Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

ÉRION RICARDO BARASUOL

MONGODB UMA BASE DE DADOS ORIENTADA A DOCUMENTOS QUE UTILIZA ORIENTACAO A OBJETOS

Assis
2012

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

ÉRION RICARDO BARASUOL

MONGODB UMA BASE DE DADOS ORIENTADA A DOCUMENTOS QUE UTILIZA ORIENTACAO A OBJETOS

Trabalho de conclusão de curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisição do Curso de Graduação.

Orientador : Me. Douglas Sanches da Cunha.

Assis
2012

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

FICHA CATALOGRÁFICA

BARASUOL ÉRION, Ricardo
MongoDB uma base de dados orientada a documentos que utiliza orientação a objetos / Érion Ricardo Barasuol. Fundação Educacional do Município de Assis -- Assis, 2012.
35.

Orientador. Me Douglas Sanches Cunha.
Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis.
1.NoSQL. 2.MongoDB. 3.Java. 4.Open Source. 5.Mophia. 6.JSON.

CDD: 001.6
Biblioteca da FEMA

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

ÉRION RICARDO BARASUOL

MONGODB UMA BASE DE DADOS ORIENTADA A DOCUMENTOS QUE UTILIZA ORIENTACAO A OBJETOS

Trabalho de conclusão de curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisição do Curso de Graduação.

Orientador : Me. Douglas Sanches da Cunha.

Analisador: Me. Fábio Eder Cardoso.

Assis
2012

Avenida Getúlio Vargas, 1200 – Vila Nova Santana – Assis/SP – CEP 19807- 634
Fone/Fax (18) 3302-1055 – Homepage: www.fema.edu.br

DEDICATÓRIA

Dedico este trabalho aos meus pais, meu irmão minha namorada, e a todos meus amigos que me ajudaram e apoiaram.

AGRADECIMENTOS

Á Deus que me permitiu chegar até o final, adquirindo conhecimento necessário para enfrentar todas as dificuldades e obstáculos encontrados no caminho.

Ao meu orientador e amigo Douglas Sanches da Cunha, que me incentivou e ensinou a transformar estudo e esforço em conhecimento.

A minha namorada e todos os meus familiares e amigos que estiveram ao me lado nos momentos mais difíceis desta jornada, e a todos que acreditaram na execução deste trabalho.

RESUMO

Este trabalho de pesquisa tem por finalidade apresentar um novo conceito de banco de dados não relacional, que está se tornando cada vez mais popular no mercado de trabalho chamado de NoSQL. MongoDB que por ser do tipo NoSQL, possui características que o difere dos bancos de dados tradicionais, como a ausência de *schemas* fixos, que possibilita um ciclo de desenvolvimento ágil, voltado para implementação do *softwares* e deixando sua estrutura por conta das relações entre as classe já que sua manipulação de dados e totalmente orientada a objetos.

Palavras – chave: NoSQL. MongoDB. Java. Open Source. Mophia. JSON.

ABSTRACT

That research work is intended to present a new concept of no relational database that is growing in the job market called NoSQL. MongoDB because it is NoSQL, has features that differ from traditional database's, such as the absence of fixed schemas, which enables an agile development cycle focused on software implementation, and leaving the structure of relationship a task between class, it's because your data manipulation is fully oriented object.

Keywords: NoSQL. MongoDB. Java. Open Source. Mophia. JSON.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

1 – Introdução

Segundo Tiwari (2011) os tipos de dados que são manipulados pelos usuários, tiveram um aumento considerável gerando grandes bases de dados, sua manipulação, análise e armazenamento já não são na mesma escala de alguns anos atrás. Essas grandes bases de dados são chamadas de *bigdata*, e trás novos desafios quanto à armazenagem e manipulação deles. Esse novo desafio levou as pessoas a questionarem, se realmente estava correto continuar manipulando o tradicional modelo de dados relacional. A missão de solucionar esses problemas levou ao surgimento de uma nova classe de bancos de dados, que consiste em *column-oriented data stores*, *key/value pair databases*, e *document databases*, todos esses são identificados como *NoSQL*.

Segundo Tiwari (2011) durante os últimos cinco anos *NoSQL* e os conceitos de gerenciamento de grandes dados tem sido generalizado e utilizado por novas companhias como *Facebook* ("Rede Social"), *Netflix* ("Video Locadora Virtual"), *Yahoo*, *Ebay* ("Empresa de Comércio Eletrônico"), *IBM* ("Desenvolvedora de *Softwares*"), e muitas outras. Além disso, muitas dessas novas empresas contribuem para o crescimento do *NoSQL* e produtos *open-source*.

MongoDB é um banco que não utiliza o modelo relacional, que todos estão acostumados a utilizar, justamente por pertencer à classe dos bancos *NoSQL*. Ferramentas, bases de dados e estruturas *NoSQL*, como será abordado no capítulo 2, não são muito comentadas, mesmo sendo utilizadas em grande escala hoje em dia. Portanto, o objetivo deste trabalho é apresentar como bases de dados não relacionais em especial MongoDB, podem ser muito mais fáceis de manipular quanto aparenta, como por exemplo, na manipulação extensiva e exaustiva dos dados, fica claro que não é um competidor dos modelos relacionais, mas uma alternativa de evolução em nossa tecnologia. E como comprovação desse estudo aplicado ao banco específico, Mongodb, será inserido no trabalho uma aplicação prática implementada em linguagem Java.·.

Segundo Banker (2012) MongoDB proporciona um desenvolvimento mais mesclado do que muitos modelos relacionais, um dos motivos pelo jornal *The New York Times* o ter aderido. Uma das razões mais obvia para esse desenvolvimento ágil é o fato dele não possuir um *schema* fixo, permitindo inserir informações sem a necessidade de seguir um padrão para todos. Ele muitas vezes complementa projetos em ciclos de desenvolvimento ágil, em equipes. O que torna esse banco de dados um objeto de estudo de suma importância para a classe de desenvolvedores de modelos *NoSQL*.

Puggle, Membrey e Peter (2012) comentam que MondoDB é um banco de dados que não possui tabelas, *squemas*, *SQL* nem linhas como na citação acima de Banker (2012). Este banco não possui transações *joins*, *foreign keys* e muitas das funções utilizadas em modelos relacionais. MongoDB é um banco muito diferente dos utilizados, principalmente para programadores que são acostumados a desenvolver modelos que utilizam *SQL*.

Banker (2012) comenta que esse banco de dados foi desenvolvido pensando em aplicações que utilizam infraestrutura da internet, tudo em um modelo de dados e estratégia construído para suportar grande volume de leitura e escrita. Possui habilidade de ser facilmente escalável com tolerância a falhas, e seu desempenho é alto, tanto para dezenas, quanto para apenas uma base de dados, pois trabalha facilmente com *clustering* de informações o que proporciona alto desempenho de suas funções.

O trabalho será organizado no formato de capítulos. O capítulo 2 será sobre a propagação do *NoSQL* pelas grandes empresas, as principais características da base de dados do MongoDB, e alguns códigos-fonte para administração dos dados na prática. No próximo capítulo será demonstrado na prática como MongoDB interage com Java, referente à camada de persistência dos dados no banco, como conexão, operações de escrita e retorno de informações. Por fim, mas não menos importante, no capitulo quatro será feito a documentação completa da implementação de um sistema de consultas e reservas, utilizando MongoDB como

base de dados e linguagem Java, cujo mesmo já possui uma implementação em banco de dados MySQL.

2 – Banco de dados MongoDB

Segundo Tiwari (2011, p. 4):

> "*NoSQL* não é um conceito novo, de fato modelos não relacionais já existiam desde que os primeiros computadores foram inventados. Bases de dados não relacionais prosperaram através do advento dos *mainframes* e tem existido especificamente em domínios específicos".

NoSQL que significa *Not Only SQL* é o termo utilizado para o leque que engloba todos os bancos de dados e sistemas de armazenamento que não seguem o tradicional modelo normalizado RDBMS. Segundo Tiwari, Shashank 2011, *NoSQL* não é apenas um produto ou uma tecnologia em específico, e sim representa uma classe de produtos e coleções de diversos tipos, e às vezes relacionada com conceitos de armazenamento e manipulação.

Tiwari (2011) comenta que, nem todas base de dados *NoSQL* são similares e criadas para resolver o mesmo problema, mas é essencial entender, qual se encaixa à determinada situação. Bases de dados *NoSQL* podem ser comparadas com a rápida evolução das linguagens de programação que nos trazem uma determinada linguagem para determina tarefa. Consequentemente os programadores necessitam conhecer mais do que uma linguagem em seu repertório. Portanto, torna-se uma ótima estratégia hoje conhecer e adotar mais de um banco de dados em nosso repertório assim como as linguagens comparadas aos programadores.

Bases de dados NoSQL possuem tamanhos, e formas diferentes e muitas soluções para eventuais problemas podem ser facilmente adequados a sua características. Portanto, *NoSQL* nos dias de hoje não passa de apenas uma nova adoção, que adequa-se ao fato de estarmos ligados diretamente às aplicações que necessitam de alto desempenho na web.

Segundo Tiwari (2011), com o passar dos anos a Google teve que construir uma infraestrutura altamente escalável para seu mecanismo de busca e suas outras aplicações, como *Google Maps, Google Earth, Gmail, Google Finance, e Google Apps*; isso foi o início para construção de infraestruturas de gerenciamento paralelo em grandes quantidades de dados.

Com a adoção de *NoSQL* por empresas como, Google e Amazon, uma imensa variedade de produtos emergiu. Uma vasta porção de programadores iniciou o estudo em cima desta ideia, e de como aplicá-las em seu desenvolvimento, empresas começaram a estudar mais a possibilidade de utilizar estes métodos. Durante os últimos cinco anos NoSQL e os conceitos de gerenciamento de grandes dados tem sido generalizado e utilizado por novas companhias como *Facebook*("Rede Social"), *Netflix*("Vídeo Locadora Virtual"), *Yahoo*, *Ebay*("Empresa de Comercio Eletrônico"), IBM("Desenvolvedor de Softwares"), e muitas outras. Além disso, várias destas citadas contribuem com produtos *open-source* e suas extensões em todo mundo (Tiwari, 2011, p.5-6).

2.1 – Introdução ao MongoDB.

Banker (2012) afirma que, aplicações escritas hoje em dia comumente utilizam um banco de dados relacional como base de dados primária. Muitos desenvolvedores utilizam *SQL* pela familiaridade, e estão conformados com a maneira que os dados estão formalizados, caso não gostem de trabalhar diretamente com o modelo relacional existe uma serie de ferramentas que eliminam um pouco da complexidade.·.

Banker (2012) comenta que MongoDB é uma base de dados para sistemas e aplicações desenvolvidas e que utilizam a infraestrutura da internet. Seu modelo de dados e estratégia foi construído para suportar grande volume de leitura e escrita, habilidade de ser facilmente escalável e com tolerância a falhas.

Chodorow, Dirolf (2010) comentam que, uma das principais razões para pessoas migrarem do modelo relacional para o MongoDB é sua capacidade de escalabilidade fácil, mas ele também possui muitos outros fatores positivos. Explicam que MongoDB é um banco de dados orientado a documentos, e não relacional, onde a ideia básica é trocar o conceito de "linhas", por algo mais flexível como "documentos", essa orientação a documentos permite uma representação hierárquica e complexa de relações em apenas um registro.

A afirmação acima é confirmada também por Banker (2012)**,** que afirma que aplicações de *E-Commerce*, por exemplo, que levariam dezenas de tabelas a serem construídas se torna complexo em comparação com a estrutura do MongoDB, que utilizaria apenas um único documento.

Chodorow, Dirolf (2010) comentam que MongoDB é uma base de dados escalável, flexível e poderoso, combinado com muitas das características mais úteis dos bancos de dados relacionais, como índices secundários, consultas de intervalo e classificação. MongoDB também é incrivelmente cheio de recursos: tem toneladas de recursos úteis, tais como suporte embutido para *MapReduce*("Semelhante ao *GROUP BY* do *SQL*") de estilo de agregação e índices geoespaciais, e ainda, possui um modelo de dados amigável, fácil administração e configuração das opções, assim ele permite o foco no desenvolvimento do software, ao invés da preocupação com a base de dados.

2.2 - História

Segundo Banker(2012) que, o MongoDB nasceu fora de um projeto muito mais ambicioso, por volta de 2007 uma empresa chamada *10gen* iniciou seu trabalho com software do tipo plataforma como serviço, composto por uma aplicação servidor e uma base de dados, que iriam hospedar aplicações web e escalá-las quando necessário. Assim como o *Google ap. Eugene*, a *10gen* proporcionava uma

plataforma para desenvolvimento sem se preocupar com infraestrutura de *software* ou *hardware*, frisando com que os desenvolvedores focassem somente no código de suas aplicações. A 10gen finalmente descobriu que os usuários estavam se sentindo descontentes com a perda de controle sobre suas tecnologias, mas queriam uma nova base de dados. Com isso, ela direcionou seus esforços para a criação de uma base de dados que recebeu o nome de MongoDB. Com a incrível adoção e implantações em grandes e pequenas aplicações, a 10gen continuou a expandir o desenvolvimento desta base de dados como projeto open-source, onde o código publicado pode ser alterado e usado. E sua grande comunidade encoraja a busca por *bugs* a serem reportados e atualizados em *patches*. Ainda assim todos os desenvolveres do MongoDB se encontram em sua central, ou são desenvolvedores da própria *10gen*. E o roteiro do projeto continua sendo determinado pelos usuários, dessa forma *10gen* não e diferente das outras empresas de software livre, que apoiam desenvolvimento de código aberto, e oferecem serviços para os usuários finais.

2.3 – Estrutura dos documentos.

Banker (2012) comenta que, a estrutura de dados do MongoDB é organizada em um documento do tipo JSON.

Segundo Plugge, Membrey e Hawkins (2010, p.6):

> "JSON permite complexas estruturas de dados serem representadas em um simples texto, que podem ser facilmente lidos pelo ser humano, geralmente é considerado mais fácil do que XML. Como XML, JSON foi visto como uma maneira de trocar dados entre clientes e aplicações web (através do *browser*). Quando utilizado de maneira correta pode descrever objetos, essa simplicidade que influência em sua escolha, para muitos programadores".

Segundo Banker (2012 p. 4) **"**Quando você abre o MongoDB em JavaScript Shell, é possível de se obter facilmente a compreensão de seu produto, com todas as informações organizadas hierarquicamente em estrutura JSON".

JSON é mais do que apenas uma maneira de trocar dados, ele também e ótimo para armazenamento. Bancos de dados relacionais (RDBMS) são fortemente estruturados em tabelas, e seu armazenamento é individual. MongoDB por outro lado utiliza apenas um documento para tudo. Em JSON não é preciso especificar a estrutura nem os dados do documento, isso o torna eficaz porque podem ser adicionados de maneira separada independente de outros. Mas atualmente MongoDB não utiliza JSON como estrutura de armazenamento, e sim um formato de dados abertos desenvolvidos pelo próprio time do MongoDB chamado de BSON, e este fato não modifica a forma com que trabalhamos com MongoDB. BSON permite com que o MongoDB trabalhe de maneira rápida, é possível de essa forma criar processar e buscar documentos facilmente (Plugge; Membrey; Hawki 2010, p.5-6).

Banker (2012) afirma que os documentos salvos pelo MongoDB são representados pelo formato BSON, incluindo todas as *queries* e comandos que são especificados utilizando este formato, portanto todos os *drivers* do MongoDB devem estar aptos a compreendê-lo. Segundo ele BSON é o formato binário utilizado para representar os dados em MongoDB, esse por sua vez define os tipos de dados suportados pelo banco que atualmente são cerca de 19, desta forma quando um documento e inserido no banco seus dados são convertidos para seus tipos correspondentes encontrados no modelo BSON.

Na tabela seguinte será apresentado um comparativo entre termos utilizados em MySQL e MongoDB.

Tabela 1 – Termos utilizados (In: http://www.mongodb.org, 2012).

MySQL term	Mongo term/concept
database	database
table	collection
index	index
row	BSON documment
column	BSON field
join	Embedding and linking
primary key	_id field
group by	aggregation

2.4 – AD HOC Queries

Segundo Banker (2012) Ad Hoc *queries* podem ser facilmente adotadas se todas as bases de dados em questão utilizam modelo relacional, mas nem todas as bases de dados estão aptas a utilizar este tipo de consulta, portanto muitos sistemas abrem mão deste modelo de consulta rica em prol de uma escalabilidade simples. Um dos objetivos do MongoDB é preservar todo o poder de consulta utilizado nos bancos de dados relacionais, portanto ele permite o use deste tipo de consulta.

Podemos observar como funciona este tipo de *query* utilizado pelo MongoDB no seguinte exemplo, onde é necessário retornar o nome de todos os usuários que tiverem idade acima de 28 anos.

SQL :

```
SELECT nome FROM usuarios WHERE idade > 28
```

MongoDB Query:

```
db.usuarios.find( { ' nome ', ' idade ' : { ' $gt ' :  28 } } );
```

Figura 1 – Select MongoDB

2.5 – Índices Secundários

Segundo Banker (2012) índices secundários em MongoDB são implementados em árvores-B ("Organização padrão para índices em bancos de dados"), assim como em muitos bancos de dados. MongoDB permite esses índices secundários para aperfeiçoar uma grande variedade de procedimentos, pode-se declarar ate 64 índices por *collections*. Os tipos de índices suportados incluem os mesmos do modelo RDMBS *ascending*, *descending*, *unique*, *compound-key*, e também *geospatial indexes*. Justamente pelo MongoDB e muitos dos modelos RDMBS terem a mesma estrutura compatível de índices, o gerenciamento avançado deles e praticamente o mesmo.

Segundo Plugge, Membrey e Hawkins (2010, p.41) "O maior beneficio de criar nossos próprios índices está na consulta que se tornará incrivelmente mais rápida, por não precisarmos navegar através de uma base de dados para coletar as informações.".

Com isso podemos observar que esses índices secundários eliminam um dos problemas em *SQL* que é o aumento gradativo do código de consulta.

2.6 – Replicação

Segundo Banker (2012 ,p.10):

> *"Replicação de dados em MongoDB utiliza uma topologia conhecida como replica set. Replica set distribui os dados através de máquinas garantindo redundância em caso de falha de conexão entre o banco. Adicionalmente e utilizado para escalonamento entre leitura intensiva, e possível sua utilização para leitura de dados através de cluster entre máquinas."*

Banker (2012) afirma que esta técnica consiste em um nó mestre e um ou mais nós secundários que fazem a replica dos dados. Onde o mestre aceita escrita e leitura, mas os escravos apenas aceitam leitura. Caso o nó mestre venha a falhar, um dos nós secundários irá tomar o seu lugar, para que os processos continuem a ocorrer sem nenhuma perca de desempenho. Caso o antigo mestre volte a operar automaticamente será coloca na posição de escravo, dessa forma sua função agora passa a ser à de um replicante também.

Plugge; Membrey; Hawkins (2010) explica que replicação pode ser usada para garantir escalabilidade, durabilidade e isolação, e que para aplicações web escalabilidade é um requerimento de *design*, principalmente quando precisam de uma base de dados *back-end*, ajudando de duas maneiras: Melhora de desempenho – Principalmente em casos que a aplicação web é grande, e a leitura na base de dados também, onde também precise distribuir as *queryes* de maneira paralela entre servidores web.

A segunda é a Melhora de Redundância – Onde é possível distribuir em d*ata center's* diferentes a replica dos dados, isso torna mais rápido a velocidade de

acesso, porque usuários podem utilizar data center's distintos tornando menor o tempo de resposta.

A Figura 2 Exemplo de Replica Set.(In: CHODOROW; DIROLF et al., 2010, p. 128), ilustra o funcionamento de uma *replica Set* entre um nó *master* e seus respectivos três nós escravos.

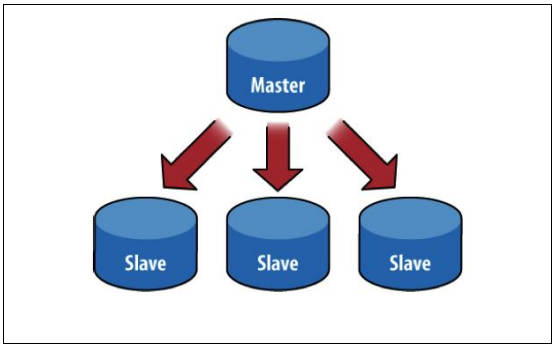

Figura 2. Exemplo de Replica Set.(In: CHODOROW; DIROLF et al., 2010, p. 128)

2.7 – Velocidade de Escrita e Durabilidade

Banker(2012) comenta que, MongoDB permite que o usuário controle a velocidade de escrita e durabilidade de sua aplicação, decidindo se quer optar ou não pelo *journaling*("Um *log* alocado em uma área especial do sistema de arquivos para possível recuperação caso haja falha, neste caso relacionado à operações com o banco de dados"), por padrão ele já vem com iniciativa de que o usuário não precisa se preocupar, *fire-and-forget*(" 'Atire e esqueça', expressão utilizada no sentido de passar informações para o banco de dados, e deixar que ele faça todo o resto do serviço"), apenas garante que o conteúdo será armazenado com sucesso, com meios de que a escrita seja enviada por meio do socket TCP, sem a necessidade de requisição de resposta do banco de dados. Caso o usuário necessite de uma reposta, pode utilizar o modo de safe *mode* ("modo utilizado do

MongoDB quando é necessária uma resposta mais especificada das operações com o banco"), que força a resposta se a escrita não tiver sido executada com sucesso. O safe *mode* pode ser configurado para esperar ate que os servidores secundários conseguirem replicar a escrita, só então gerar o retorno. Para grandes volumes de dados que não são tão importantes, o modo *fire-and-forget* é o mais adequado, em contrapartida para dados de suma importância o *safe mode* e mais aconselhado. Por padrão o *journaling* vem como padrão no MongoDB, portanto todas as operações de escritas geram um *log*, caso haja uma eventual falha no servidor as informações podem ser restauradas com garantia de integridade no retorno do mesmo, e se você estiver usando um *hardware* fisicamente, poderá achar muito caro a compra de um servidor mais poderoso.

2.8 – Scalling

Banker (2012) explica que a maneira mais fácil de escalar bancos de dados é dar um *upgrade* no *hardware*, e o nome dado as técnicas de aumentar o desempenho de apenas um banco de dados, em somente um nó de rede, são conhecidas como *vertical scaling* ou *scaling up*. *Vertical scaling* possui certas vantagens bem simples: é confiável e possui preço baixo para aumentar até certo ponto. Se estivermos usando um *hardware* virtualizado talvez o tamanho não seja suficientemente grande. Portanto começa a fazer sentido pensar em *horizontal scaling* ou *scaling out* que seguem o principio de distribuir a base de dados entre múltiplas máquinas, e utilizar o *hardware* que achar melhor para sua arquitetura, diminuindo o custo. Além do mais distribuir em múltiplas máquinas acaba por diminuir o impacto caso ocorram falhas já que inevitavelmente de tempos em tempos máquinas irão falhar, e caso isso aconteça ao estar utilizando uma arquitetura do tipo *vertical scalling*, todo o sistema estará comprometido. Diferentemente quando utilizado *horizontal scaling*, por mais catastrófica que seja o problema de uma

máquina, será afetado apenas uma porcentagem pequena do sistema como um todo.

Banker (2012) afirma que MongoDB proporciona gerenciamento de *horizontal scaling*, e isso é feito através de um mecanismo de separação baseado em intervalos, conhecido como *auto-sharding*, que gerência e distribui automaticamente os dados através dos nós. O sistema de *sharding* lida automaticamente com os nós adicionais, e facilita também *failover* automático. Dessa forma *Shards* individuais são constituídos por um conjunto de replicas consistido em pelo menos dois nós, assegurando recuperação automática caso um nó singular possua alguma falha. Tudo significa que o código da aplicação não precisa lidar com essas logísticas, devendo preocupar-se em conversar com apenas um nó, que seria o *cluster* com *Sharp*.

A Figura 3 – *Scalling up* e *Scalling out* (In: BANKER et al. 2012, p. 13) a seguir ilustra as duas formas de *upgrade* em uma base de dados que foi abordada neste capítulo 2.8 que são *Scalling up* e *Scalling*.

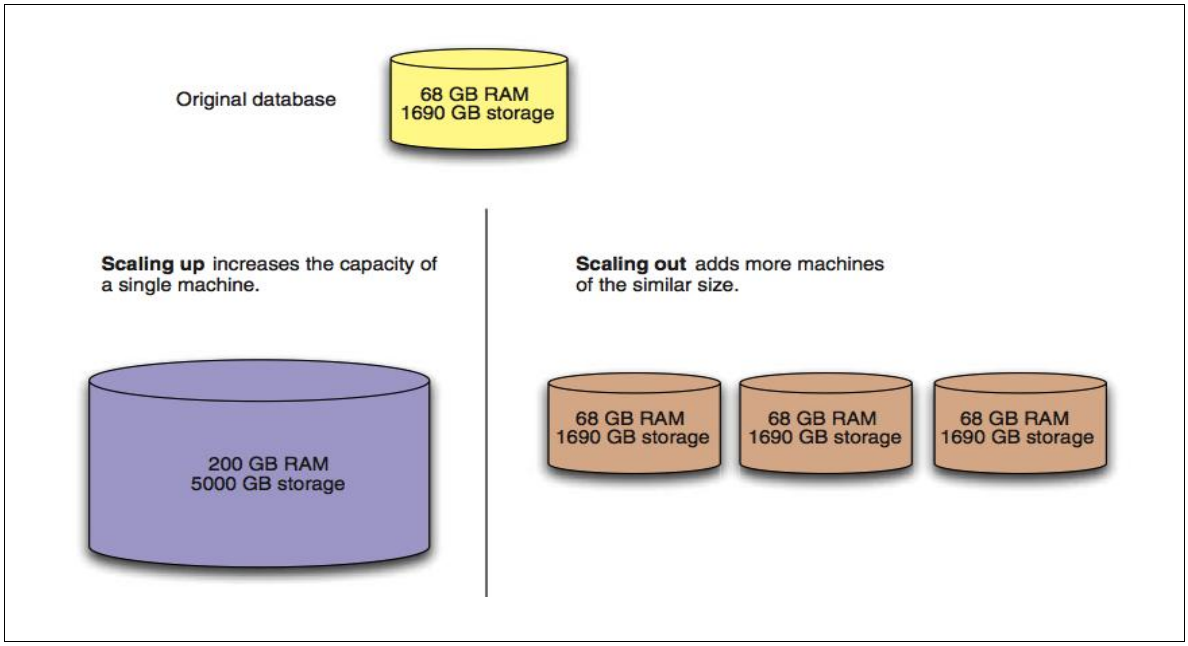

Figura 3 – *Scaling up* e *Scaling out*. (In: BANKER et al., 2012, p. 13)

Plugge; Membrey; Hawkis (2010) explicam que *Auto Sharding* é a característica que mais expressa o poder do MongoDB. Que existem dois tipos de implementações auto e manual, e quando utilizamos manual, o cenário consiste em 2 servidores mestres onde é configurado para se armazenar parte dos dados em um e o restante no outro. É possível inclusive escolher o que será armazenado em qual servidor. Mas utilizando este modelo acaba-se perdendo uma das maiores características do MongoDB, a sua simplicidade, por isso que é recomendado o uso de *auto sharding*, pois assim o próprio MongoDB irá tomar conta de armazenar os dados de maneira correta, distribuíndo entre os servidores.

A figura 4 – *Sharding*.(In: http://www.mongodb.org, 2012), ilustrará como a operação de *sharding* trabalha com os nós da rede, para que utilize também dos recursos de *Scalling out.*

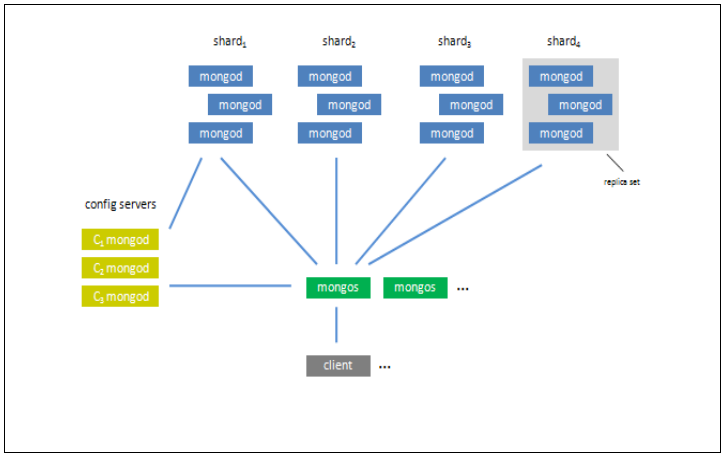

Figura 4 – *Sharding*.(In: http://www.mongodb.org, 2012)

2.9 – Limitações

Banker, Kile (2012) explica que existem algumas limitações para colocarmos o MongoDB em execução. Sua primeira é em relação ao tipo de sistema em que estará operante. Sabendo-se que sistemas de 32-bits não reconhecem mais do que

4GB de memória RAM, e que parte dessa memória é alocada para uso do sistema operacional, sendo que o MongoDB necessita de 2GB livres para mapear os arquivos de dados, proporciona um certo limite para à criação de índices, portanto é necessário um sistema de 64-bits. Uma segunda consequência do uso de mapeamento de memória virtual é que os dados serão atribuídos automaticamente conforme necessário. E finalmente e de suma importância o uso de replicação com MongoDB, principalmente se o *journaling* estiver inativo, pois como sua forma de trabalhar e com mapeamento de arquivos em memória, uma queda de energia qualquer poderia corrompê-los.

2.10 – Base de Dados

Segue abaixo a tabela com o nome dos respectivos executáveis sobre os bancos de dados MysQL, Oracle e MongoDB:

Tabela 2 – Executáveis (In: http://www.mongodb.org, 2012).

MySQL executable	Oracle executable	Mongo executable
mysqld	oracle	mongod
mysql	slqplus	mongo

No capitulo 2.3 foi abordado como funciona a estrutura dos documentos em MongoDB, agora veremos como a base de dados se comporta.

Segundo Plugge; Menbrey; Hawkins (2010) por MongoDB não ser uma base de dados não relacional, e não possuir colunas, nem tipo pré-definidos, o trabalho torna-se muito flexível, porque consequentemente não possui também estruturas pré-definidas para *documents*. Dessa forma e possível ter centenas de *documents* estruturados diferentemente e ainda não quebrar nenhuma regra da *collection*.

A Figura 5 mostra um documento de uma base de dados qualquer no MongoDB com seus respectivos valores.

```
{
    "_id" : Object("4d3ed089fb60ab729684b7e9"),
    "tipo" : "aluno",
    "nome" : "Érion R. Barasuol",
    "ano_entrada" : "2008",
    "ano_saida" : "2012",
    "curso" : "BCC"
    "notas" : [{ "ano1" : "8.5","ano2" : "9.0","ano3" : "8.0",}]
}
```

Figura 5 – Exemplo Document 1

A Figura 6 – ilustraria exatamente as mesmas características da Figura 5, se não fosse o fato do documento conter um campo e valores ("notas_pic" : [{"ano2" : "8.0","ano3" : "9.5"}]) a mais do que na figura anterior, esse campo por incrível que pareça não influenciará em absolutamente nada perante a collection, pois como foi abordado anteriormente MongoDB possibilita uma estruturação homogenia entre os documents dentro de uma collection.

```
{
    "_id" : Object("4d3ed0693b63ch534684b7e9"),
    "tipo" : "aluno",
    "nome" : "Adriel F.  Santiago Cavaleiro",
    "ano_entrada" : "2008",
    "ano_saida" : "2012",
    "curso" : "BCC"
    "notas" : [{"ano1" : "7.5","ano2" : "8.0","ano3" : "8.0","ano4" : "9.0"}]
    "notas_pic": [{"ano2": "8.0", "ano3": "9.5"}]
}
```

Figura 6 – Exemplo Document 2

Como já foi apresentado no capítulo 2.3 o exemplo demonstra que documents são escritos numa estrutura JSON, por sinal muito semelhante às classes que são

implementadas na maioria das linguagens orientadas a objeto. Também expressam como é possível o uso de *documents* diferentes na mesma *collection* do MongoDB.

A Figura 7 ilustra como é o modelo de dados do MongoDB e o nome dos elementos utilizados em usa estrutura como *Collections* e *Documents*.

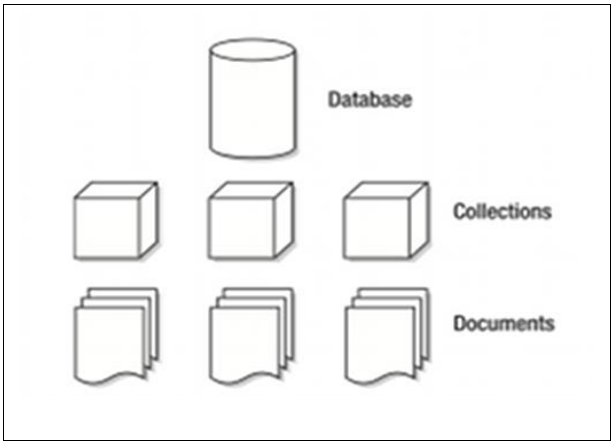

Figura 7 – Modelo de dados MongoDB.(In: PLUGE; MENBREY; HAWKINS et al., 2012, p. 37)

Segue abaixo a estrutura do modelo de dados relacionais, representado pela Figura 8.

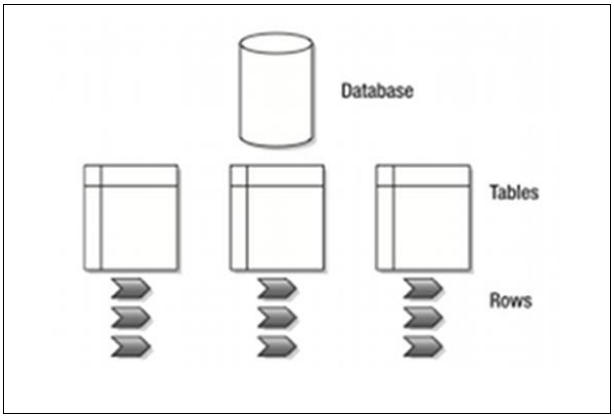

Figura 8 – Modelo de dados Relacional.(In: BANKER et al., 2012, p. 37)

Nas figuras 7 e 8 o modelo não difere muito, segundo Plugge; Menbrey; Hawkins (2010), as *collections* possuem tamanho expansível ao necessário, ou seja, ela cresce conforme novos *documents* são inseridos.

2.11 – Relações entre Documents

Plugge; Menbrey; Hawkins (2010) explicam que como em bases de dados relacionais, MongoDB também possui relações entre *documents*, mas de uma maneira diferente, normalmente utilizando *embeding* ou *referencing*. *Embeding* seria adicionar um *array* com mais informações dentro do documento, e *referencing* é parecido com as referencias externa em modelos relacionais, onde referenciamos dados de outro document.

Eles afirmam que a prática de *embeding* aumenta em muito a velocidade com que as informações são consultadas, porque desta forma, os dados já se encontram localizados no disco.

Banker (2012, p.249) afirma que "*Embed* quando os objetos filhos sempre aparecerem dentro do contexto dos pais. Caso contrário armazene os objetos filhos em *collections* separadas.".

A Figura 9 ilustra o código de uma relação *embeding* em MongoDB, entre funcionário e dependentes:

```
{
    "_id": Object ("4d3ed089fb60ab534684b7e9"),
    "nome": "Ruan Tavares Alves",
    "funcao": "Desenvolvedor Web",
    "email": "ruanTav@gmail.com",
    "dependente": {
                    "nome": "Marcelo Tavares Alves",
                     "relacao": "filho"
                  }
 }
```

Figura 9 – Exemplo embeding

A imagem 10 ilustra o código de uma relação *referencing* em MongoDB, relação entre Jogador e Time:

```
{
   "_id": Object ("4d3ed089fb60ab534684b7e9"),
   "nome": "Jorge Henrique",
   "posicao": "meio-campo",
   "time": Object(4d3ed089zx40ab546684b5e9)
}

{
   "_id": Object(4d3ed089zx40ab546684b5e9),
   "nome": "Corinthians",
   "pais": "Brasil",
   "regiao": "São Paulo"
}
```

Figura 10 – Exemplo *referencing*

2.12 – Operações na Base de Dados

Segundo Banker (2012) as operações com MongoDB podem ser realizadas no terminal e funcionam da seguinte forma:

Para salvar documento no MongoDB e preciso especificar em qual *collection* será inserido, por exemplo:

```
>db.alunos.insert({nome: "Érion", curso: "BCC"})
```

Para efetuar uma consulta utiliza-se a mesma sintaxe de inserção trocando apenas o método insert pelo find(), por exemplo:

```
>db.alunos.find()
```

Este comando retornará todos os documentos inseridos na collection alunos, quando retornamos algum documento de alguma *collection*, podemos observar um

campo que caso não tenhamos especificado será criado automaticamente, chamado de _id (Object Id).

Para sabermos quantos documentos estão em uma *collection* usamos o count(), por exemplo:

```
>db.alunos.count()
>1
```

Neste exemplo é informado que na base de dados db dentro da *collection* alunos possui 1 documento armazenado.

As formas de busca em MongoDB também podem ser mais aprimoradas, por exemplo, caso queiramos criar uma busca por todos os *documents* da *collection* aluno, que possuam o nome Erion ficaria parecido com:

```
>db.alunos.find( {"nome" : "erion"} )
```

db = Nome da *database*.

alunos = nome da *collection*.

find = método de busca em MongoDB.

nome = campo no documento desejado a buscar.

Erion = nome para busca no *document*.

Para que todos os updates funcionem é necessário especificar dois argumentos. O primeiro a informar, é onde será realizado a operação, e o segundo será a própria alteração em si. Por exemplo:

```
>db.alunos.update({nome: "Erion"}, {$set: {country: "Brasil"}})
```

Neste exemplo e informado que quando o nome for igual à Erion o país será trocado para Brasil.

Caso não seja mais necessária essa alteração, então podemos facilmente desfaze-la utilizando o $unset. Exemplo:

>db.users.update({username: "erion"}, {$unset: {country: 1}})

Tendo em mente que documentos podem ser também estruturados dentro de outros documentos como abaixo:

```
{
    nome: "Adriel"
    hobbies: {
                games: ["WoW", "Point Blank"],
                TV: ["Globo Esporte","4 rodas", "UFC - The Ultimate Fighter"]
    }
}
```

Figura 11 – Exemplo *document* estruturado

Um update de document dentro de outro document também é valido, e pode ser efetuado da seguinte forma:

>db.alunos.update({username: Erion},{$set: {hobbies:{games: ["Call of Duty", "Crisys"]}}})

Neste caso acabamos de efetuar um update para que ele fique em uma situação similar ao exemplo anterior, estruturado com dois documentos se relacionando.

É possível retornar através da navegabilidade do MongoDB os dados armazenados nessa estrutura de *documents* relacionados, da seguinte maneira.

Por exemplo a *query* abaixo retorna todos os *documents* que possuem *Crisys* como um game dentro de *hobbies*:

> db.alunos.find({"hobbies.games" : "Crisys"})

O ponto entre *hobbies* e *games* indica que acorrerá uma navegabilidade dentro de um outro documento para retorno da *query*.

Outra forma de *update* em MongoDB seria o caso de por exemplo termos a estrutura elaborada acima, e quiséssemos colocar mais um jogo na lista *games* para todos os documents que possuírem o jogo *Crysis*, usando o conceito de navegabilidade novamente podemos utilizar $push ou $addToSet, ambos adicionam um elemento a lista, mas o segundo em especial evita duplicação de valores, um exemplo a seguir:

```
> db.users.update({"hobbies.games" : "Crysis"},
{$addToSet: {"hobbies.games" : "Dinasty Warriors"} },false,true)
```

O primeiro argumento informado será para todos os documents que possuírem Crysis em sua lista de games, o segundo insere *Dinasty Warriors* pelo $addToSet. Por padrão o *multiupdate* é aplicado somente ao primeiro *document*, caso queiramos utilizar para todos, é necessário indicar explicitamente por meio deste argumento com *true*.

Para remover dados de uma *collection* no MongoDB utilizamos o método chamado remove() ex:

```
> db.users.remove()
```

O comando implementado acima remove todos os registros de uma *collection*, o que muitas vezes não é a situação mais desejada. Para isso e preciso informar uma *query* de expressão como parâmetro entre os parênteses do método remove(). Exemplo de remoção de um *document*:

```
> db.users.remove({"idade": "20"})
```

Já este comando acima demostra que será removido todos os *documents* de uma *collection* que contenham idade igual a 20. Caso seja necessário remover uma *collection* inteira o comando drop() deve ser utilizado então. EX:

```
> db.users.drop()
```

Neste capítulo foram apresentados os conceitos e exemplos do MongoDB. Sua principais características e como são representadas as informações que nele são armazenadas.

No próximo capítulo será apresentado o uso de tecnologias, que em conjunto com o MongoDB possibilitam sua utilização no desenvolvimento de sistemas.

3 – FERRAMENTAS COM MONGODB

MongoDB assim como a maioria dos bancos de dados possui uma ampla lista de *drivers* para linguagens de programação, dentre elas java foi escolhida para a representação de acesso as operações do MongoDB.

Apesar de possuir um *driver* de conexão com muitas facilidades prontas para utilização do MongoDB, fica mais fácil com o auxilio de POJOMappers("Mapeadores de classes e objetos Java") como o Morphia, todas essas tecnologias podem ser observadas com mais detalhes ao decorrer do capítulo.

3.1 – Java & MongoDB

Segundo site da Oracle "Java é uma linguagem de programação e uma plataforma de computação lançada pela primeira vez pela *Sun Microsystems* em 1995. É a tecnologia que capacita muitos programas da mais alta qualidade, como utilitários, jogos e aplicativos corporativos, entre muitos outros, por exemplo. O Java é executado em mais de 850 milhões de computadores pessoais e em bilhões de dispositivos em todo o mundo, inclusive telefones celulares e dispositivos de televisão."

Segundo site da 10gen - MongoDB "O *driver* para MongoDB de Java é uma *thread* segura. Se você estiver utilizando um ambiente de serviço *web*, por exemplo, é possível criar apenas uma única instância, e ainda usa-la para qualquer requisição. O objeto do MongoDB mantêm um *pool* interno de conexões com a base de dados. Para toda requisição feita à base de dados (*find*, *insert*, etc) a *thread* do java obtém uma conexão através do *pool*, executa as operações, e libera a conexão. Isto significa que a conexão(*socket*) pode ser diferente a cada vez."

Segundo informações obtidas através da MongoDB API Docs for java (2012), abaixo seguem as devidas implementações para MongoDB utilizando seu Driver e

API para Java.

3.1.1 – Criando conexão

Como iremos observar no exemplo da Figura 12, existem 3 maneiras de se criar um objeto para conexões com o banco de dados em MongoDB.

Os objetos da Figura 11 demonstram 3 maneiras de se obter uma instância do MongoDB, primeiroExemplo não usa parâmetros, segundoExemplo apenas utiliza o ip e terceiroExemplo é mais específico requisitando ip e porta do banco.

```
16  public class ExemploTcc {
17
18      //Cria a variavel de referência do MongoDB
19      private Mongo primeiroExemplo = null;
20      private Mongo segundoExemplo = null;
21      private Mongo terceiroExemplo = null;
22      //Adiciona o ip do MongoDB a variavel
23      private String ip = "localhost";
24      //Adiciona a porta que o MongodDB utiliza
25      private int porta = 27017;
26      //Cria a variavel da base de dados que sera utilizada mais tarde
27      private String BASEDADOS = "erion";
28      //
29      private DB db = null;
30
31      public ExemploTcc() {
32          try {
33              //Cria instancias do MongoDB e as adiciona
34              //a suas variaveis de referencia
35              primeiroExemplo = new Mongo();
36              segundoExemplo = new Mongo(ip);
37              terceiroExemplo = new Mongo(ip, porta);
38
39          } catch (MongoException e) {
40              //Captura MongoException
41              e.printStackTrace();
42
43          } catch (UnknownHostException e) {
44              //Captura UnknownHostException
45              e.printStackTrace();
46          }
47      }
48
49      public DB retornaObjetoBanco() {
50          db = terceiroExemplo.getDB(BASEDADOS);
51          return db;
52      }
53  }
```

Figura 12 – Conexão

3.1.2 – Inserindo dados no Banco

A Figura 13 demonstra como inserir *documents* no MongoDB. Os objetos Java persistidos não sofrem nenhum tipo de alteração, o que acontece e que os valores contidos nele, são retornados e passados como parâmetro, para um tipo específico aceito pelo MongoDB, chamado BasicDBObject.

Feito isso só resta realizar a persistência no banco através do método insert, que por sinal é o mesmo utilizado no capítulo 2.12.

```
26
27      public boolean inserir(Object obj) {
28          //Faz um cast para objeto do tipo Cliente
29          Cliente cli = (Cliente) obj;
30          try {
31              //Abre uma nova conexao
32              mgConn.conectar();
33              //Cria um objeto DB do banco
34              DB db = mgConn.retornaObjetoBanco();
35              //Retorna uma collection(Tabela) desejada
36              DBCollection coll = db.getCollection("mycoll");
37              //Cria um document para inserir o objeto no banco
38              BasicDBObject doc = new BasicDBObject();
39              if (!((cli.getId() == null) && (cli.getNome() == null)
40                      && (cli.getEmail() == null))) {
41                  doc.put("id", cli.getId());
42                  doc.put("nome", cli.getNome());
43                  doc.put("email", cli.getEmail());
44                  //Adiciona o objeto a collection ativa
45                  //OBS: automaticamente e inserido no banco
46                  coll.insert(doc);
47                  //retorna true para o metodo
48                  return true;
49              } else {
50                  return false;
51              }
52          } catch (MongoException ex) {
53              //Traca a pilha de execoes
55              ex.printStackTrace();
56              //retorna false para o metodo
57              return false;
58          }
      }
```

Figura 13 – Inserir

3.1.3 – Atualizando dados do Banco

Este exemplo realiza *update* dos dados existentes em um documento, onde a condição que determina com exatidão é o atributo "id".

O método *update* necessita de dois BasicDBObject`s como parâmetro, o primeiro é a critéria (condição de busca) para o documento, e o segundo finalmente e o documento já formado com os dados já atualizados.

```
59
60      public boolean atualizar(Object obj) {
61          //Faz um cast para objeto do tipo Cliente
62          Cliente cli = (Cliente) obj;
63          try {
64              //Abre uma nova conexao
65              mgConn.conectar();
66              //Cria um objeto DB do banco
67              DB db = mgConn.retornaObjetoBanco();
68              //Retorna uma collection(Tabela) desejada
69              DBCollection coll = db.getCollection("mycoll");
70              //Cria um novo document para atualizar o objeto no banco
71              BasicDBObject newDoc = new BasicDBObject();
72              //Testa se o objeto nao e null
73              if (!((cli.getId() == null) && (cli.getNome() == null)
74                      && (cli.getEmail() == null))) {
75                  newDoc.put("id", cli.getId());
76                  newDoc.put("nome", cli.getNome());
77                  newDoc.put("email", cli.getEmail());
78                  //Adiciona o objeto a collection ativa
79                  //OBS: automaticamente e atualizado no banco
80                  coll.update(new BasicDBObject().appehd("id", cli.getId()), n
81                  //retorna true para o metodo
82                  return true;
83              } else {
84                  return false;
85              }
86          } catch (MongoException ex) {
87              //Traca a pilha de execoes
88              ex.printStackTrace();
89              //retorna false para o metodo
90              return false;
91          } finally {
92              //Desconeca do banco
93          }
94      }
```

Figura 14 – Atualizar

3.1.4 – Selecionar dados do Banco

A Figura 15, demonstra que através do método find() visto no capítulo 2.12 é retornado uma DBCollection("lista de *documents*") do banco, e criado um DBCursor, para rodar dentro do bloco *while*, onde é criada uma lista de objetos Java para receber esses dados retornados da *collection*.

```
95
96      public List<Cliente> selecionar() {
97          //Cria a lista de clientes
98          List<Cliente> cliList;
99          //Cria uma referencia para a classe Cliente
100         Cliente cli1;
101         try {
102             //Abre uma nova conexao
103             mgConn.conectar();
104             //
105             cliList = new ArrayList<Cliente>();
106             //Cria um objeto DB do banco
107             DB db = mgConn.retornaObjetoBanco();
108             //Retorna uma collection(Tabela) desejada
109             DBCollection coll = db.getCollection("mycoll");
110             //Declara um cursor para obter todas os documents(Registros)
111             DBCursor cur = coll.find();
112             while (cur.hasNext()) {
113                 //Cria o objeto
114                 cli1 = new Cliente();
115                 //Atribui os valores ao objeto
116                 cli1.setId((cur.next().get("id")).toString());
117                 cli1.setNome((cur.curr().get("nome")).toString());
118                 cli1.setEmail((cur.curr().get("email")).toString());
119                 cliList.add(cli1);
120             }
121             return cliList;
122         } catch (MongoException ex) {
123             //Traca a pilha de execoes
                ex.printStackTrace();
125             //Retorna null
126             return null;
127         }
128     }
```

Figura 15 – Selecionar

3.1.5 – Remover dados do Banco

Na Figura 16, o método deletar recebe apenas o id do *document* a ser removido da *collection*. Quando passamos um novo BasicDBObject ().append("_id", id), estamos dizendo que irá remover todos os documentos da collection que contiverem o id de número correspondente ao que foi passado por parâmetro.

```
130     public boolean deletar(int id) {
131         try {
132             //Abre uma nova conexao
133             mgConn.conectar();
134             //Cria um objeto DB do banco
135             DB db = mgConn.retornaObjetoBanco();
136             //Retorna uma collection(Tabela) desejada
137             DBCollection coll = db.getCollection("mycoll");
138             coll.remove(new BasicDBObject().append("_id", id));
139             return true;
140         } catch (MongoException ex) {
141             //Traca a pilha de execoes
            ex.printStackTrace();
143             //retorna false para o metodo
144             return false;
145         }
146     }
147
```

Figura 16 – Deletar.

3.2 – Persistindo objetos Java pelo Morphia

Segundo D'Emic (2011)

> *"Morphia é um projeto Google Code com licença Apache que permite tornar persistente, recuperar, excluir e consultar POJOs armazenados como documents no MongoDB. Morphia consegue isso fornecendo um conjunto de anotações e um wrapper em torno do driver Java do Mongo. Morphia tem conceito semelhante ao de mapeadores relacionais de objetos, tais como implementações Java Persistence API (JPA) ou Java Data Objects (JDO)".*

Como foi abordado no capítulo 3, MongoDB facilita muito a persistência dos dados por não utilizar SQL's nem Joins, mas apesar disso ainda é necessário fazer com que os objetos BSON recebam os valores dos objetos Java, o que apesar de fácil acaba sendo um tanto custoso, se levarmos em consideração o tempo de produção de um software. Por isso como D'Emic explicou acima, Morphia desempenha esse papel semelhante a mapeadores relacionais de objetos, mas sem a necessidade de transformar objetos java em modelos compreensivos por bancos de dados relacionais.

Os códigos reais implementados abaixo demonstrarão a facilidade de persistência de objetos Java com Morphia.

3.2.1 – Inserindo com Morphia

A figura 17 expressa a maneira mais simplificada de se inserir um objeto no banco com Morphia. Pode-se observar que a maneira com que esta operação é feita se torna mais fácil pelo fato de não necessitar de intercâmbio de informações do objeto java para BasicDBObject como abordado durante o capítulo 3.

```
29      public boolean inserir(Object obj) {
30
31          model.Estado estd = (model.Estado) obj;
32          try {
33              mgConn.connect();
34              Morphia morphia = new Morphia();
35              morphia.map(model.Estado.class)
36                      .map(model.Cidade.class);
37              Datastore ds = mgConn.returnDatabaseObject(morphia);
38              ds.save(estd);
39              return true;
40          } catch (MongoException ex) {
41
              ex.printStackTrace();
43
44              return false;
45          }
46      }
```

Figura 17 – Inserir com Morphia

3.2.2 – Atualizar com Morphia

A figura 18 demonstra como a operação de atualização com morphia deve ser feita. Na variável *'up'* é atribuído o valor que será atualizado no banco, exatamente no campo *'cidList'*.

A variável queryOp contem a condição especificada para se efetuar tal operação, que no caso se trata da comparação entre o campo *'descrição'* e o valor obtido por *'getDercricao()'* no objeto *'estd'*. Ambas as variáveis são utilizadas como parâmetros no método *update*.

```
87      public boolean atualizar(Object obj) {
88
89          model.Estado estd = (model.Estado) obj;
90          try {
91              mgConn.connect();
92              Morphia morphia = new Morphia();
93              morphia.map(model.Estado.class)
94                     .map(Cidade.class);
95              Datastore ds = mgConn.returnDatabaseObject(morphia);
96
97              UpdateOperations<model.Estado> up = ds.createUpdateOperations
98                      (model.Estado.class).add("cidList", estd.getCidList().get(0));
99              Query<model.Estado> queryOp = ds.createQuery
00                     (model.Estado.class).field("descricao").equal(estd.getDescricao());
01
02
03              ds.update(queryOp, up);
04              return true;
05          } catch (MongoException ex) {
06
              ex.printStackTrace();
08
09              return false;
10          }
11      }
```

Figura 18 – Atualizar com Morphia

3.2.3 – Deletar com Morphia

Como é foi feito na figura 19, para se deletar do banco de dados apenas é necessário passar por parâmetro para o método *'delete()'*, um objeto contendo os valores a serem removidos.

```
68      public boolean deletar(Object obj) {
69
70          model.Estado estd = (model.Estado) obj;
71          try {
72              mgConn.connect();
73              Morphia morphia = new Morphia();
74              morphia.map(model.Estado.class)
75                      .map(model.Cidade.class);
76              Datastore ds = mgConn.returnDatabaseObject(morphia);
77              ds.delete(estd);
78              return true;
79          } catch (MongoException ex) {
80
82              ex.printStackTrace();
83
84              return false;
85          }
      }
```

Figura 19 – Deletar com Morphia

3.2.4 – Selecionar com Morphia

Por último, mas não menos importante, na figura 20 é possível observar que morphia permite que os dados sejam retornados do banco, especificando apenas qual classe java deve ser utilizada como modelo para adicionar os dados, que por sinal já retornam em formato de lista. Tudo através do método *'find()'.*

Ainda é possível fazer uma busca mais específica utilizando, por exemplo, o código que está comentado. Através de uma query como é demonstrado, onde o campo descrição do banco deveria ser igual a *'São Paulo'*.

```
48      public List<model.Estado> selecionarCidadesDeSaoPaulo(){
            List<model.Estado> estdList = null;
50          try {
51              mgConn.connect();
52              Morphia morphia = new Morphia();
53              morphia.map(model.Estado.class)
54                     .map(model.Cidade.class);
55              Datastore ds = mgConn.returnDatabaseObject(morphia);
56              //Query q = ds.createQuery(Estado.class).field("descricao").equal("São Paulo");
57              estdList = ds.find(model.Estado.class).asList();
58
59          } catch (MongoException ex) {
60
                ex.printStackTrace();
62
63              return null;
64          }
65          return estdList;
66      }
```

Figura 20 – Selecionar com Morphia

Neste capítulo foram apresentados tecnologias que podem ser utilizadas juntamente com o MongoDB para o desenvolvimento de um sistema. Java como uma linguagem de programação orientada a objetos que possibilita o desenvolvimento de sistemas destinados a ambientes tanto web quanto *desktop*, e Morphia, que proporciona um ciclo de desenvolvimento ágil, por ser um *framework* para manipulação dos dados mantidos no MongoDB.

No próximo capítulo será apresentado a documentação completa e a implementação de um sistema que comprovará o estudo, e demonstrará na prática como funciona o ambiente de desenvolvimento, quando adotado MongoDB e as tecnologias citadas no capítulo 3.

4 – SISTEMA DE RESERVAS ACADÊMICAS.

O sistema de reservas acadêmicas será desenvolvido para comprovar a base teórica abordada no capítulo 3. Neste capítulo será abordada a documentação completa do sistema como caso de uso, especificação do caso de uso etc...

4.1 – CRONOGRAMA DA ESTRUTURA DE DESENVOLVIMENTO

O desenvolvimento do software irá seguir o cronograma quinzenal definido na Tabela 03 – Cronograma de Desenvolvimento**.**

CRONOGRAMA	AGO		SET		OUT		NOV		DEZ	
Análise dos Requisitos										
Modelagem dos Requisitos										
Documentação										
Desenvolvimento										
Testes										
Implantação										

Tabela 3 – Cronograma de Desenvolvimento.

4.2 – DADOS DA EMPRESA

Empresa: FEMA – Fundação Educacional do Município de Assis.

Nome Fantasia: FEMA

AV: Avenida Getúlio Vargas, 1200
CEP: 19807-634
Telefone: (18) 3302-1055
Cidade: Assis - SP

4.3 – MÉTODO DE DESENVOLVIMENTO

Para desenvolvimento desse sistema será utilizado a linguagem Java por ser uma linguagem de programação totalmente orientada a objetos, o que permite o maior aproveitamento dos recursos do MongoDB. A ferramenta de desenvolvimento será o NetBeansIDE 7.2 por sua facilidade e popularidade com a linguagem.

O banco de dados será o MongoDB que é o objeto de estudo desse trabalho, e por final mas não menos importante a ferramenta utilizada para produção da UML e documentação do software será o Astah Community.

4.3.1 – Java

Linguagem de programação totalmente orientada a objetos, escolhida justamente por esse propósito. Explorara sua facilidade em um ambiente de desenvolvimento NoSQL e ao mesmo tempo usufruir dos recursos que o MongoDB disponibiliza para a linguagem. Mais detalhes sobre o Java podem ser encontrados no capítulo 3.1.

4.3.2 – MongoDB

MongoDB foi escolhido por ser o objeto de estudo desse trabalho será comprovado seu funcionamento com linguagens de programação totalmente orientadas a objeto como o Java.

4.3.3 – NetBeans

Ferramenta para desenvolvimento de sistemas na plataforma Java pode ser encontrada gratuitamente para diversos Sistemas Operacionais, alem de possibilitar o desenvolvimento de outras linguagens como em www.netbeans.org.

4.3.4 – Astah Community

Segundo o site www.astah.net/editions/community essa ferramenta gratuita possibilita a elaboração de modelos UML para documentação fácil e rápida, de uma maneira intuitiva.

4.3.5 – PLANEJAMENTO DO SISTEMA

Esse projeto será a reimplementação do sistema de reservas da FEMA, portanto será feita análise de requisitos através de entrevista com o desenvolvedor de sua atual versão.

4.3.6 – WBS – Work Breakdown Structure

Nesse subcapítulo a Figura 21 - WBS – Work Breakdown Structure, apresenta a estrutura e organização no desenvolvimento do sistema.

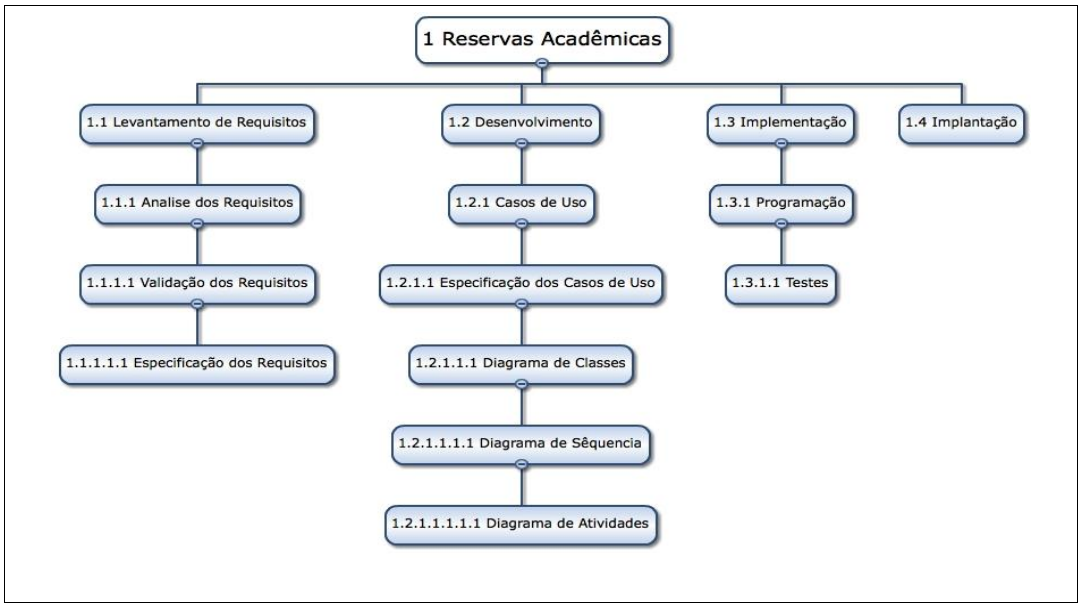

Figura 21 - WBS – Work Breakdown Structure

4.3.7 – SEQUENCIAMENTO DAS ATIVIDADES

Neste subcapítulo a Figura 22 – Sequenciamento de Atividades, apresenta o sequenciamento das atividades a serem desenvolvidas no sistema.

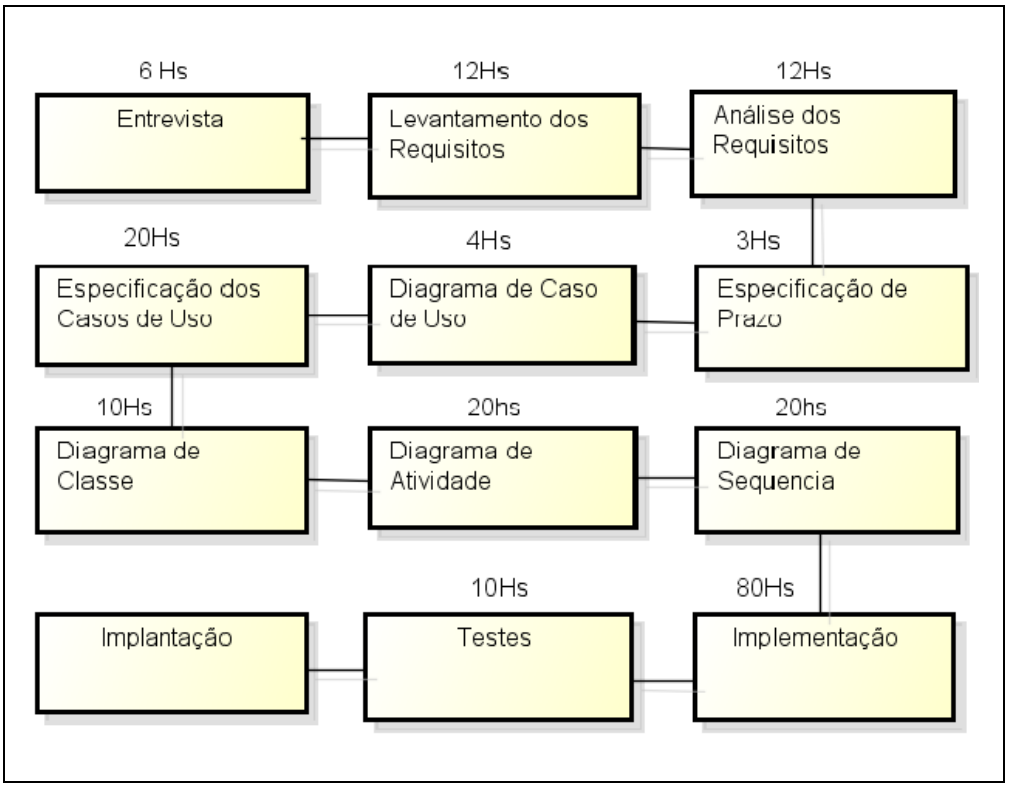

Figura 22 – Sequenciamento de Atividades

4.3.8 – Recursos Necessários para o Desenvolvimento do Projeto

1 (um) Analista de Sistema;

1 (um) Programador Java;

Sistema Operacional MacOS X;

IDE para o desenvolvimento: *Netbeans*;

IDE para modelagem de dados: JUDECommunity;

Linguagem para o desenvolvimento: Java;

Servidor de Aplicação: *Apache Tomcat 7*;

Bando de dados: *MongoDB*;

Aplicativos: *LibreOffice* e *GanttProject*.

4.3.9 – Estimativa de Custos

Analista de Sistema

Quantidade horas: 107

Custo hora: R$ 0,00

Custo total: (Total de horas * Custo hora) = R$ 0,00

Programador

Quantidade horas: 90

Custo diário: R$ 0,00

Custo total: (Total de horas * Custo hora) = R$ 0,00

Custo: R$ 0,00 (*Freeware*) **IDE para o desenvolvimento - *Netbeans***

Custo: R$ 0,00 (*Freeware*) **IDE para a modelagem de dados - JUDECommunity**

Custo: R$ 0,00 (*Freeware*) **Linguagem para o desenvolvimento: Java;**

Custo: R$ 0,00 (*Freeware*) **Servidor de Aplicação: *Tomcat 7*;**

Custo: R$ 0,00 (*Freeware*) **Bando de dados NoSQL: *MongoDB;***

Aplicativos: *LibreOffice* e *GanttProject*. Custo: R$ 0,00 (*Freeware*)

4.4 – LEVANTAMENTO DOS REQUISITOS

O levantamento de requisitos será feito mediante a presença do desenvolvedor da atual versão do sistema, para que todas as dúvidas sejam esclarecidas.

Entrevista

A entrevista foi realizada com o Professor Me. Douglas Sanches Cunha, responsável pelo desenvolvimento e manutenção do site da FEMA, dessa forma foi possível solucionar as dúvidas sobre o desenvolvimento do sistema.

Analista: Qual o principal motivo para atualização na arquitetura do sistema?
Cliente: Atualmente a lógica de negócio do sistema e sua interface necessitam de uma implementação mais flexível, por exemplo, os horários de agendamento que não possui a possibilidade de alteração ou inclusão.

Analista: Atualmente quais as tecnologias que foram utilizadas no desenvolvimento?
Cliente: A linguagem é JSP, webcontainer é o Glassfish e toda a interface com Icefaces.

Analista: Como deverá ser feito a lógica da tela de Reservas, já que você citou a lógica de negócios como um fator que necessita ser reimplementado.
Cliente: A lógica pode ser feita como quiser, desde que seja possível um agendamento flexível, no que diz respeito a data e hora, onde o usuário deverá escolher os mesmos em uma interface parecida com uma agenda.

Analista: Essa interface seria exatamente como a atual do sistema ou poderão ocorrer algumas modificações?

Cliente: Sim. Algumas modificações são aceitáveis, contanto que não influenciem no processo de agendamento que já e utilizado pelos professores atualmente.

4.5 – ANÁLISE DOS REQUISITOS

Nesse subcapítulo será abordada toda a análise dos requisitos como classificação, exigência e prioridades.

4.5.1 – Classificação dos Requisitos

Os seguintes requisitos foram levantados perante a entrevista. :

- Login
- Reservar Recurso
- Consultar Reserva
- Consultar Recurso
- Manter Hora
- Manter Data
- Manter Login
- Manter Categoria
- Manter Recurso

4.5.2 – Exigências

- O único usuário que poderá efetuar operações de crude será o administrador.
- O professor somente poderá consultar reservar ou remover suas reservas.
- O aluno somente poderá consultar as reservas pela data desejada.

- Reservas cadastradas no banco não podem ser removidas por outros usuários
- Uma categoria não poderá ser excluída caso haja recursos vinculados a ela.
- Um recurso não poderá ser excluído caso haja alguma reserva feita para o mesmo.
- Cada usuário deverá ter sua respectiva área de acesso logo após o login.

4.5.3 – Prioridades

O sistema deve efetuar reserva dos recursos utilizados pelos professores, bem como possibilitar o cadastro de horários e datas, além de login´s de acesso para usuários de 3 categorias diferentes: administrador, professor e aluno.

4.5.4 – Proposta de Solução

A proposta é desenvolver o sistema espelhando-se no já existente, mudando apenas a lógica de negócios da tela de reservas, garantindo maior flexibilidade para inclusão de horários e datas especiais como feriados, que não deveriam aparecer na seleção de data para consulta da tabela.

4.6 – Diagrama de Caso de Uso Global

A figura 23 – Diagrama de Caso de Uso Global, ilustra todas as funcionalidades do sistema integrando todas as visões (Administrador, Professor e Aluno), ao decorrer do capítulo 4 os demais diagramas ilustrarão de uma forma mais clara cada visão de uma maneira mais específica.

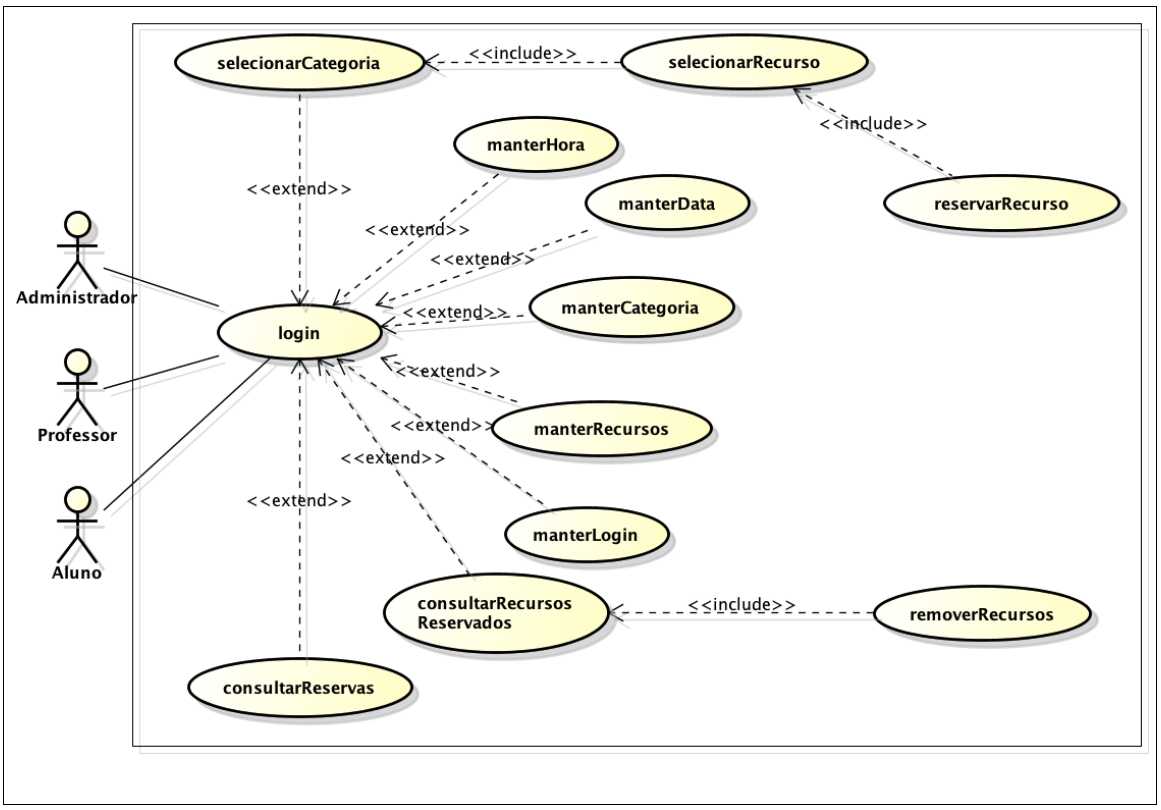

Figura 23 - Diagrama de Caso de Uso Global

4.7 – Especificação dos Casos de Uso.

Nesse subcapítulo será descrito os casos de uso bem como sua especificação e diagrama de sequência.

4.7.1 – Reservar Recurso

A Figura 24 – Caso de Uso: Reservar Recurso ilustra como os usuários Administrador e o Professor pode reservar algum recurso pelo sistema.

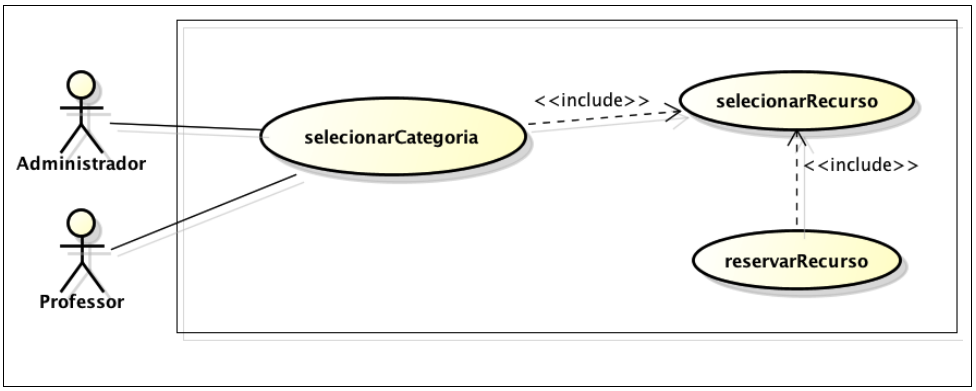

Figura 24 - Caso de Uso: Reservar Recurso

Finalidade/Objetivo:
Permite ao usuário reservar um recurso no sistema.

Atores:
Administrador, Professor.

Evento Inicial:
O usuário deve selecionar a opção reserva de recursos no menu principal do sistema, logo após efetuado o login.

Fluxo Principal:

O usuário deve selecionar em qual categoria estará o recurso, através de um drop down list;

O usuário deve selecionar qual recurso correspondente à categoria selecionada, através de um drop down list;

O usuário deve selecionar em qual data e horário o recurso será reservado, através da tabela de reservas logo abaixo do drop down list("categoria, recursos").

Fluxos Alternativos:

O usuário pode cancelar o processo de reserva e retornar ao menu principal através do botão "Voltar".

Fluxos de Exceção:

Caso o usuário não selecione nenhuma categoria e recurso, não será possível para o sistema exibir a tabela de reservas, e assim concluir o processo.

Pós Condições:

O usuário poderá remover o recurso reservado, em uma tabela que se posiciona logo abaixo da tabela de reservas, clicando no botao "remover".

Casos de Testes:

Tentar efetuar uma reserva na tabela sem selecionar categoria e recurso;

Tentar efetuar uma reserva em uma posição da tabela que já esteja com reserva;

Efetuar uma reserva em uma posição da tabela que não esteja com reserva;

Remover um recurso da tabela de recursos reservados;

Remover um recurso da tabela de recursos reservados e adiciona-lo novamente.

4.7.1.1 – Diagrama de Sequência: Reservar Recursos Admin

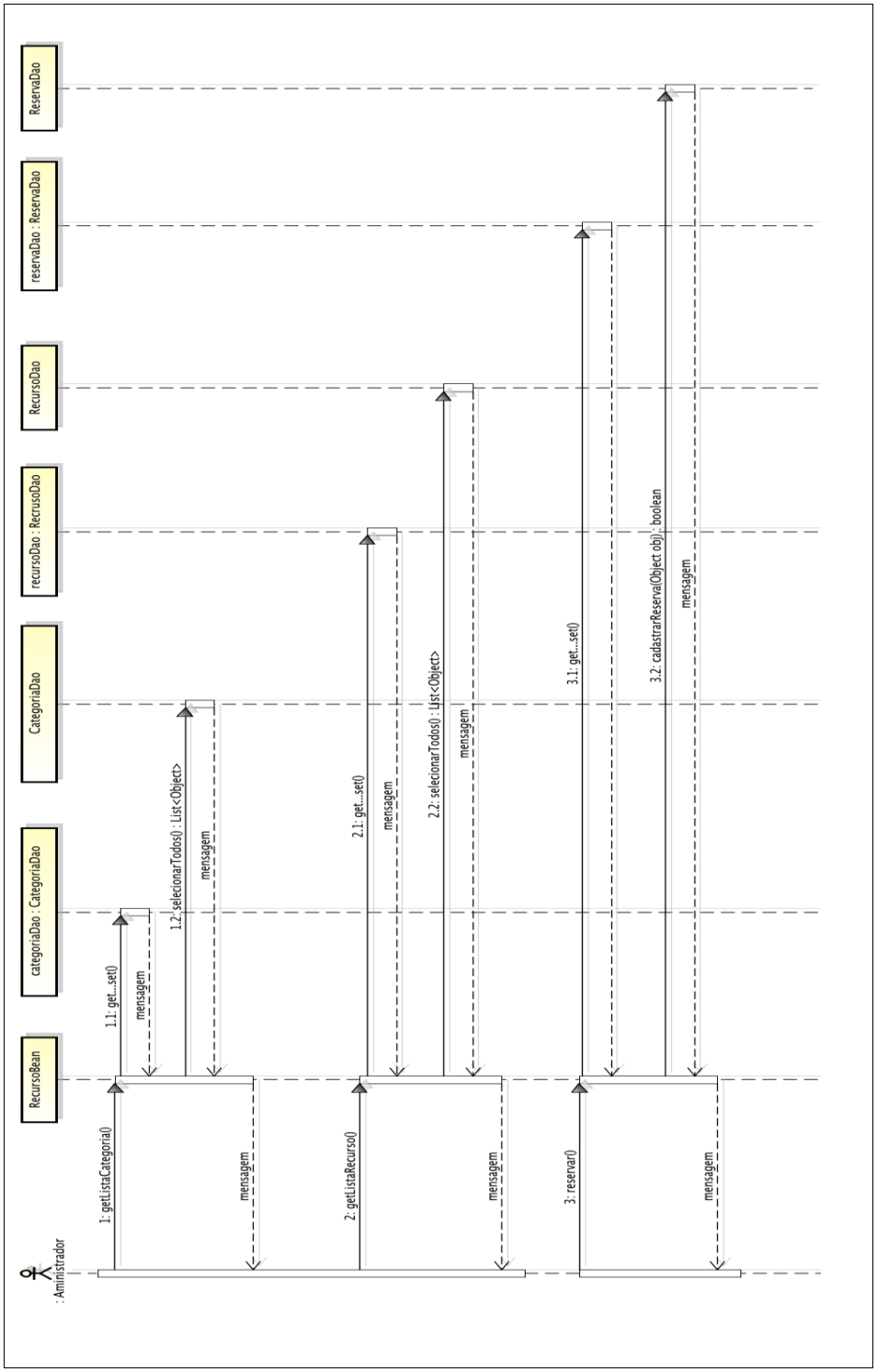

Figura 25 - Diagrama de Sequência: Reservar Recursos Admin

4.7.1.2 – Diagrama de Sequência: Reservar Recursos Professor

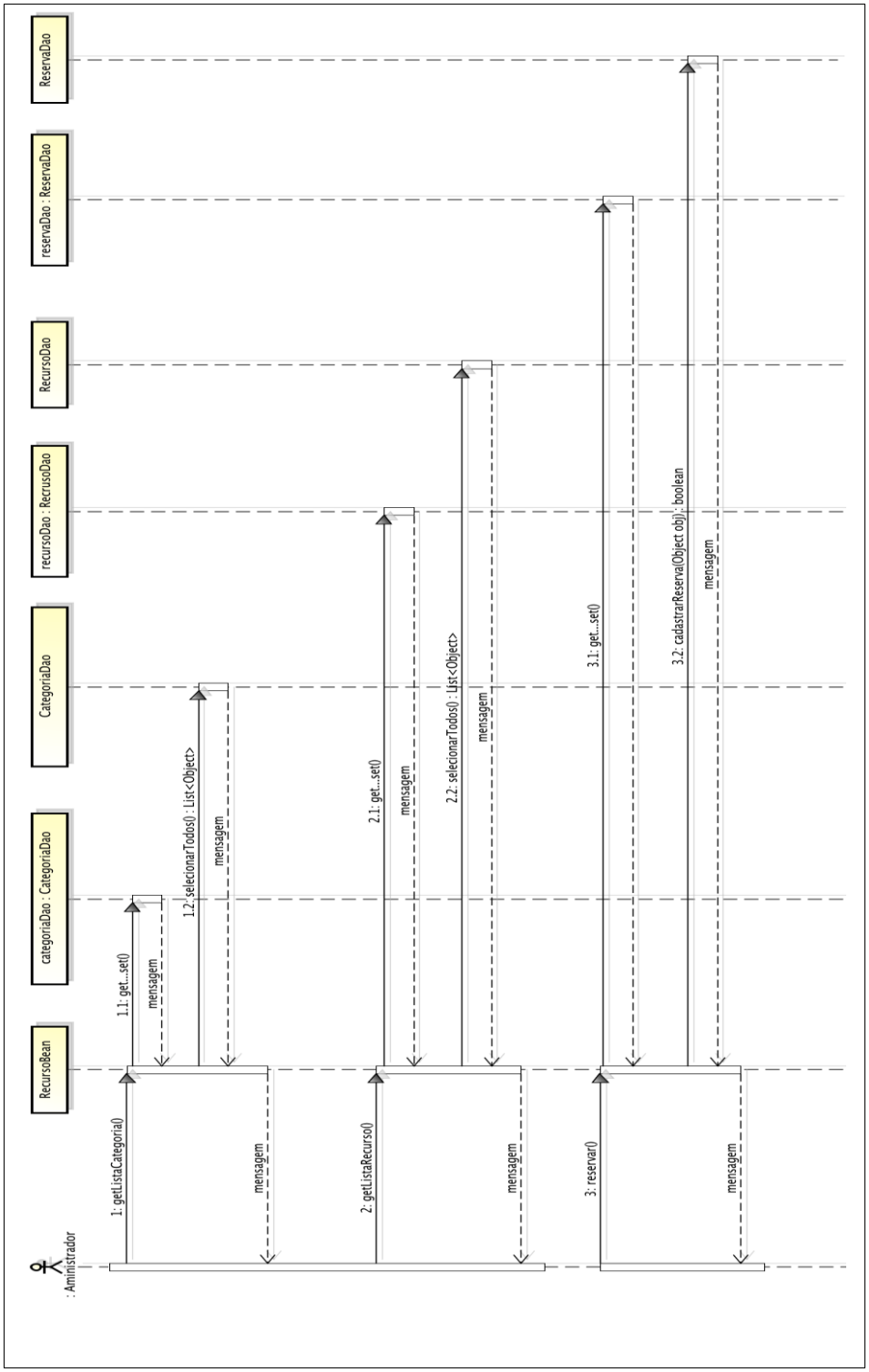

Figura 26 - Diagrama de Sequência: Reservar Recursos Professor

4.7.2 – Manter Categoria

A Figura 27 – Caso de Uso: Manter Categoria ilustra como o usuário Administrador pode gerenciar as categorias que disponibilizam recursos para reserva no sistema, de maneira que ele possa adicionar remover ou alterar a mesma.

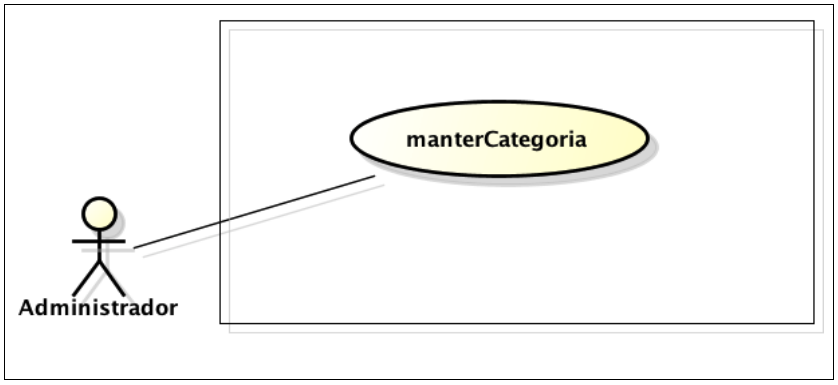

Figura 27 - Caso de Uso: Manter Categoria

Finalidade/Objetivo:

Permite ao usuário cadastrar, consultar ou remover uma categoria no sistema.

Atores:

Administrador.

Evento Inicial:

O usuário deve selecionar a opção cadastro de categorias no menu principal do sistema, logo após efetuar o login.

Fluxo Principal:

O usuário deve preencher a categoria com as informações necessárias que podem ser observadas logo na tela de cadastro.

Fluxos Alternativos:

O usuário pode cancelar o processo de cadastro de categorias e retornar ao menu principal através do botão "Voltar";

O usuário pode fazer uma rápida consulta através da tabela que mostras todas as categorias cadastradas;

O usuário pode remover uma categoria clicando no botão remover posicionado em cada linha correspondente na tabela de categorias.

Fluxos de Exceção:

Caso o usuário tente remover uma categoria que esteja sendo utilizada por algum recurso o sistema, irá informar a seguinte mensagem: "A categoria não pode ser removida, verifique se não há algum recurso com dependências".

Pós Condições:

Após o cadastro de uma nova categoria, caso o usuário deseje, poderá cadastrar um novo recurso relacionado à mesma através do caso de uso Manter Recurso.

Casos de Testes:

Validar os campos de cadastro de categoria;

Tentar inserir uma categoria sem valor algum;

Inserir uma categoria com os campos corretos;

Tentar remover uma categoria com recursos associados à mesma;

Remover uma categoria sem recursos associados à mesma.

4.7.2.1 – Diagrama de Sequência: Manter Categoria

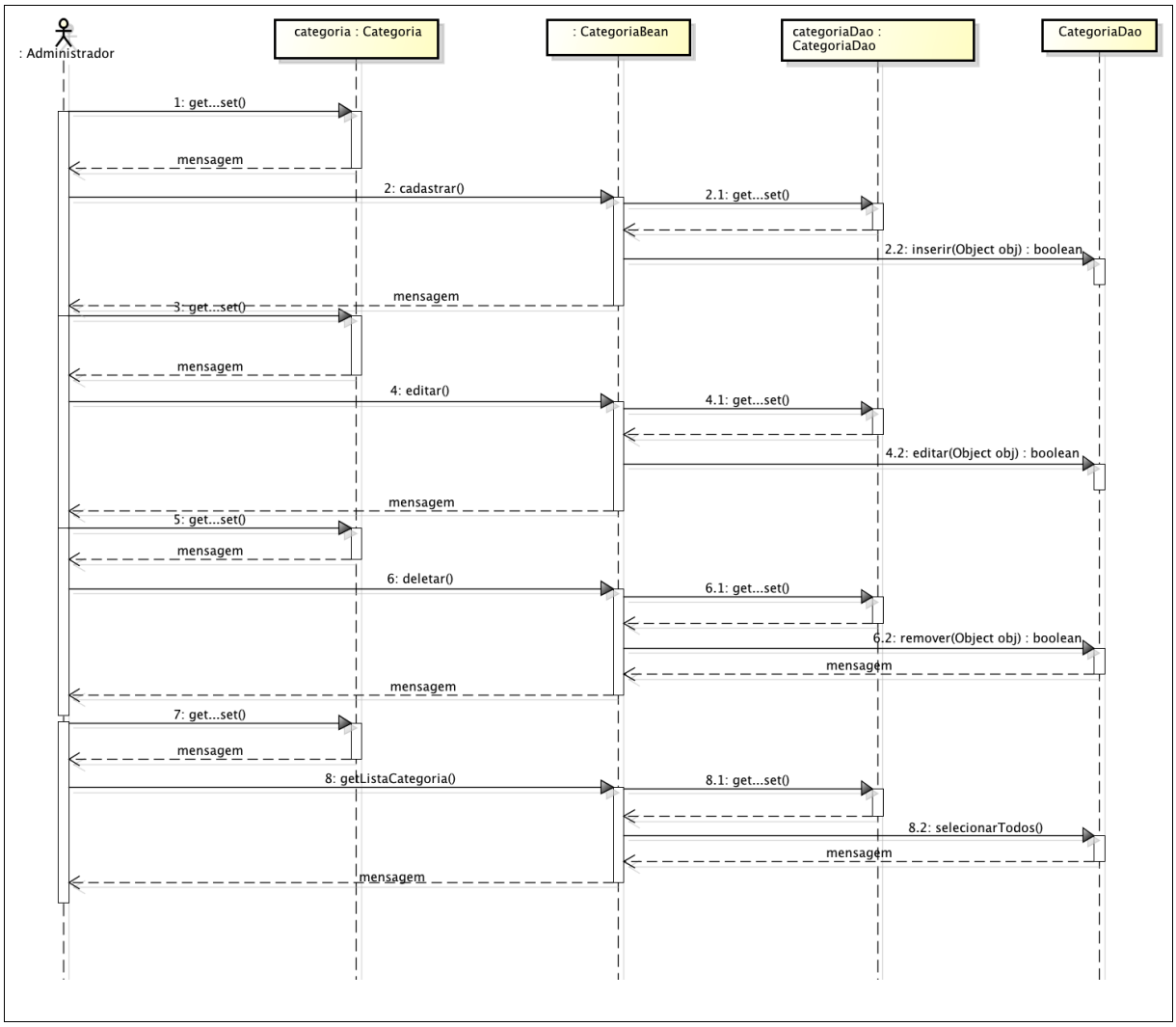

Figura 28 - Diagrama de Sequência: Manter Categoria

4.7.3 – Caso de Uso: Manter Recurso

A Figura 29 – Caso de Uso: Manter Recurso ilustra como o usuário Administrador pode gerenciar os recursos para reserva no sistema, de maneira que ele possa adicionar remover ou alterar o mesmo.

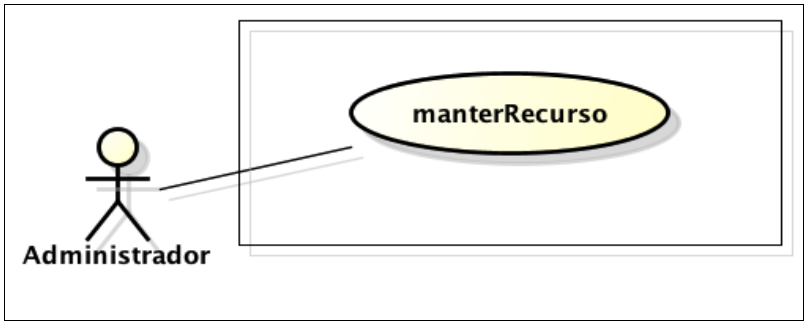

Figura 29 - Caso de Uso: Manter Recurso

Finalidade/Objetivo:

Permite ao usuário cadastrar, consultar ou remover um recurso no sistema.

Atores:

Administrador.

Evento Inicial:

O usuário deve selecionar a opção cadastro de recursos no menu principal do sistema.

Fluxo Principal:

O usuário deve preencher o recurso com as informações necessárias que podem ser observadas logo na tela de cadastro.

Fluxos Alternativos:

O usuário pode cancelar o processo de cadastro de recursos e retornar ao menu principal através do botão "Voltar";
O usuário pode fazer uma rápida consulta através da tabela que mostras todos os recursos cadastrados;
O usuário pode remover um recurso clicando no botão remover posicionado em cada linha correspondente na tabela de recursos.

Fluxos de Exceção:

Caso o usuário tente remover um recurso que esteja reservado por algum outro usuário, o sistema irá informar a seguinte mensagem: "O recurso não pode ser removido, verifique se não há algum usuário que possa ter reservado tal".

Pós Condições:

Após o cadastro de um novo recurso, caso o usuário deseje, poderá reservar um recurso através do caso de uso Reservar Recurso.

Casos de Testes:

Validar os campos de cadastro de recurso;
Tentar inserir um recurso sem valor algum;
Inserir uma recurso com os campos corretos;
Tentar remover um recurso que esteja reservado;
Remover um recurso que não esteja reservado.

4.7.3.1 – Diagrama de Sequência: Manter Recurso

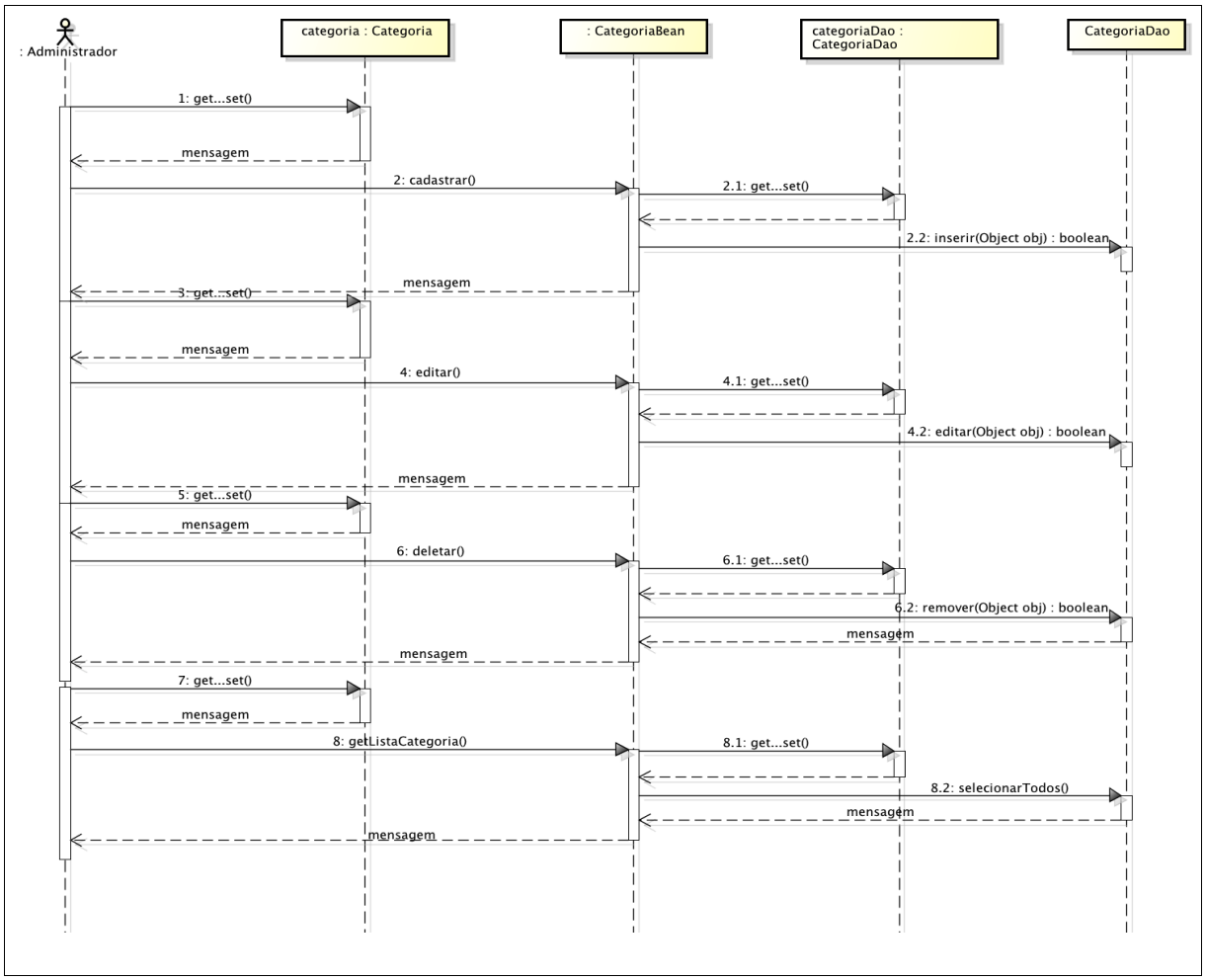

Figura 30 - Diagrama de Sequência: Manter Recurso

4.7.4 – Caso de Uso: Consultar Recurso

A Figura 31 – Caso de Uso: Consultar Recurso ilustra como os usuários Administrador e Professor, podem consultar seus recursos que foram reservados no sistema, além da possibilidade de removê-los caso seja necessário.

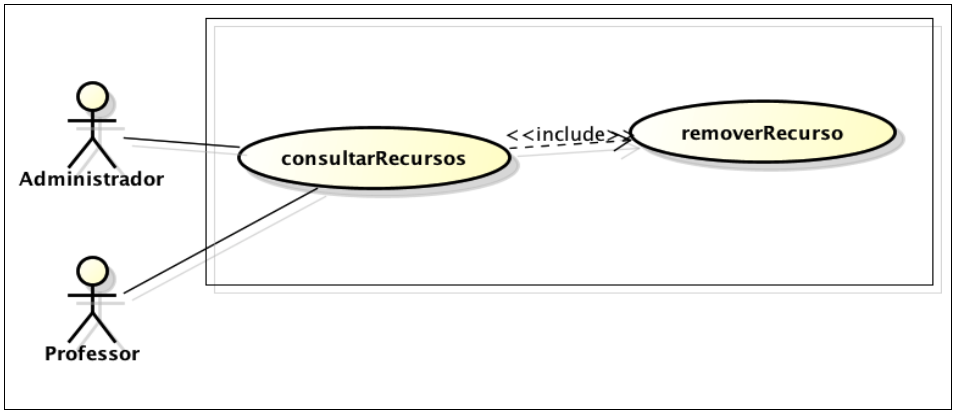

Figura 31 - Caso de Uso: Consultar Recurso

Finalidade/Objetivo:

Permite ao usuário consultar e remover recursos que foram reservados pelo mesmo.

Atores:

Administrador, Professor.

Evento Inicial:

O usuário deve selecionar a opção Consultar Recursos no menu principal do sistema, logo após efetuar o login.

Fluxo Principal:

O usuário pode filtrar os recursos que ficam localizados na tabela.

Caso o usuário queira poderá também remover um recurso, clicando no botão

remover na linha correspondente da tabela.

Fluxos Alternativos:

O usuário pode voltar para a tela principal do sistema clicando no botão "Voltar".

Fluxos de Exceção:

Não há fluxos de exceção.

Pós Condições:

Não há pós condições.

Casos de Testes:

Testar a remoção de reservas de recurso.

4.7.4.1 – Diagrama de Sequência: Consultar Recurso

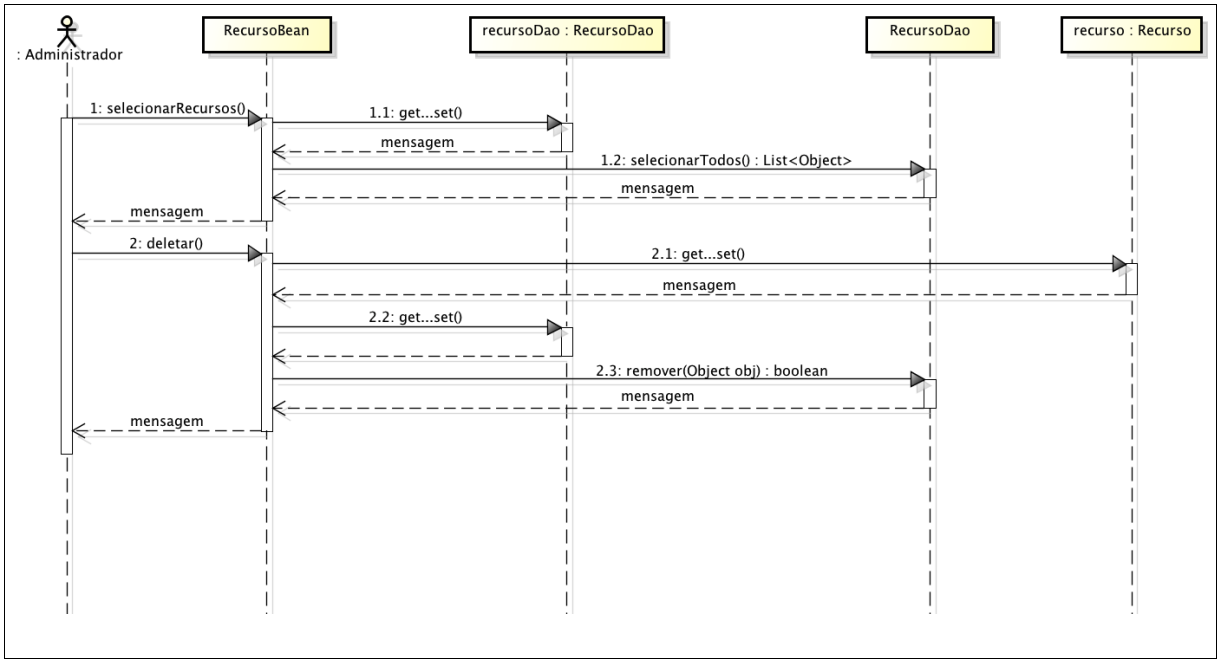

Figura 32 - Diagrama de Sequência: Consultar Recurso

4.7.5 – Caso de Uso: Consultar Reservas

A Figura 33 – Caso de Uso: Consultar Reservas ilustra como os usuários (Administrador, Professor, Aluno) podem fazer a consulta das reservas feitas no sistema.

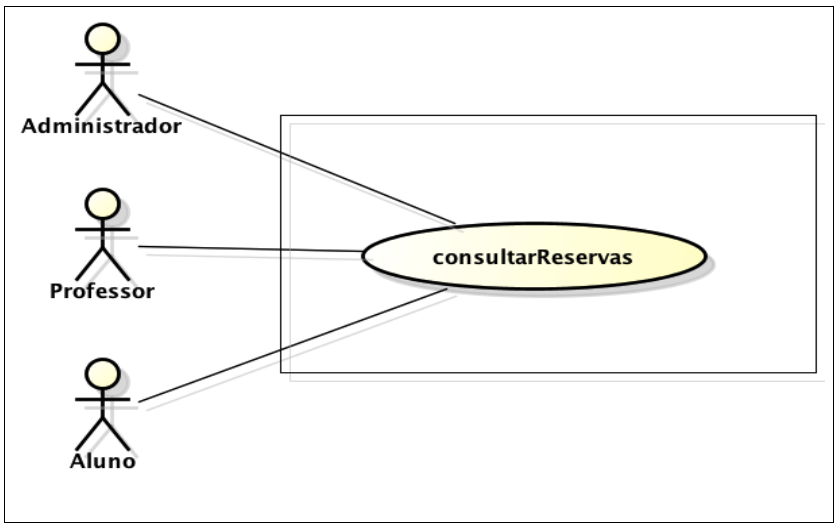

Figura 33 - Caso de Uso: Consultar Reservas

Finalidade/Objetivo:

Permite ao usuário consultar reservas no sistema.

Atores:

Administrador, Professor, Aluno.

Evento Inicial:

O usuário deve selecionar a opção consultar reservas no menu principal do sistema, logo após efetuado o login.

Fluxo Principal:

O usuário deve selecionar em qual categoria estará o recurso, através de um drop down list;

O usuário deve selecionar qual recurso correspondente à categoria selecionada, através de um drop down list;

Após as etapas acima uma tabela com relação de reservas será exibida para consulta.

Fluxos Alternativos:

O usuário pode cancelar o processo de consulta e retornar ao menu principal através do botão "Voltar".

Fluxos de Exceção:

Caso o usuário não selecione nenhuma categoria e recurso, não será possível para o sistema exibir a tabela de reservas, e assim concluir o processo.

Pós Condições:

Não há pós condições.

Casos de Testes:

Efetuar consultas com recursos diferentes.

4.7.5.1 – Diagrama de Sequência: Consultar Reserva Administrador

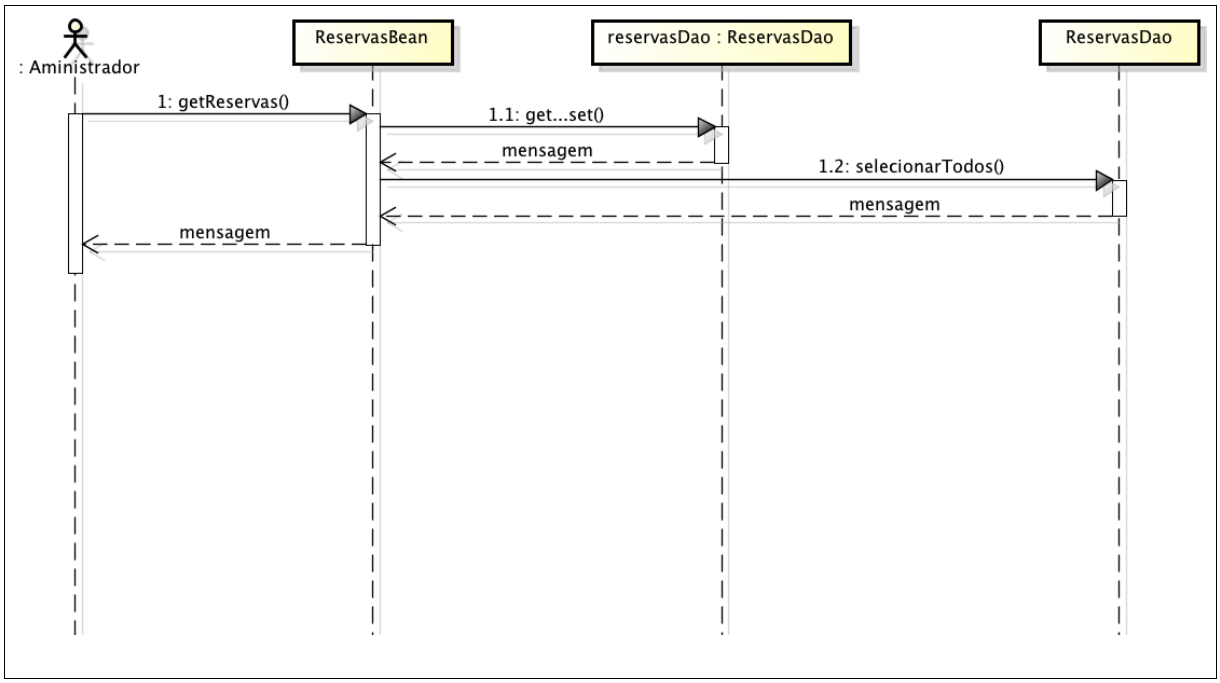

Figura 34 - Diagrama de Sequência: Consultar Reserva Administrador

4.7.5.2 – Diagrama de Sequência: Consultar Reserva Professor

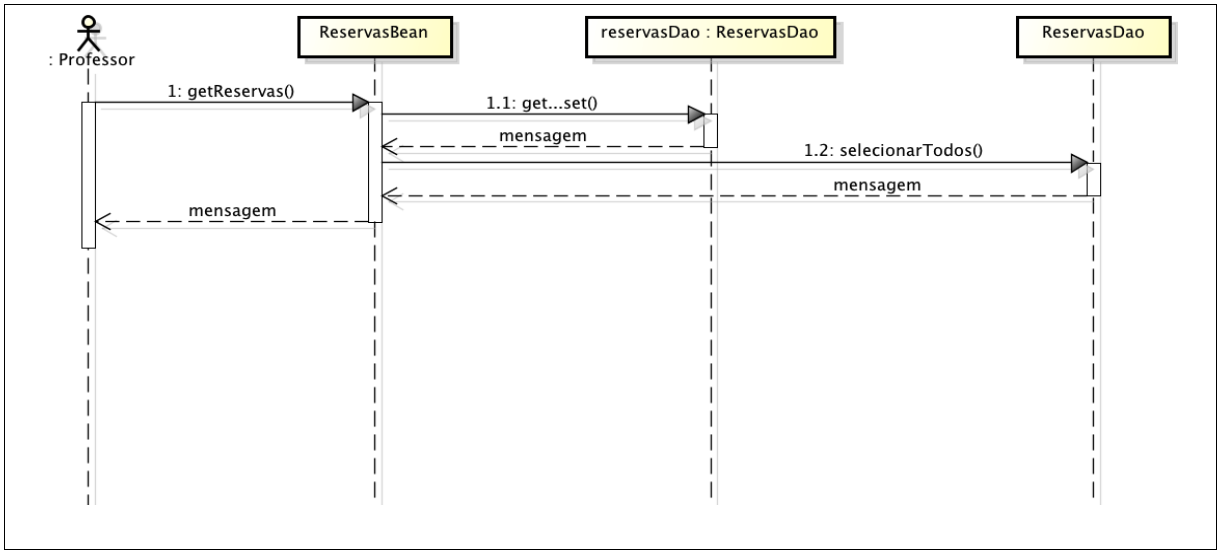

Figura 35 - Diagrama de Sequência: Consultar Reserva Professor

4.7.5.3 – Diagrama de Sequência: Consultar Reserva Aluno

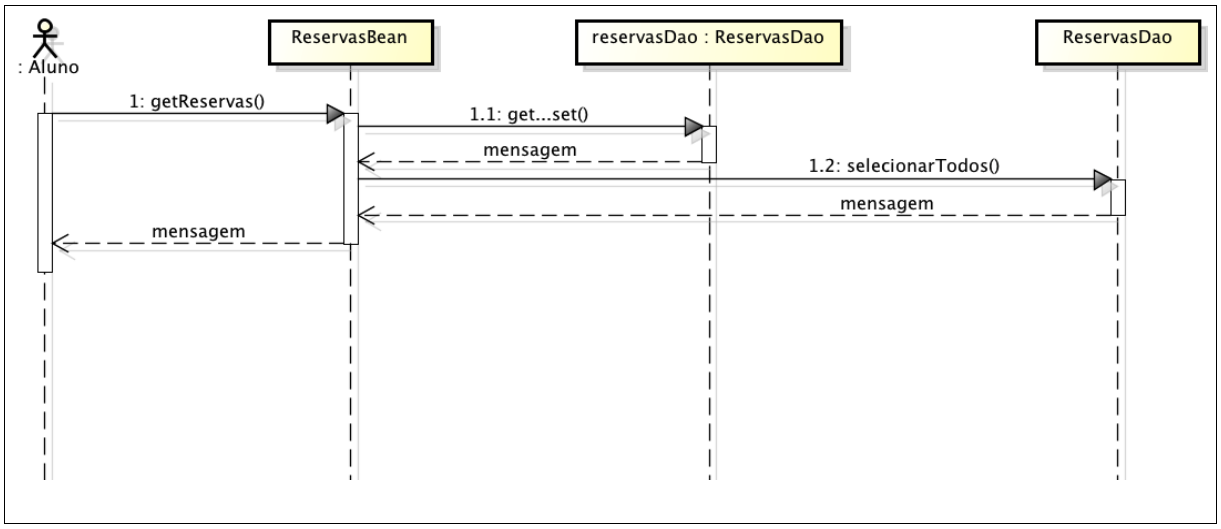

Figura 36 - Diagrama de Sequência: Consultar Reserva Aluno

4.7.6 – Caso de Uso: Login

A Figura 37 – Caso de Uso: Login ilustra como os usuários (Administrador, Professor, Aluno) podem fazer seu devido login, dessa forma podem acessar suas respectivas telas de utilização do sistema.

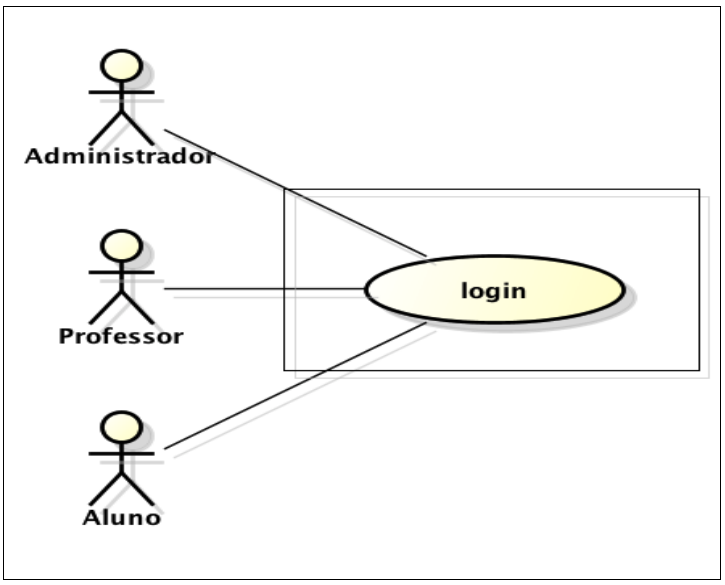

Figura 37 - Caso de Uso: Login

Finalidade/Objetivo:

Permite ao usuário efetuar login no sistema.

Atores:

Administrador, Professor, Aluno.

Evento Inicial:

O usuário deve entrar no sistema.

Fluxo Principal:

O usuário deve colocar seu nome de usuário e senha nos respectivos campos e clicar no botão "Entrar".

Logo após confirmado a conta o usuário será direcionado para a tela principal do sistema.

Fluxos Alternativos:

Não há fluxos alternativos.

Fluxos de Exceção:

Caso o usuário informe nome de usúario ou senha incorretos o sistema ira informar a seguinte mensagem: "Login ou Senha incorretos".

Pós Condições:

Após o login o usuário poderá ir para qualquer um dos casos de usos dependendo de sua permissão: Manter Recurso, Manter Categoria, Consultar Reservas, Consultar Recursos, Reservar Recurso.

Casos de Testes:

Tentar logar com informações inexistentes.

Tentar logar sem informar nada.

Logar com informações existentes.

4.7.6.1 – Diagrama de Sequência: Login

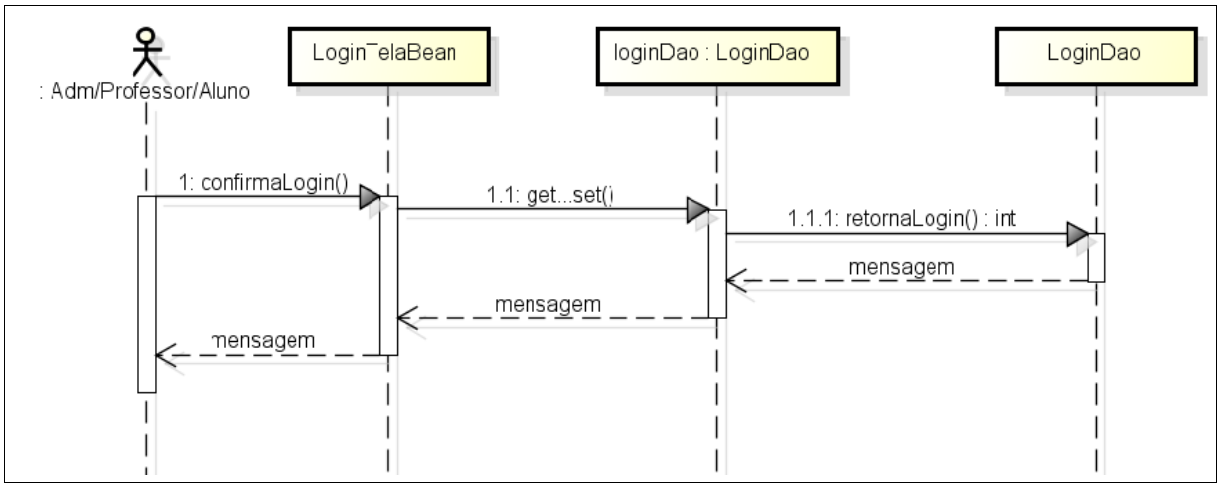

Figura 38 - Diagrama de Sequência: Login

4.7.7 – Caso de Uso: Manter Hora

A Figura 39 – Caso de Uso: Manter Hora ilustra como o usuário Administrador pode gerenciar os horários que estão disponíveis para utilizar na tela de reservas do sistema.

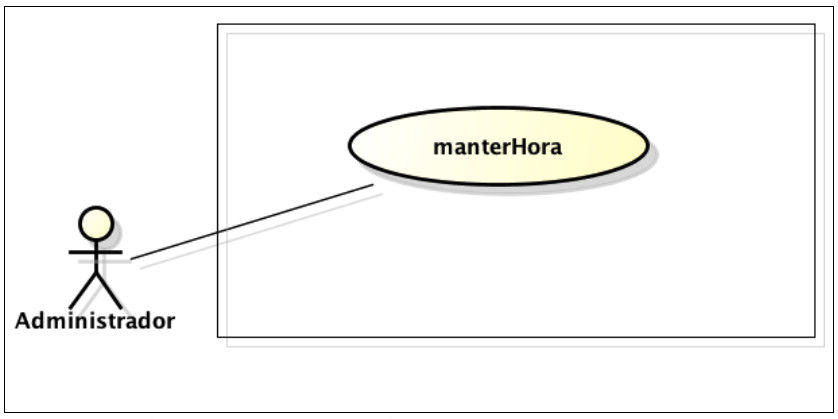

Figura 39 - Caso de Uso: Manter Hora

Finalidade/Objetivo:
Permite ao usuário cadastrar, consultar ou remover uma hora no sistema.

Atores:
Administrador.

Evento Inicial:
O usuário deve selecionar a opção cadastro de hora no menu principal do sistema, logo após efetuar o login.

Fluxo Principal:
O usuário deve preencher a hora com as informações necessárias que podem ser observadas logo na tela de cadastro.

Fluxos Alternativos:

O usuário pode cancelar o processo de cadastro de hora e retornar ao menu principal através do botão "Voltar";

O usuário pode fazer uma rápida consulta através da tabela que mostras todas as horas cadastradas;

O usuário pode remover uma hora clicando no botão remover posicionado em cada linha correspondente na tabela de horas.

Fluxos de Exceção:

Caso o usuário tente remover uma hora que esteja sendo utilizada pelo sistema, irá informar a seguinte mensagem: "A hora não pode ser removida, verifique se não há dependências".

Pós Condições:

Após o cadastro de uma nova hora, caso o usuário deseje, poderá utilizá-la na tela de reservas.

Casos de Testes:

Validar os campos de cadastro de hora;

Tentar inserir uma hora sem valor algum;

Inserir uma hora com os campos corretos;

Tentar remover uma hora com dependências;

Remover uma hora sem dependências.

4.7.7.1 – Diagrama de Sequência: Manter Hora

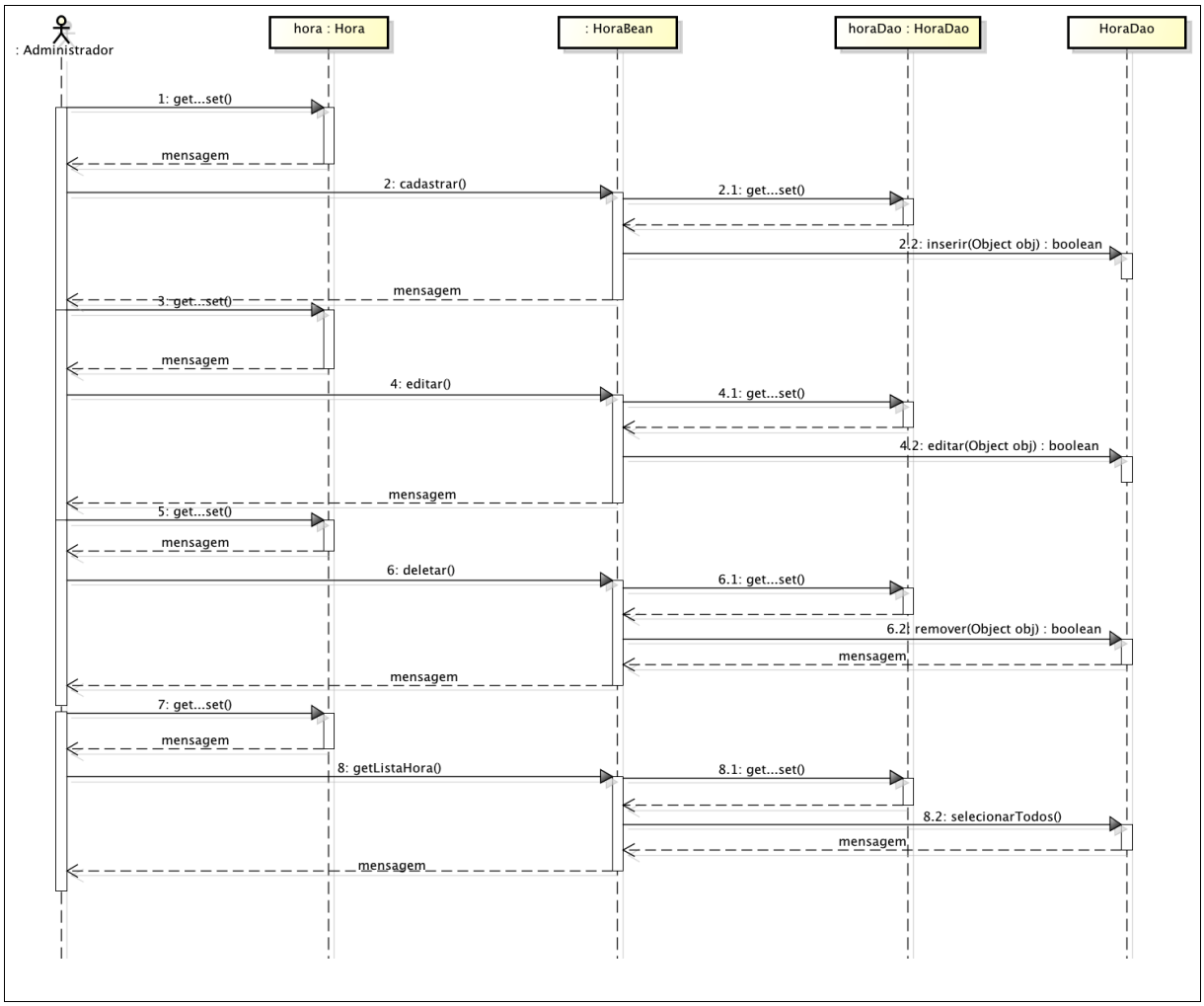

Figura 40 - Diagrama de Sequência: Manter Hora

4.7.8 – Manter Data

A Figura 41 – Caso de Uso: Manter Data, ilustra como o usuário Administrador pode gerenciar as datas que não devem estar disponíveis na tela de reservas do sistema, dessa forma somente as datas não cadastradas poderão ser utilizadas para a reserva dos recursos.

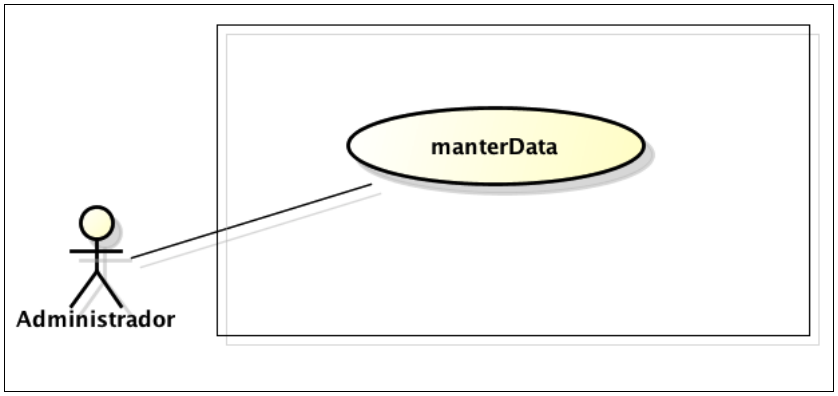

Figura 41 - Caso de Uso: Manter Data

Finalidade/Objetivo:

Permite ao usuário cadastrar, consultar ou remover uma data no sistema.

Atores:

Administrador.

Evento Inicial:

O usuário deve selecionar a opção cadastro de data no menu principal do sistema, logo após efetuar o login.

Fluxo Principal:

O usuário deve preencher a data com as informações necessárias que podem ser observadas logo na tela de cadastro.

Fluxos Alternativos:

O usuário pode cancelar o processo de cadastro de data e retornar ao menu principal através do botão "Voltar";

O usuário pode fazer uma rápida consulta através da tabela que mostras todas as datas cadastradas;

O usuário pode remover uma data clicando no botão remover posicionado em cada linha correspondente na tabela de horas.

Fluxos de Exceção:

Caso o usuário tente remover uma data que esteja sendo utilizada pelo sistema, irá informar a seguinte mensagem: "A data não pode ser removida, verifique se não há dependências".

Pós Condições:

Após o cadastro de uma nova data, caso o usuário deseje, poderá utilizá-la na tela de reservas.

Casos de Testes:

Validar os campos de cadastro de data;

Tentar inserir uma data sem valor algum;

Inserir uma data com os campos corretos;

Tentar remover uma data com dependências;

Remover uma data sem dependências.

4.7.8.1 – Diagrama de Sequência: Manter Data

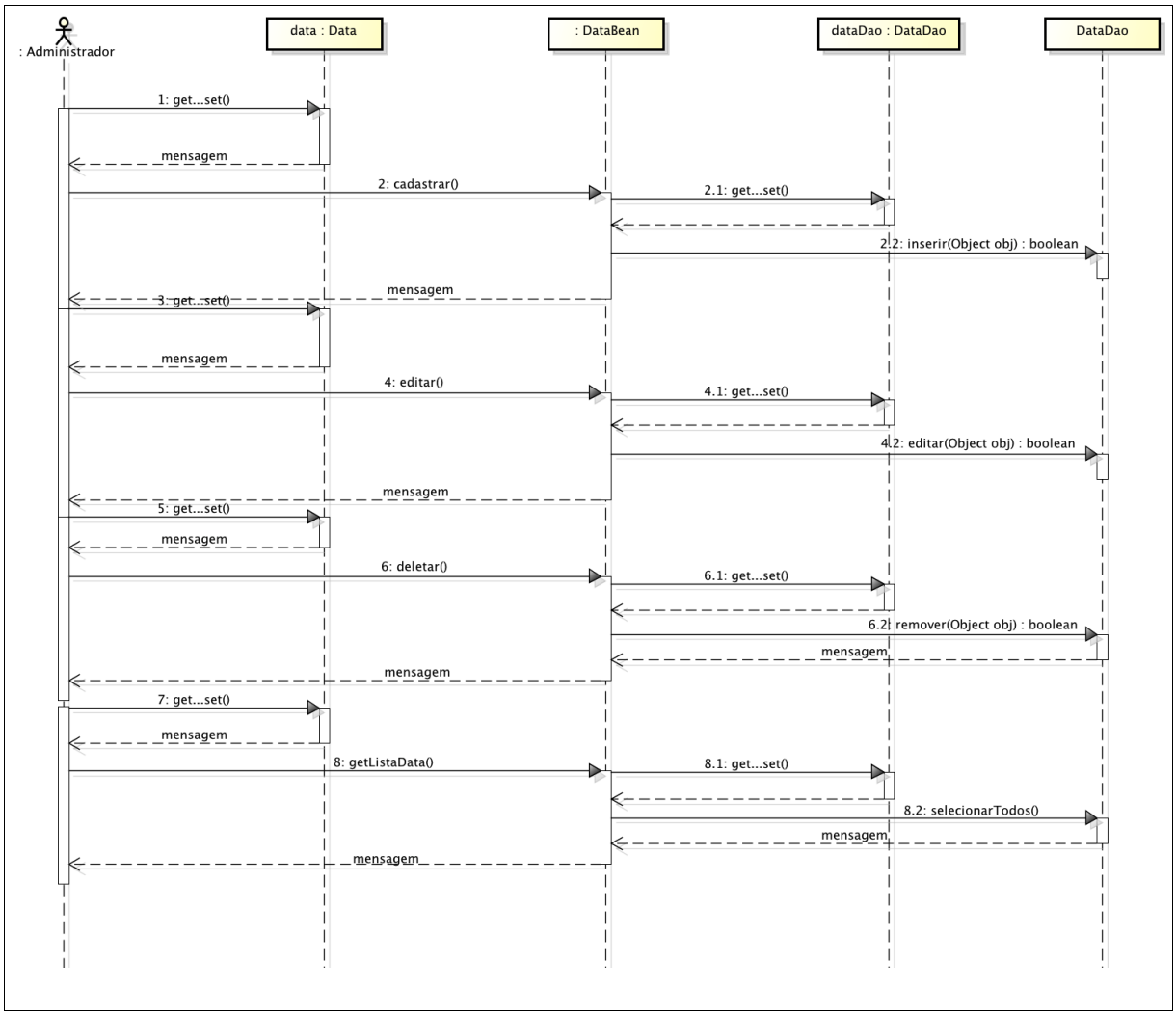

Figura 42 - Diagrama de Sequência: Manter Data

4.7.9 – Caso de Uso: Manter Login

A Figura 43 – Caso de Uso: Manter Login ilustra como o usuário Administrador pode gerenciar o cadastro de todos os *login's* de acesso ao sistema, incluindo cadastro, remoção, e atualização dos mesmos.

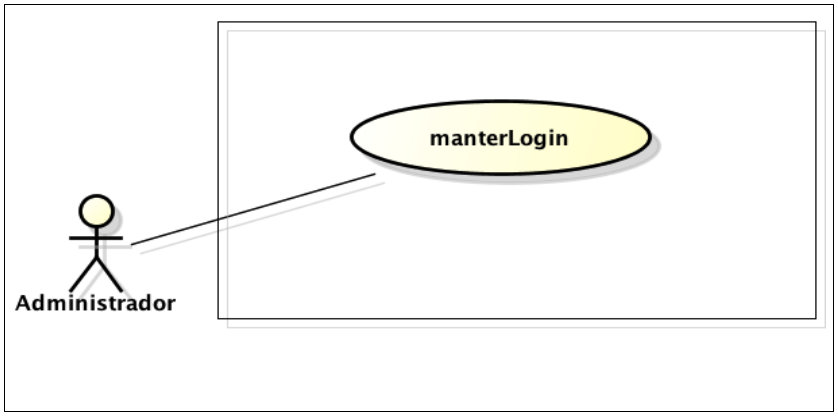

Figura 43 - Caso de Uso: Manter Login

Finalidade/Objetivo:

Permite ao usuário cadastrar, consultar ou remover um login no sistema.

Atores:

Administrador.

Evento Inicial:

O usuário deve selecionar a opção cadastro de login no menu principal do sistema, logo após efetuar o login.

Fluxo Principal:

O usuário deve preencher o login com as informações necessárias que podem ser observadas logo na tela de cadastro.

Fluxos Alternativos:

O usuário pode cancelar o processo de cadastro de login e retornar ao menu principal através do botão "Voltar";

O usuário pode fazer uma rápida consulta através da tabela que mostras todos os login's cadastrados;

O usuário pode remover um login clicando no botão remover posicionado em cada linha correspondente na tabela de login.

Fluxos de Exceção:

Caso o usuário tente cadastrar um login sem senha ou sem o próprio nome de usuário o sistema retornará a seguinte mensagem: "O login não pode ser criado, verifique se não há algum campo vazio ou inválido".

Pós Condições:

Após o cadastro do login o mesmo poderá ser utilizado na próxima tentativa de acesso ao sistema.

Casos de Testes:

Validar os campos de cadastro de login;

Tentar inserir uma login sem valor algum;

Inserir uma login com os campos corretos;

Adicionar um login novo;

Remover um login existente;

4.7.9.1 – Diagrama de Sequência: Manter Login

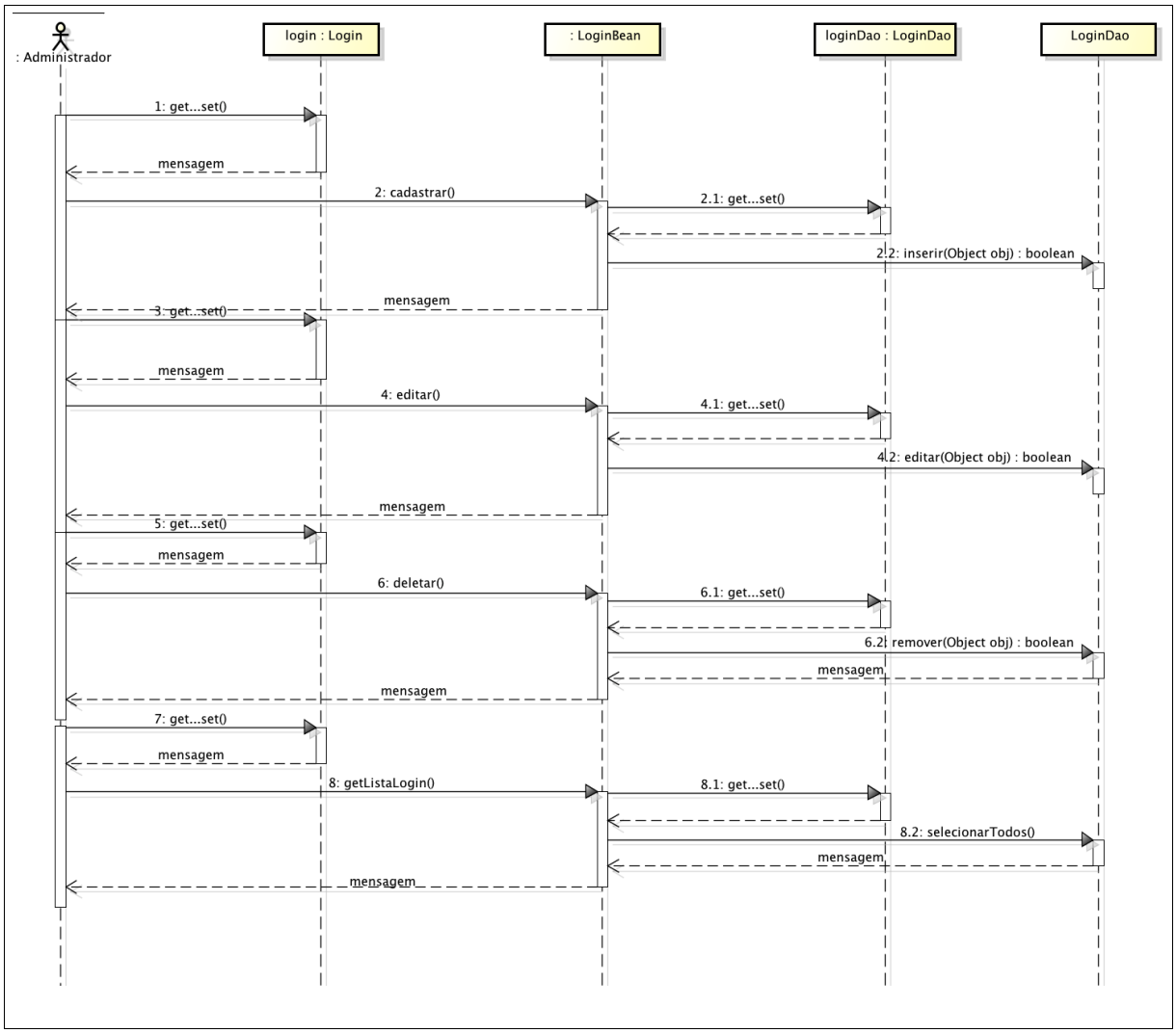

Figura 44 - Diagrama de Sequência: Manter Login

4.8 – Diagramas de Classe

Neste subcapítulo serão mostrados os diagramas de classes que compõem a especificação do sistema. Com o diagrama de classes ficará muito mais fácil de entender a estrutura das classes que serão criadas no desenvolvimento do sistema.

4.8.1 – Diagrama de Classe: dao

A Figura 45 – Diagrama de Classe: dao ilustra as classes que serão criadas no pacote 'dao. Elas serão responsáveis direta ou indiretamente por todas as transações com o banco de dados, ou seja, insert, delete, update e as demais operações, serão implementadas nessa camada do software.

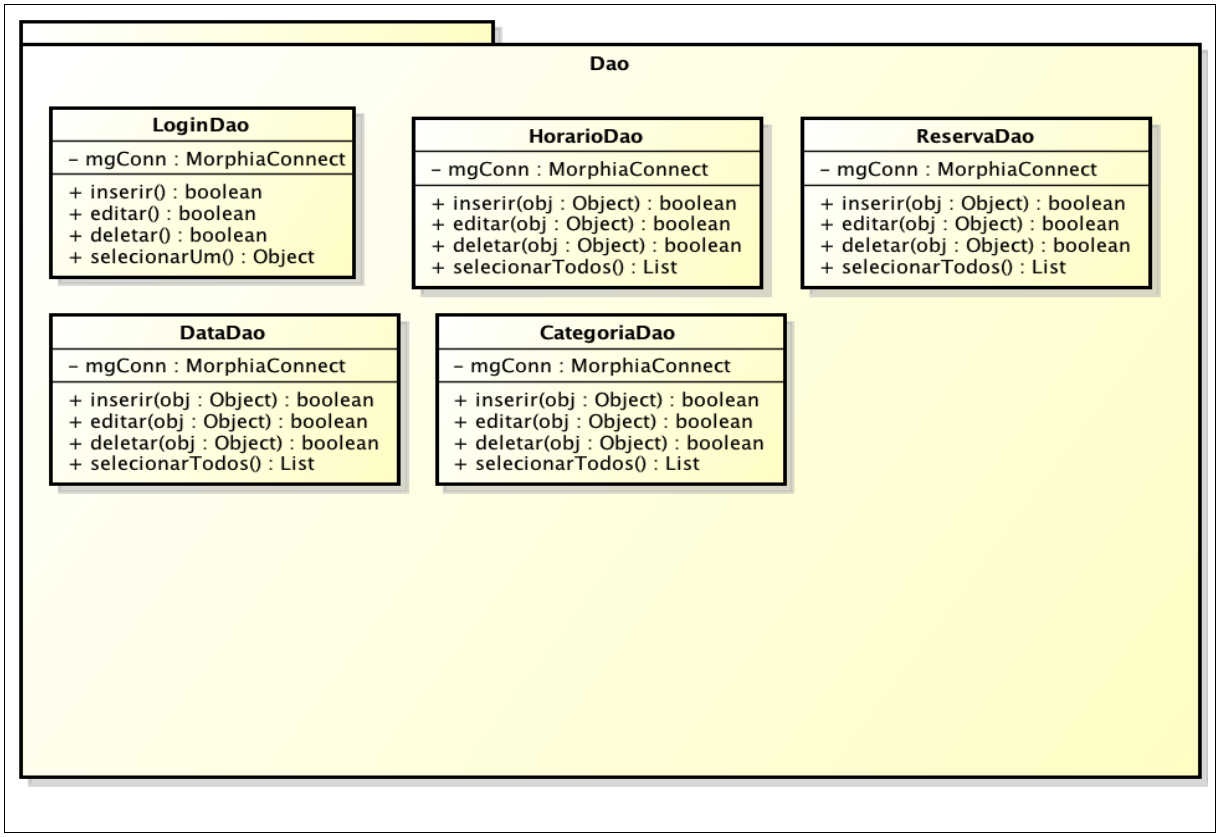

Figura 45 - Diagrama de Classe: Dao

4.8.2 – Diagrama de Classe: modelo

A Figura 46 – Diagrama de Classe: modelo ilustra as classes que serão criadas no pacote 'modelo. Elas serão responsáveis por garantir o relacionamento do software, de forma que a estrutura do MongoDB será determinada através da criação dessas.

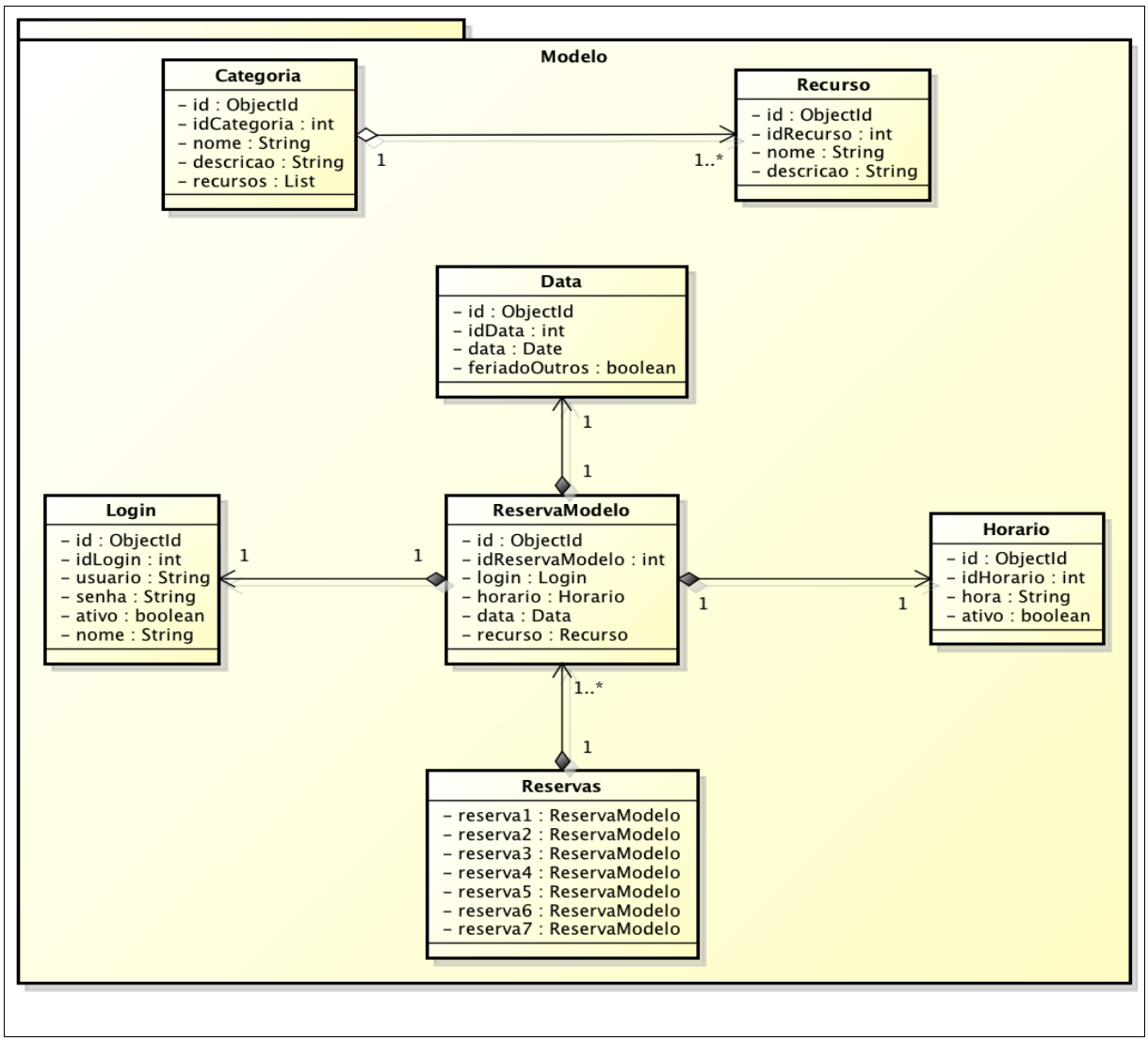

Figura 46 - Diagrama de Classe: Modelo

4.8.3 – Diagrama de Classe: view

A Figura 47 – Diagrama de Classe: view ilustra as classes que serão criadas no pacote 'view'. Caso elas não existissem, seria impossível garantir uma troca de informações entre as telas do sistema ("página web"), isso acontece porque é aqui que se encontram os *ManagedBeans,* que conectam a interface da páginas *web* com a lógica de negócio das classes do pacote 'dao'.

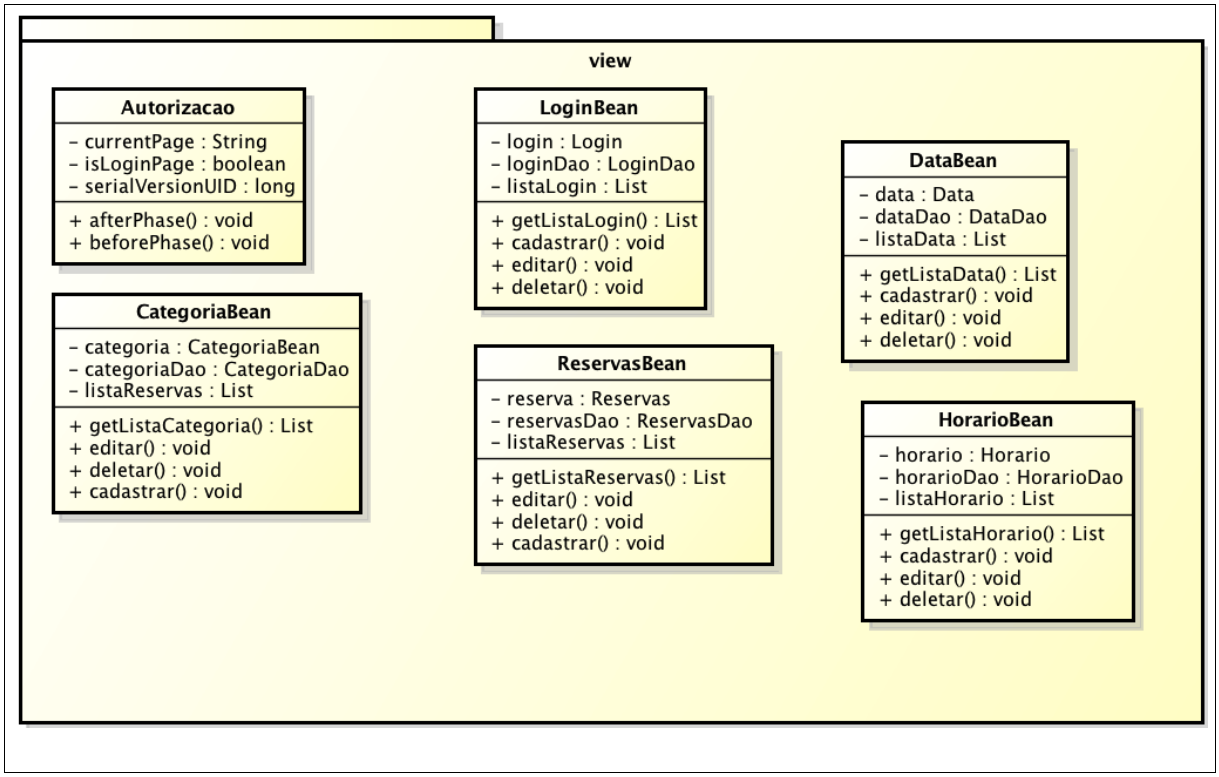

Figura 47- Diagrama de Classe: View

4.8.4 – Diagrama de Classe: Connection

A Figura 48 – Diagrama de Classe: conexão ilustra as classes que serão criadas no pacote 'conexão. Sem sua utilização, as operações com o MongoDB necessitariam de um esforço maior com as classes do pacote dao, já que ela permite um melhor gerenciamento das conexões com o banco de dados.

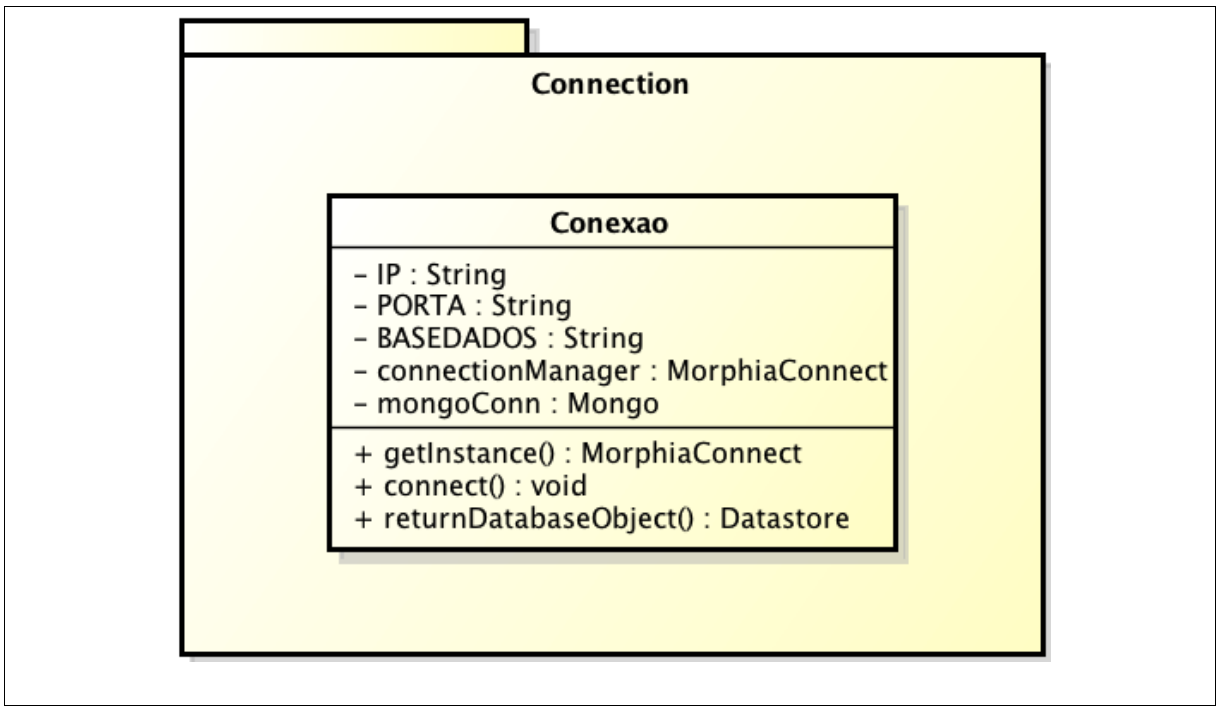

Figura 48 - Diagrama de Classe: Connection

4.9 – Diagramas de Atividade

Neste subcapítulo serão mostrados os diagramas de atividade que compõem a especificação do sistema.

4.9.1 – Diagrama de Atividade: Administrador

A Figura 49 – Diagrama de Atividade: Administrador, ilustra o fluxo de interação que o usuário administrador possui ao se logar no sistema, isso inclui todos os casos de usos abordados no capítulo 4.3 referentes ao login do administrador.

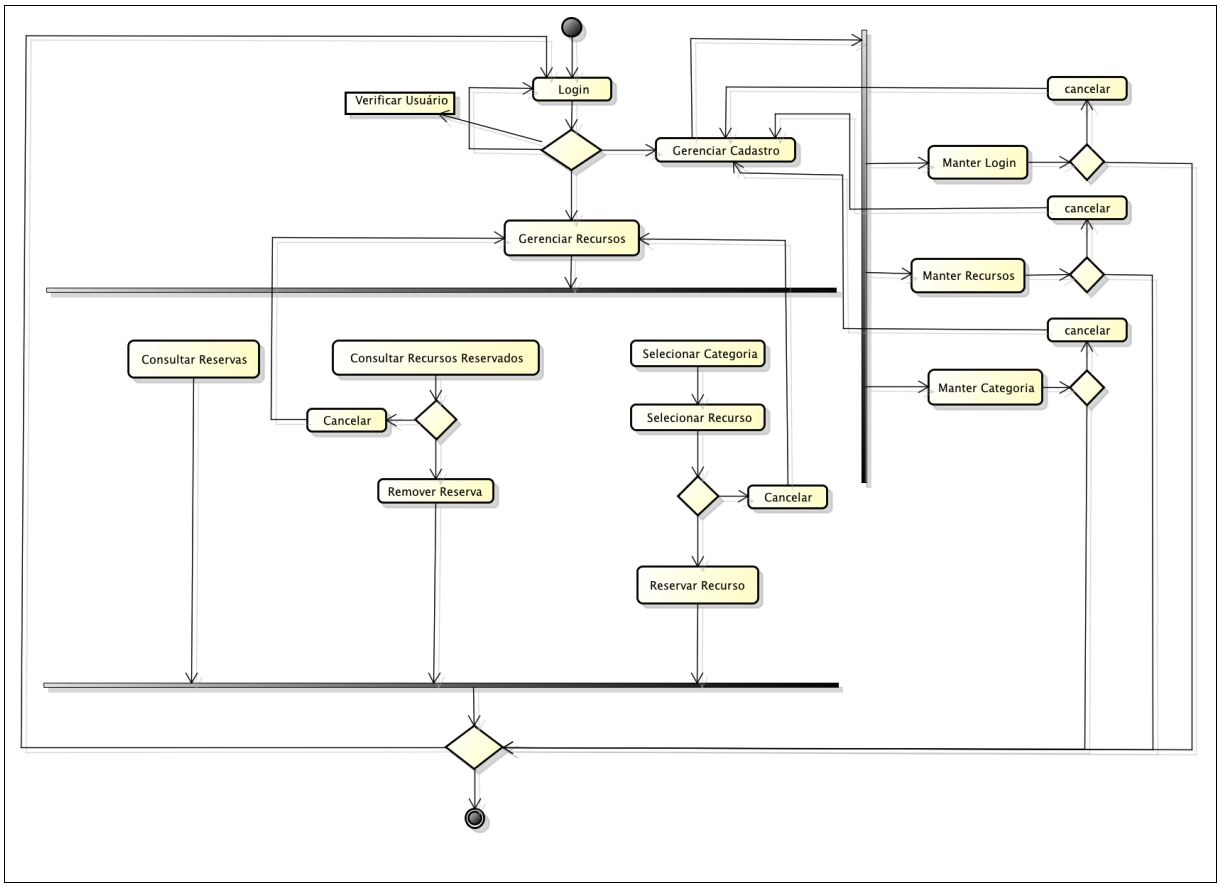

Figura 49 - Diagrama de Atividade: Administrador

4.9.2 – Diagrama de Atividade: Professor

A Figura 50 – Diagrama de Atividade: Professor ilustra o fluxo de interação que o usuário professor possui ao se logar no sistema, isso inclui todos os casos de usos abordados no capítulo 4.3 referentes ao login do professor.

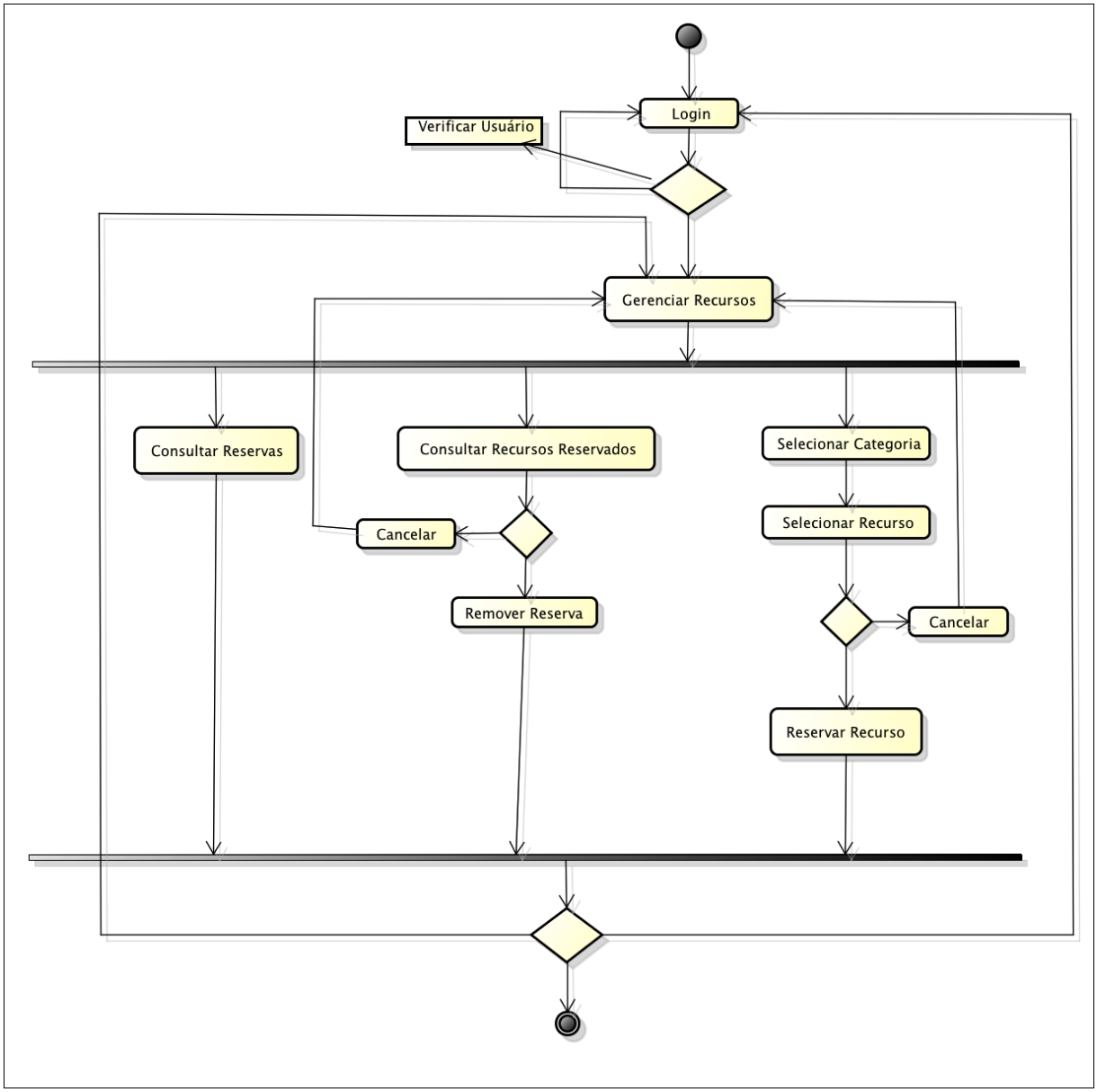

Figura 50 - Diagrama de Atividade: Professor

4.9.3 – Diagrama de Atividade: Aluno

A Figura 51 – Diagrama de Atividade: Aluno ilustra o fluxo de interação que o usuário aluno possui ao se logar no sistema, isso inclui todos os casos de usos abordados no capítulo 4.3 referentes ao login do aluno.

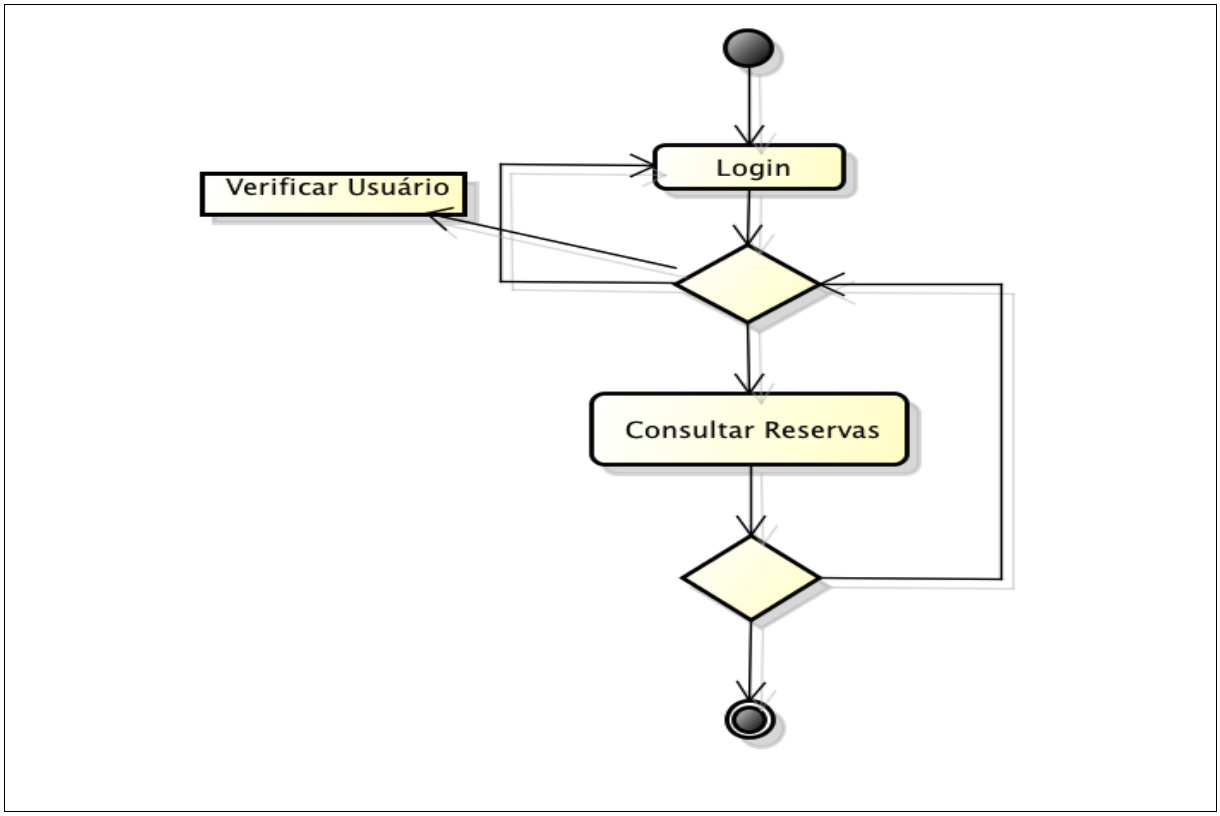

Figura 51 - Diagrama de Atividade: Aluno

5 – CONCLUSÃO

O trabalho proposto apresentou-se através de uma pesquisa sobre um banco de dados do tipo *NoSQL*, do qual se difere dos tradicionais, tanto na implementação, quanto gerenciamento do mesmo. Por meio dos autores abordados foi demonstrado sua estrutura de armazenamento, manipulação dos dados bem como características peculiares de bancos de dados, que demonstram o principio '*NoSQL'.*

Com o uso do MongoDB observou-se que a velocidade de transações no banco de dados é notavelmente alta, e a facilidade de manipulação dessas informações se deve ao fato do mesmo adequar-se a estrutura criada na fase de programação, já que a estrutura do banco somente confirmará à partir do momento em que as próprias classes definidas permitirem tal função.

Obteve-se uma forte ajuda da comunidade de desenvolvedores que também utilizam *NoSQL* em projetos já operantes, além do total apoio da empresa desenvolvedora, no que diz respeito solução de dúvidas e suporte ao cliente.

A linguagem de programação Java provou-se apta a ser utilizada em projetos com MongoDB, tanto de grande como pouca escalabilidade, e seu *driver* de conexão com o banco também possui atualização continua para melhor desempenho.

Tornou-se possível através do Morphia uma rápida implementação do sistema de reservas, e também se observou uma facilidade em buscas totalmente orientadas a objeto, quando necessários critérios mais complexos.

Portanto a utilização do MongoDB agiliza notavelmente o processo de desenvolvimento de software, já que a fase de estruturação e definição do banco de dados, fica por conta do modelo definido totalmente orientada a objetos, que são as classes e suas relações no sistema.

Para trabalhos futuros seria interessante o estudo do MongoDB utilizando ferramentas para mineração de dados como o "*apache Hadoop"*, utilização em clusterização dos dados por meio de seus *replica-sets* e *sharding*, ou ate mesmo um estudo sobre seus métodos de segurança como: *journaling*, acesso restrito de usuários ou *backup* dos dados.

REFERENCIAS BIBLIOGRÁFICAS

Banker, Kyle. **MongoDB in Action.** Shelter Island: Manning, 2012.

Chodorow, Kristina; Michael Dirolf. **MongoDB The Definitive Guide,** First Edition. Gravenstein Highway North: O'Reilly, 2010.

Documentação MongoDB e Tutorial para implementação em linguagem Java. Disponível em: <http://www.mongodb.org/display/DOCS/Java+Language+Center>. Acesso em: 5 de março 2012.

IBM - D'Emic, John. **Persistência de modelo de domínio com Morphia e MongoDB.** 2011. Disponível em: <http://www.ibm.com/developerworks/br/library/j-morphia/>. Acesso em: 7 de julho 2012.

MongoDB - 10gen. Java Driver Concurrency. 2012. Disponível em: <http://www.java.com/pt_BR/download/faq/whatis_java.xml>. Acesso em: 3 de junho 2012.

Oracle. **O que é a tecnologia Java e por que é necessária?**. 2012. Disponível em: <http://www.java.com/pt_BR/download/faq/whatis_java.xml>. Acesso em: 27 de maio 2012.

Tiwari, Shashank. **Professional NoSQL**. Indianapolis: John Wiley & Sons, 2011.

ANEXO 1

A Figura 1 – Tela de Login ilustra a interface que o usuário se depara na tentativa de fazer login no sistema.

Figura 1 – Tela de Login

A Figura 2 – Menu do Sistema ilustra o menu principal de onde o usuário poderá escolher algum link de acesso para as outras áreas.

Figura 2 – Menu do Sistema

A Figura 3 – Manter Data ilustra a interface que o usuário se depara na tentativa de manter as datas do sistema.

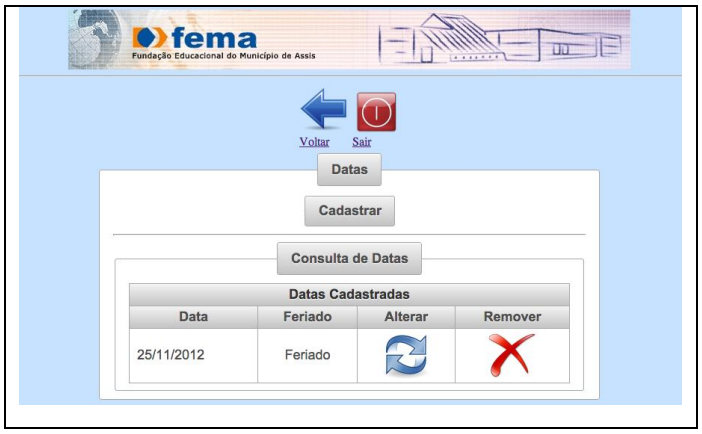

Figura 3 – Manter Data

A Figura 4 – Manter Horários ilustra a interface que o usuário se depara na tentativa de manter os horários do sistema.

Figura 4 – Manter Horários

A Figura 5 – Manter Categoria ilustra a interface que o usuário se depara na tentativa de manter as categorias do sistema.

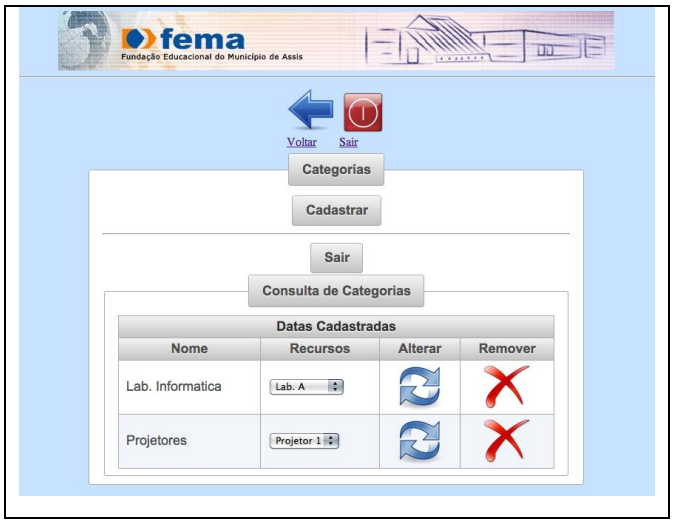

Figura 5 – Manter Categoria

A Figura 6 – Manter Recursos ilustra a interface que o usuário se depara na tentativa de manter os recursos do sistema.

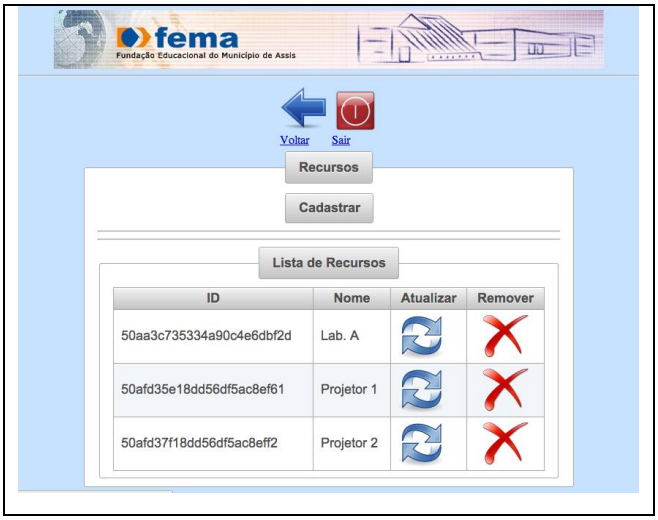

Figura 6 – Manter Recursos

A Figura 7 – Reservas Parte1 ilustra a interface que o usuário se depara na tentativa de reservar recursos no sistema. A tela possui duas etapas, nessa primeira é necessário escolher a categoria e os recursos para exibir a tabela de agendamento.

Figura 7 – Reservas Parte1

A Figura 8 – Reservas Parte2 nessa segunda etapa da tela de reservas é ilustrado como o usuário pode agendar os recursos pela tabela, além da possibilidade de consultar os que já estão reservados.

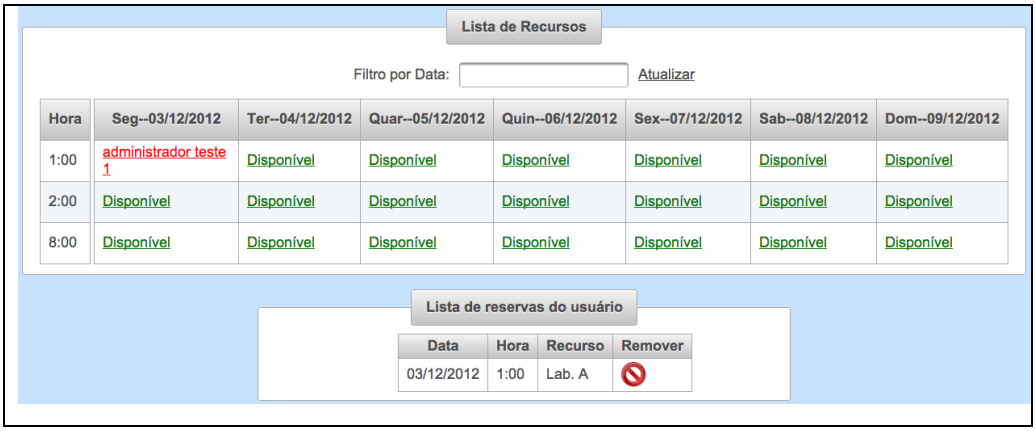

Figura 8 – Reservas Parte2

ANEXO 2

As Figuras 1, 2 e 3 ilustram como o código principal do sistema está implementado. Esse código é responsável pelo posicionamento das reservas do Anexo 1 – Figura 8.

```
79      public List<Reservas> selecionarTodos(java.util.Date inicio) {
80
81          List<Reservas> variasLinhasDaTabela;
            List<ReservaModelo> listaReservas = null;
83          String inicioString = Util.converterDataUtilToString(inicio);
84          int inicioInt = Util.converterDataStringToDias(inicioString);
85
86          try {
87
88              //Cria objetos Datas para comparacoes posteriores
89              Datas data1;
90              Datas data2;
91
92              //carrega a lista de datas que nao poderao ser utilizadas para agendamento
93              List<Datas> datas;
94              DataDao dataDao = new DataDao();
95              datas = dataDao.selecionarTodos();
96
97              //carrega a lista de horarios que poderao ser utilizados para agendamento
98              List<Horario> horarios;
99              HorarioDao horaDao = new HorarioDao();
100             horarios = horaDao.selecionarTodos();
101
102             variasLinhasDaTabela = new ArrayList<Reservas>();
103
104             for (int i = 0; i < horarios.size(); i++) {
105                 vazio = new Reservas();
106                 int j = 0, l = 0;
107                 while (j < 7) {
108
109                     java.util.Date dataParaAux = new java.util.Date();
110                     Datas dataAux = new Datas(new ObjectId(), dataParaAux,
111                             Util.converterDataUtilToString(dataParaAux),
                            Util.converterDataStringToDias(Util.converterDataUtilToString(dataParaAux)), true);
113                     dataAux.getData().setDate(inicio.getDate() + l);
114                     dataAux.setDataConvertida(Util.converterDataUtilToString(dataAux.getData()));
115                     boolean teste = true;
116
117                     for (int k = 0; k < datas.size(); k++) {
118
119                         if (dataAux.getDataConvertida().equals(datas.get(k).getDataConvertida())) {
120
121                             teste = false;
122                         }
123                     }
124                     if (teste) {
```

Figura 1 – Tela Reservas Parte1

```
                if (teste) {

                    nada = new ReservaModelo();
                    nada.setData(dataAux);
                    nada.setHorario(horarios.get(i));
                    nada.setLogin(new Login());
                    nada.setRecurso(new Recurso());
                    vazio.setVazio(j, nada);
                    vazio.setHorarioNome(horarios.get(i).getHora());
                    j++;
                }

                l++;
            }
            variasLinhasDaTabela.add(vazio);
        }

        //conecta e mapeia a classe para a memória
        mgConn.connect();
        Morphia morphia = new Morphia();
        morphia.map(ReservaModelo.class);;
        Datastore ds = mgConn.returnDatabaseObject(morphia);

        //Retorna uma lista com todas as reservas inseridas no mongodb
        Query<ReservaModelo> rq = ds.createQuery(ReservaModelo.class).filter("data.dataDias >=", inicioInt).order("horario.minutosTotais");
        listaReservas = rq.asList();

        for (ReservaModelo reserva : listaReservas) {

            data1 = reserva.getData();
            //Datas testeData = new Datas();


            for (int j = 0; j < variasLinhasDaTabela.size(); j++) {

                if (variasLinhasDaTabela.get(j).getHorarioNome()
                        .equals(reserva.getHorario().getHora())) {
                    if (variasLinhasDaTabela.get(0).getReserva1().getData().getDataConvertida()
                            .equals(data1.getDataConvertida())) {
                        variasLinhasDaTabela.get(j).setAutomatico(0, reserva);
                    }
                    if (variasLinhasDaTabela.get(0).getReserva2().getData().getDataConvertida()
                            .equals(data1.getDataConvertida())) {
                        variasLinhasDaTabela.get(j).setAutomatico(1, reserva);
                    }
                    if (variasLinhasDaTabela.get(0).getReserva3().getData().getDataConvertida()
                            .equals(data1.getDataConvertida())) {
                        variasLinhasDaTabela.get(j).setAutomatico(2, reserva);
                    }
```

Figura 2 – Tela Reservas Parte2

```
                    if (variasLinhasDaTabela.get(0).getReserva4().getData().getDataConvertida()
                            .equals(data1.getDataConvertida())) {
                        variasLinhasDaTabela.get(j).setAutomatico(3, reserva);
                    }
                    if (variasLinhasDaTabela.get(0).getReserva5().getData().getDataConvertida()
                            .equals(data1.getDataConvertida())) {
                        variasLinhasDaTabela.get(j).setAutomatico(4, reserva);
                    }
                    if (variasLinhasDaTabela.get(0).getReserva6().getData().getDataConvertida()
                            .equals(data1.getDataConvertida())) {
                        variasLinhasDaTabela.get(j).setAutomatico(5, reserva);
                    }
                    if (variasLinhasDaTabela.get(0).getReserva7().getData().getDataConvertida()
                            .equals(data1.getDataConvertida())) {
                        variasLinhasDaTabela.get(j).setAutomatico(6, reserva);
                    }

                }



            }
        }


    } catch (MongoException ex) {

        ex.printStackTrace();
        System.out.println("Problemas com Class: RecursoDao | Metodo: selecionarTodos");
        variasLinhasDaTabela = null;
    }
    return variasLinhasDaTabela;
}
```

Figura 3 – Tela Reservas Parte3