades na aplica¸c˜ao cliente ´e a questa˜o das propriedades poderem ser *nullable*. Organizando os construtores para que todas as propriedades sejam *nullable* enquanto se faz a verificac¸a˜o no corpo do construtor para a ausˆencia destas propriedades, permite-se construir objetos atribuindo valores padra˜o a todas as propriedades que n˜ao existem na altura da criac¸a˜o. Este detalhe de implementac¸a˜o ajuda a gerar objetos vazios sem os problemas que ocorrem frequentemente na manipulac¸a˜o de valores *null*.

**Cap´ıtulo 5**

**Testes**

Este cap´ıtulo aborda os testes executados no projeto.

# Aplicac¸˜ao Servidor

Nesta fase do projeto, os testes feitos `a parti¸c˜ao da aplica¸c˜ao servidor foram executados sem a participac¸a˜o da aplica¸c˜ao cliente.

Para a execuc¸˜ao destes testes, foram usadas duas ferramentas distintas:

*XAMMP free open-source cross-platform web server solution stack* para criar um servidor de testes.

*Postman* API cliente para gerar pedidos HTTP.

No ficheiro *apliccation.properties* existente na *framework Spring* foram escritas as propri- edades essenciais para garantir a ligac¸a˜o entre a aplica¸c˜ao e o servidor *XAMMP*.

spring.jpa.hibernate.ddl-auto=update spring.datasource.url=jdbc:mysql://${MYSQL\_HOST:localhost}:3306/test spring.datasource.username=root

spring.datasource.password=

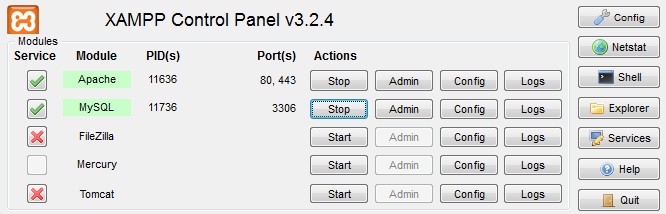


Figura 5.1: Figura do painel de controlo do *XAMMP*.

As seguintes sub-secc¸o˜es demonstram os v´arios testes aos *endpoints* da classe **Evento**.

## Testes de GET

Os testes feitos aos *endpoints* de *GET* foram repetidos ao longo dos testes dos outros m´etodos para fins de observa¸c˜ao de resultados, pelo que nesta sub-secc¸a˜o apenas se observam dois tes- tes realizados antes da gera¸c˜ao de informac¸a˜o de teste.

Teste : *GET ALL*

*Endpoint* : http://localhost:8080/event/all

*Body* : Nenhum

Resultado : Lista de objetos *JSON* de todos os Eventos.

*Status* : *200 OK*

Nota : Para efeitos de teste, foi colocado previamente um Evento para observa¸c˜ao.

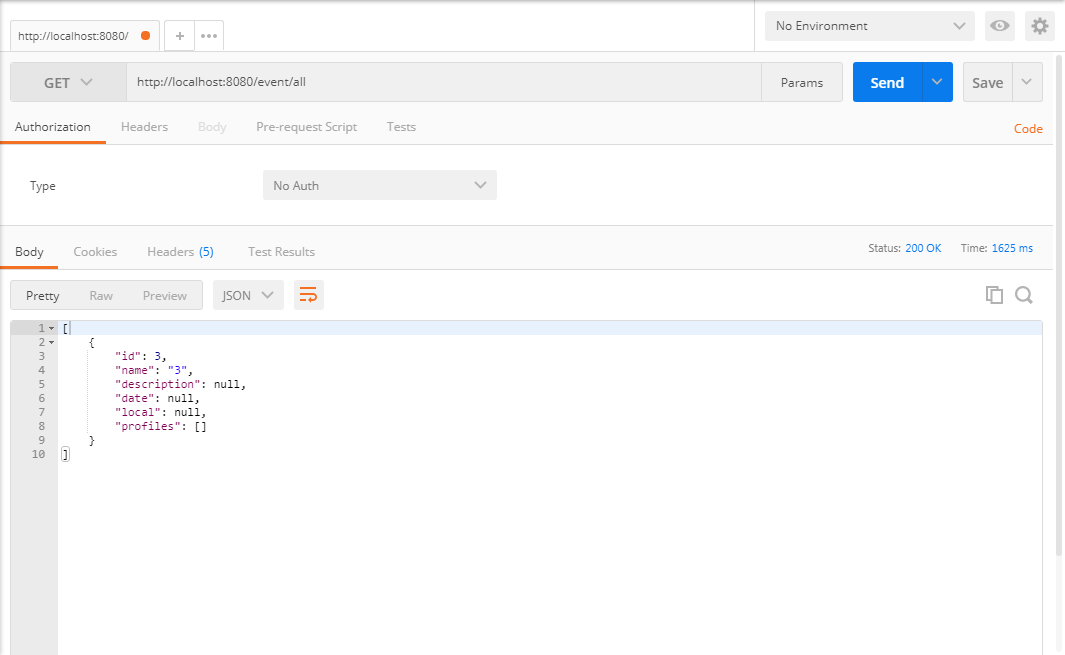


Figura 5.2: Resultado observ´avel no *Postman*.

Teste : *GET BY EXISTENT ID*

*Endpoint* : http://localhost:8080/event/findById/3

*Body* : Nenhum

Resultado : Objeto *JSON* do evento com id 3.

*Status* : *200 OK*

Nota : Para efeitos de teste, foi colocado previamente um Evento com id 3 para observa¸c˜ao.

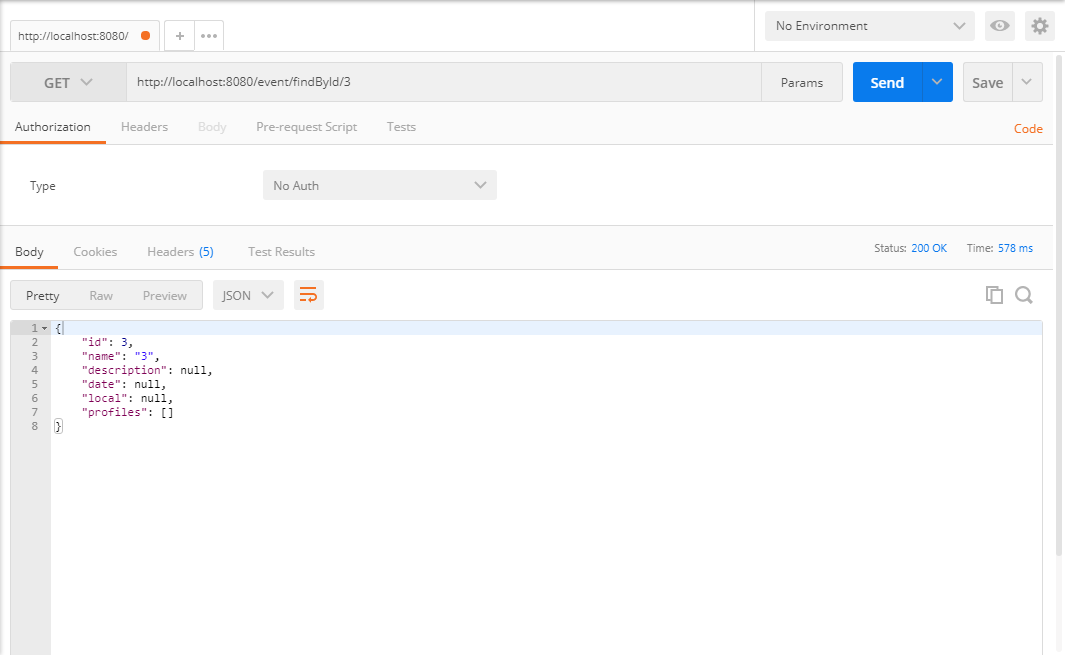


Figura 5.3: Resultado observ´avel no *Postman*.

Teste : *GET BY NONEXISTENT ID*

*Endpoint* : http://localhost:8080/event/findById/4

*Body* : Nenhum Resultado : Erro

*Status* : *404 NOT FOUND*

Nota : Para efeitos de teste, foi escolhido um id que n˜ao existe na base de dados.

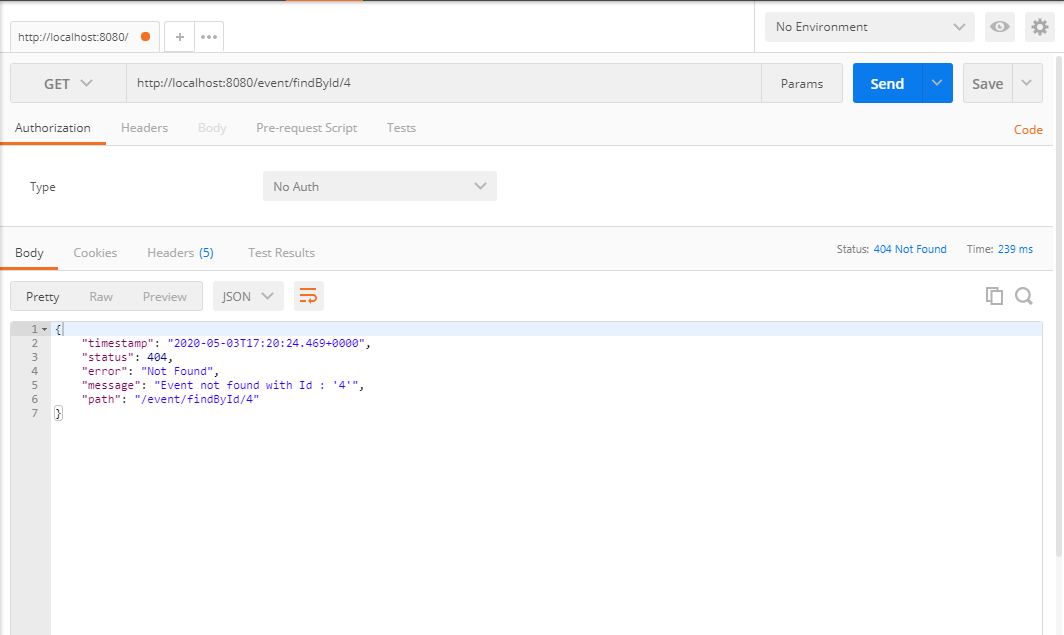


Figura 5.4: Resultado observ´avel no *Postman*.

## Testes de POST

Os testes feitos ao *endpoints* de *POST* s˜ao executados com um objecto *JSON* enviado no corpo do pedido. O id n˜ao ´e enviado juntamente com o resto da informac¸a˜o do objeto, pois

´e um campo gerado automaticamente e ´e retornado no corpo da resposta. Teste : *POST*

*Endpoint* : http://localhost:8080/event/post

*Body* : *{*

”name”: ”TestEvent”,

”description”: ”Description Of TestEvent”, ”date”: ”2020-05-04T00:00:00.000Z”,

”local”: ”Portugal”,

”profiles”: []

*}*

Resultado : id do objeto criado.

*Status* : *200 OK*

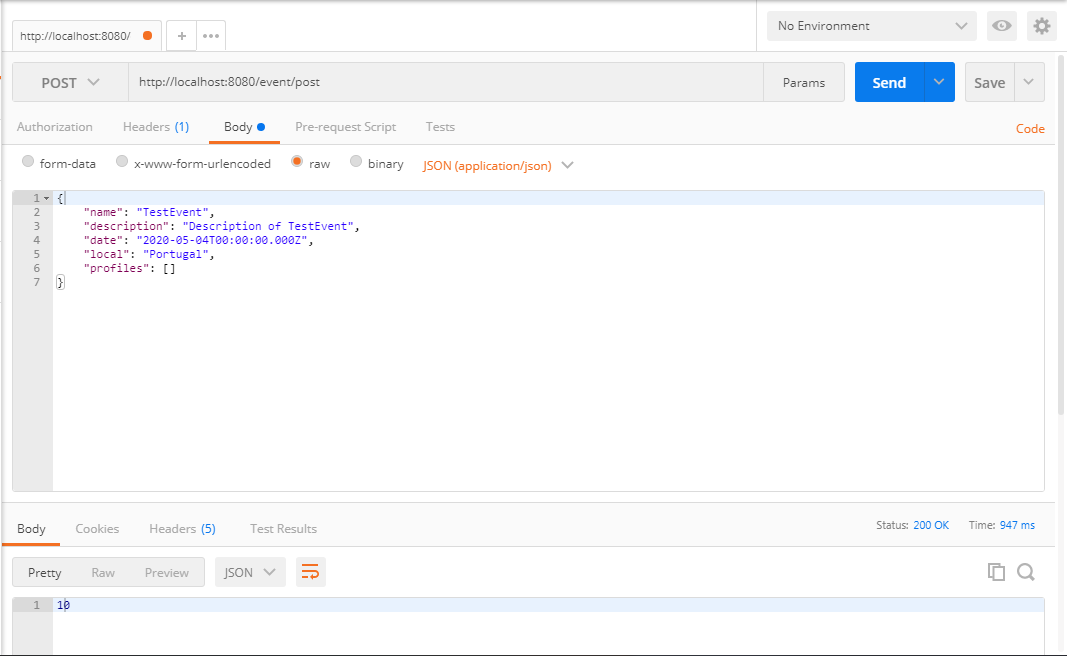


Figura 5.5: Resultado observ´avel no *Postman*.

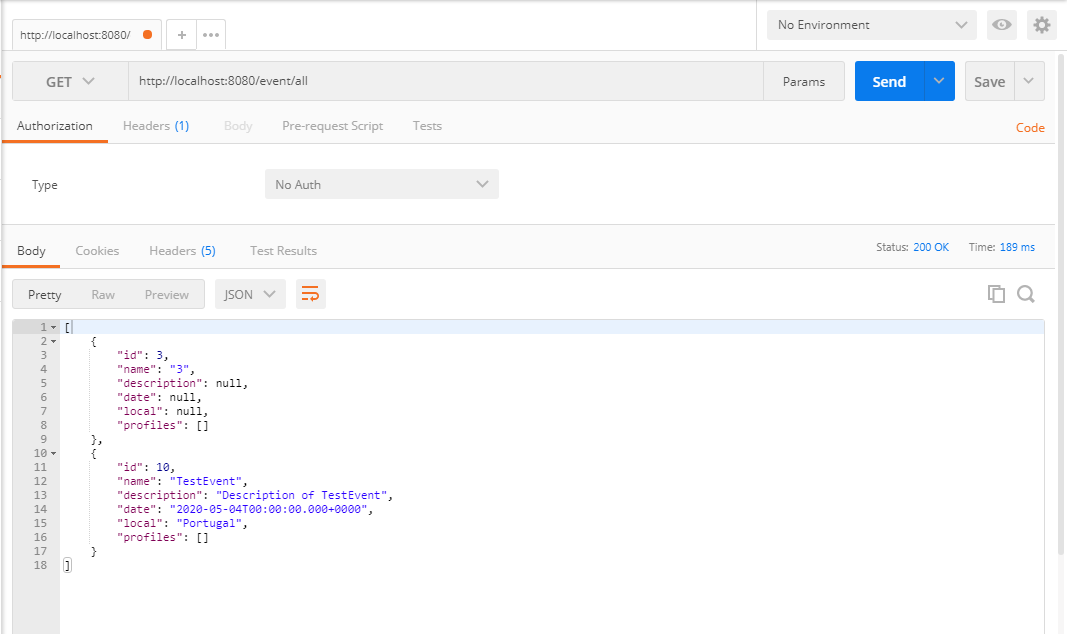


Figura 5.6: Resultado de um *GET ALL* depois do *POST* observ´avel no *Postman*.

## Testes de UPDATE

Os testes feitos ao *endpoints* de *PUT* s˜ao executados com um objecto *JSON* enviado no corpo do pedido com o id do objeto a ser alterado, assim como os campos a alterar. O id continua a ser retornado no corpo da resposta.

Teste : *UPDATE EXISTENT OBJECT*

*Endpoint* : http://localhost:8080/event/update

*Body* : *{*

”id”: 10,

”name”: ”TestEventUpdated”,

”description”: ”Updated Description Of TestEvent”, ”date”: ”2020-05-05T00:00:00.000Z”,

”local”: ”Spain”,

”profiles”: []

*}*

Resultado : id do objeto alterado.

*Status* : *200 OK*

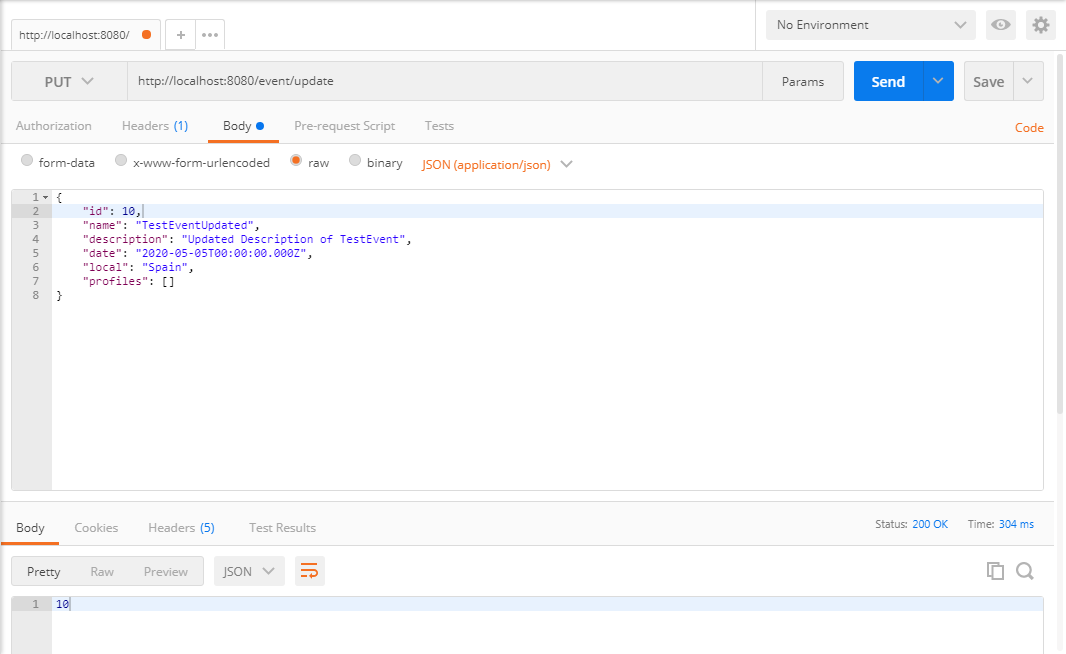


Figura 5.7: Resultado observ´avel no *Postman*.

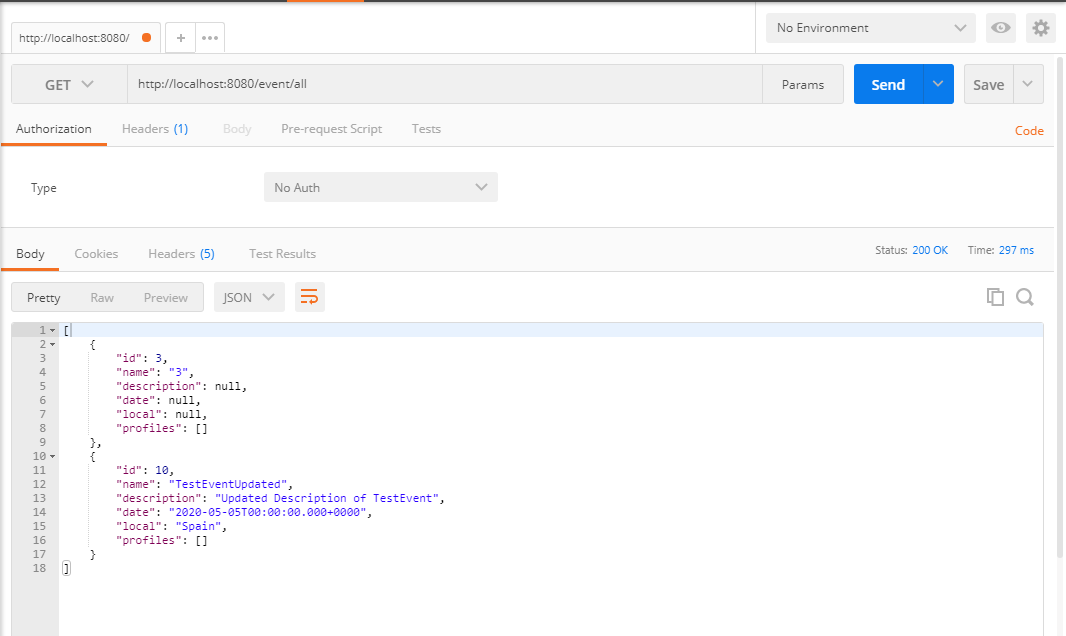


Figura 5.8: Resultado de um *GET ALL* depois do *PUT* observ´avel no *Postman*.

Teste : *UPDATE NONEXISTENT OBJECT*

*Endpoint* : http://localhost:8080/event/update

*Body* : *{*

”id”: 9

”name”: ”TestEventUpdated”,

”description”: ”Updated Description Of TestEvent”, ”date”: ”2020-05-05T00:00:00.000Z”,

”local”: ”Spain”,

”profiles”: []

*}*

Resultado : Erro.

*Status* : *404 NOT FOUND*

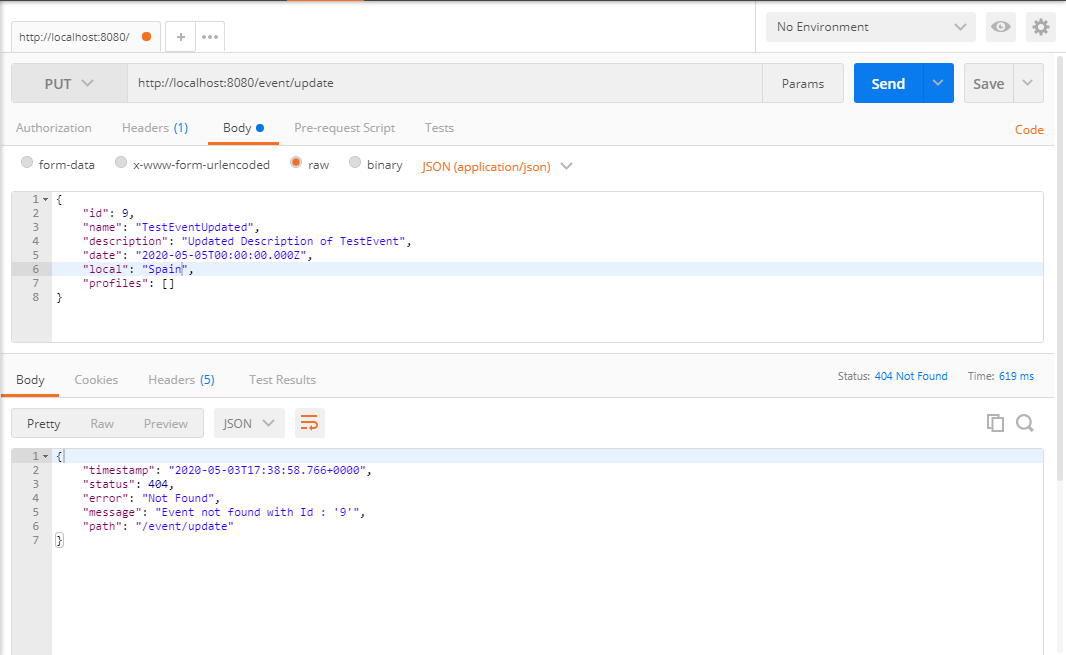


Figura 5.9: Resultado observ´avel no *Postman*.

## Testes de DELETE

Os testes feitos ao *endpoints* de *DELETE* s˜ao executados com um objecto *JSON* enviado no corpo do pedido com o id do objeto a ser apagado.

Teste : *DELETE EXISTENT OBJECT*

*Endpoint* : http://localhost:8080/event/delete

*Body* : *{*

”id”: 10

*}*

Resultado : Nenhum

*Status* : *200 OK*

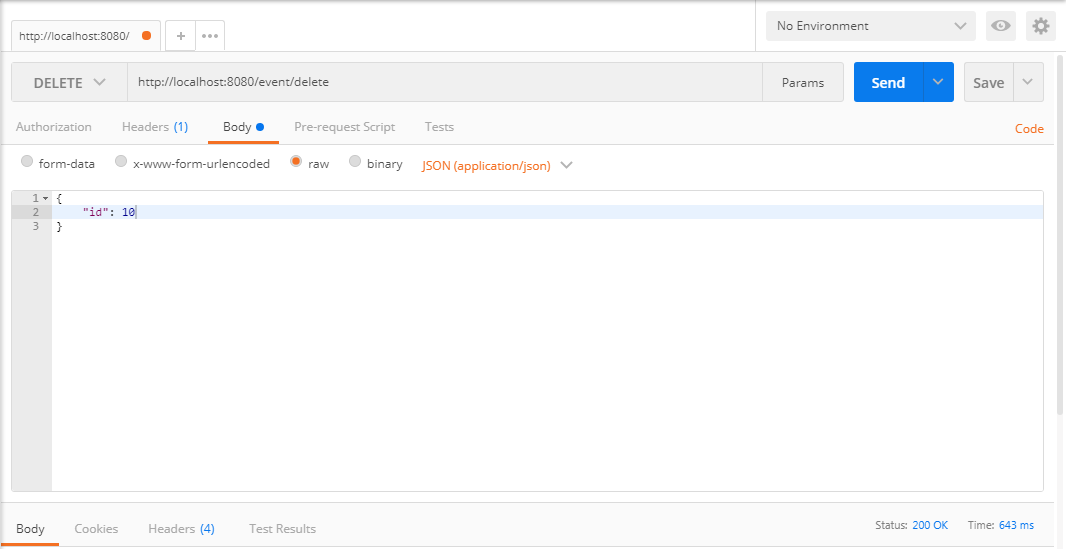


Figura 5.10: Resultado observ´avel no *Postman*.

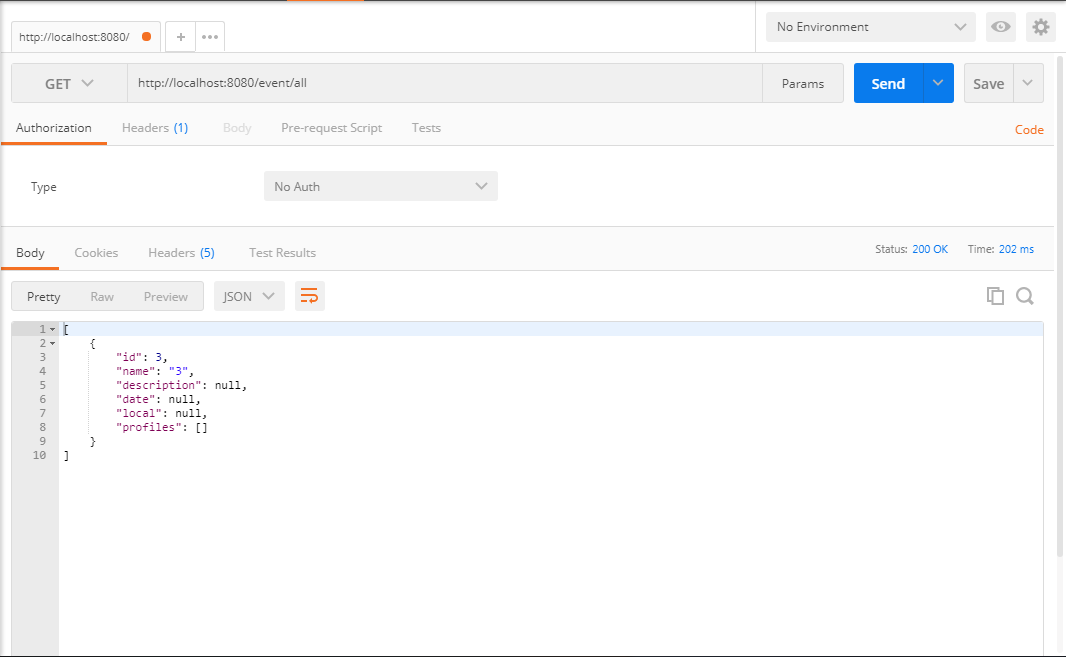


Figura 5.11: Resultado de um *GET ALL* depois do *DELETE* observ´avel no *Postman*.

Teste : *DELETE NONEXISTENT OBJECT*

*Endpoint* : http://localhost:8080/event/delete

*Body* : *{*

”id”: 9

*}*

Resultado : Erro

*Status* : *404 NOT FOUND*

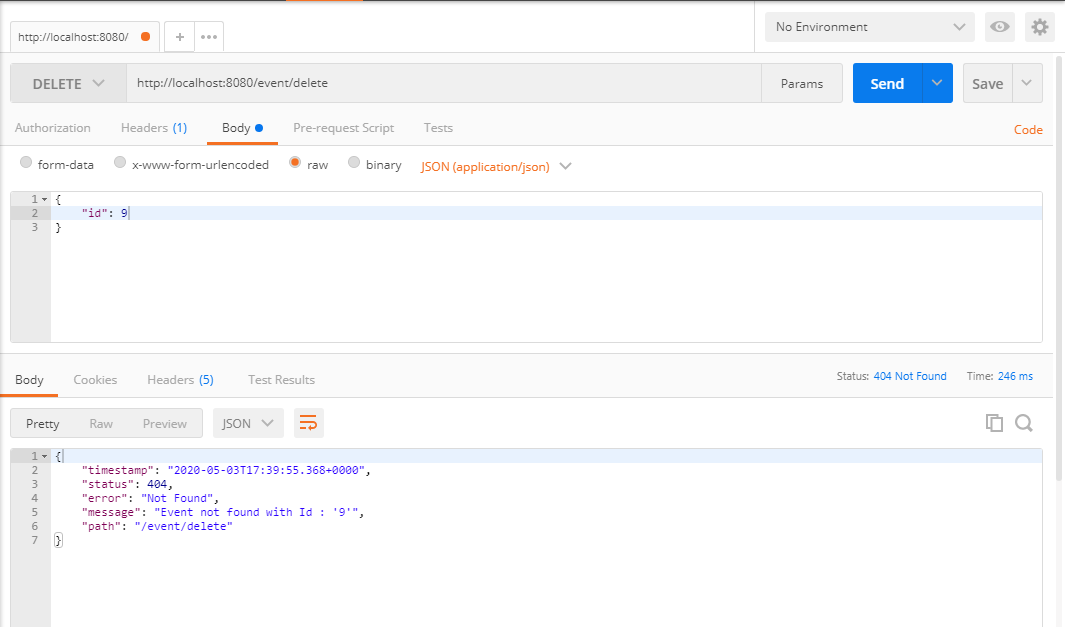


Figura 5.12: Resultado observ´avel no *Postman*.

**Cap´ıtulo 6**

**Conclus˜ao**

Este cap´ıtulo descreve as concluso˜es que foram adquiridas do trabalho feito at´e ao momento.

# Recapitulac¸˜ao

Como referido na secc¸a˜o 2.1, foi poss´ıvel, atrav´es de reunio˜es com clubes deste desporto, definir as propriedades principais e secunda´rias da nossa aplica¸c˜ao. Ap´os estas propriedades estarem definidas e estruturadas, foi gerado o modelo de entidades onde assenta a nossa aplica¸c˜ao (demonstrado no Apˆendice A).

Todo o trabalho feito at´e agora foi maioritariamente do lado da aplica¸c˜ao servidor. Como referido no cap´ıtulo 3, foram definidas todas as estruturas, tanto em termos de entidades e informac¸a˜o persistente, como em termos do formato em que a aplica¸c˜ao no geral ira´ ter acesso a esta informac¸a˜o. Distribuindo os focos principais nas camadas de Modelo, Repo- sito´rio, Neg´ocio e Controlador, podemos de uma forma organizada separar todo o processo que envolve o caminho desde o *browser* at´e `a base de dados. Com o aux´ılio das ferramentas apresentadas na secc¸a˜o 2.3, conseguimos organizar toda esta partic¸a˜o e mantˆe-la consistente.

Do lado da aplicac¸a˜o cliente, como apresentado no cap´ıtulo 4, foram apenas geradas as classes que correspondem `as classes de Entidades da aplica¸c˜ao servidor.

Todos os testes feitos foram ao lado da aplica¸c˜ao servidor. Podemos observar o compor- tamento dos *endpoints* e a persistˆencia dos dados na base de dados.

**Apˆendice A**

**Apˆendice A**

O Apˆendice A contem a lista de todas as entidades e as suas propriedades.

A lista de entidades representa as propriedades que s˜ao tipos primitivos pelo seu nome, as propriedades que referem associa¸c˜oes de um para um pelo nome da entidade a que a associac¸a˜o se refere, e as propriedades que referem associac¸o˜es de um para muitos por uma lista de entidades a que a associa¸c˜ao se refere.

Lista de Entidades

* + 1. *Athlete* (id,height,weight,athleteNumber,comment,*Profile*,List*<*Practice*>*,List*<TrainingSchedule>*, List*<Game>*,List*<AthleteGameStats>*)
    2. *AthleteGameStats* (id,*Athlete*,*Stats*,*Game*)
    3. *Event* (id,name,description,date,local,List*<Profile>*)
    4. *Game* (id,date,local,comment,*Opponent*,List*<Athletee >*,List*<AthleteGameStats>*)
    5. *Opponent* (id,name,photo)
    6. *Practice* (id,date,local,comment,List*<Athlete>*)
    7. *Profile* (id,name,birth,address,mail,phone,photo,List*<Event >*)
    8. *Staff* (id,staffNumber,*Profile*,*StaffType*)
    9. *StaffType*(id,name)
    10. *Stats* (id,errors,fouls,turnOvers,yellowCards,redCards,tries,mauls,playingTime,*Tackle*,*Mellee*,

*ConvertionKick*,*GoalKick*,*DropKick*,*OffSide*,*LineOut*,List*<AthleteGameStats>*)

* + 1. *Tackle* (tackleHits,tackleMiss)
    2. *Mellee* (melleeHits,melleeMiss)
    3. *ConvertionKick* (convertionKickHits,convertionKickMiss)
    4. *GoalKick* (goalkickHits,goalkickMiss)
    5. *DropKick* (dropKickHits,dropkickMiss)
    6. *OffSideKick* (offsideHits,offsideMiss)
    7. *LineOut* (lineOutHits,lineOutMiss)
    8. *Tournament* (id,classification,comment)
    9. *TrainingSchedule* (id,description,link,date,List*<Athlete>*)

Todas estas entidades foram implementadas diretamente na camada do Modelo.