



INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
UNIVERSIDADE DE SÃO PAULO

Bacharelado em Ciências da Computação

Library Mapper

Thiago Gomes Toledo
Supervisor: Prof. Dr. Marcelo Finger

São Paulo - SP

Segundo semestre de 2011

Agradeço a minha mãe Mara Lucia Gomes Toledo e a meu pai João Batista Toledo por terem me dado a oportunidade de realizar meu sonho; A OPUS-Software por proporcionar diversos recursos para a realização desse sonho; A minha esposa Lucianna Thomaz Almeida Toledo por me dar conforto, ajuda e sábias dicas; A meu filho que me traz alegria só por imaginar seu rosto e a Deus por ter me dado todos e tudo que sempre precisei.

1

¹Para mais informações sobre essa monografia, apresentação, código, etc.. acesse o site: www.linux.ime.usp.br/renoir/mac499/proposta.html

Sumário

1	Introdução	3
2	Parte Objetiva	3
2.1	Objetivo	3
2.2	Problema	3
2.3	Solução	3
2.4	Conceitos e tecnologias estudadas	4
2.4.1	O que são Qr-Codes?	4
2.4.2	Grid Graphs	5
2.4.3	Breadth-First Search (BFS)	7
2.4.4	Bi-directional BFS	8
2.4.5	pgAdmin - Interface gráfica para gerenciamento do banco de dados PostgreSQL	8
2.4.6	PostgreSQL	9
2.4.7	Mybatis - Mapeando SQL para JAVA	12
2.4.8	JSP, Servlets e Apache Tomcat	13
2.4.9	JSF2 - Managed Beans	14
2.4.10	JavaScript, jQuery e JSON	14
2.4.11	HTML5 - Canvas	14
2.4.12	Desenvolvimento mobile	15
2.4.13	Integração Contínua	15
2.5	Library Mapper, o cliente	16
2.5.1	Construindo o seu mapa Web	16
2.5.2	JavaScript para Nodes e Nodes para o Banco de Dados	18
2.6	Library Mapper, o usuário	19
2.6.1	Identificando o usuário	19
2.6.2	A consulta e a ligação com a biblioteca	20
2.6.3	Lista de livros	21
2.6.4	Montando o mapa com HTML5 e canvas	21
2.6.5	A busca pela estante	22
2.7	Caso de Uso	23
2.8	Resultados e produtos obtidos	23
2.9	Conclusões	23
3	Parte Subjetiva	24
3.1	Desafios e frustrações	24
3.2	Relações entre Disciplinas	24
3.3	Os próximos passos	25

1 Introdução

Como frequentador da biblioteca do Instituto de Matemática e Estatística da USP desde 2008 e trabalhando na biblioteca do Instituto de Geociências da USP em 2010-2012, pude notar que as reclamações numa biblioteca sobre a dificuldade em se achar um livro lideram o ranking de problemas apontados por visitantes que usualmente têm que apelar para a boa vontade dos funcionários, pois acham muito complicado ou até sem nenhum sentido o modo como as fileiras de livros são identificadas.

Em ambos os casos os alunos ainda eram capazes de encontrar seus livros ou teses dado que a maior delas possuía não mais que 20.000 exemplares. Contudo, numa biblioteca como a da Faculdade de Filosofia, Letras e Ciências Humanas que hoje possui um inventário de mais de 500 mil livros e ainda 600 mil teses¹, a busca se torna completamente inviável sem nenhum auxílio real.

2 Parte Objetiva

Nessa parte do texto, serão apresentados os conceitos teóricos do trabalho, além da parte prática de desenvolvimento.

2.1 Objetivo

Com uma ferramenta que mostre como ir do ponto onde está o visitante até o livro através de um mapa, esse dificilmente se sentiria perdido ou precisaria contar com o auxílio de terceiros. Portanto, o objetivo desse projeto Library Mapper é que o visitante busque, escolha o livro e receba no seu aparelho mobile um mapa com o caminho correto até esse livro.

2.2 Problema

Como dispositivos GPS não mapeam dentro de residências, é inviável utilizar a Geolocalização do visitante para qualquer fim e portanto era necessário criar algum método para identificar onde ele estava localizado, como era esse lugar e como gerar uma busca entre onde ele estava e onde devia chegar.

2.3 Solução

Inicialmente desenvolvida para dispositivos IOS, essa ferramenta utiliza tags com códigos QR para auxiliar o reconhecimento da localização atual do visitante.

O visitante ao entrar numa biblioteca procura essa tag e com seu dispositivo IOS, reconhece o código nela contido. Esse código abre um Web Browser no dispositivo e uma página da biblioteca na internet é mostrada na tela. O visitante digita o livro num campo de busca e uma vez que esse existe nessa biblioteca, a tela do web browser é substituída por uma tela com um mapa que mostra o caminho da tag selecionada até a fileira onde está o livro.

Para ser possível a localização do livro, será utilizada a prévia catalogação feita pela biblioteca da posição do mesmo e uma réplica proporcional do mapa real da biblioteca. Essa réplica é desenvolvida através de uma plataforma web que gera mapas 2D também desenvolvida nesse projeto. As informações obtidas nesse mapa, localização de estantes e suas correspondentes identificações, Qr-Codes, obstáculos e limites da biblioteca, serão enviados e armazenados num banco de dados da aplicação.

¹Dados levantados em janeiro 2012 com a diretora da biblioteca, Maria Aparecida Laet

O mapa será gerado num grafo tipo grid e cada nó está dividido em 4 tipos principais que são QR-Code, Forbidden, Free e Bookshelf numa classe Node e será baseado nesses tipos que a busca pelo livro dado será feita.

Como todas as arestas deste grafo tem peso único um, o algoritmo de busca usado é uma Bi-direcional BFS (Breadth-first search) com uma busca saindo da origem (QR-Code selecionado pelo visitante) e outra do destino (estante onde está o livro).

Essa busca é feita de uma maneira paralela onde cada thread é um dos pólos do caminho Bi-direcional.

2.4 Conceitos e tecnologias estudadas

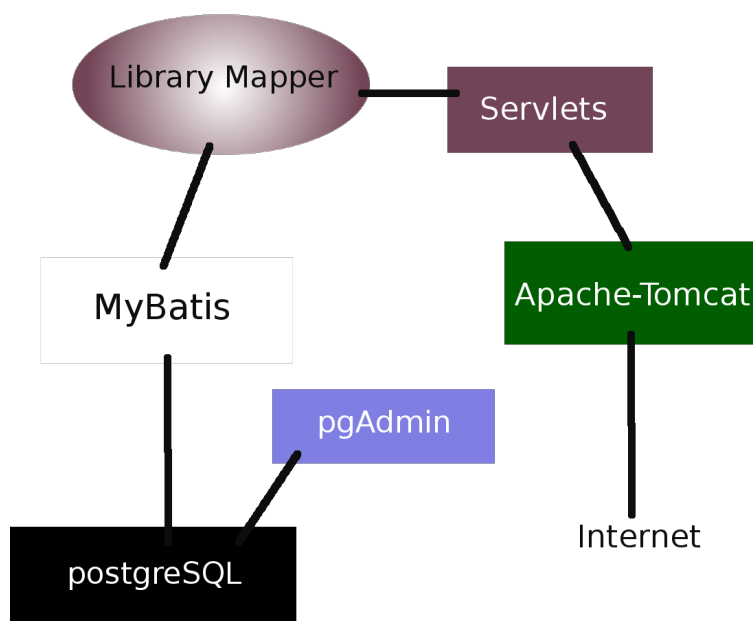


Figura 1: Tecnologias usadas

2.4.1 O que são Qr-Codes?

Quick Response Code foi inventado por DENSO WAVE, uma empresa Japonesa de captura automática de dados, FA e campos de solução de negócios, em 1994 e foi um dos primeiros códigos de barra 2D a ser usado em aplicações onde era necessário uma combinação de camera de aparelhos celulares e códigos de barra 2D. [5]Esses códigos trocam barras e espaços por pontos e espaços arranjados num vetor ou uma matrix, aumentando consideravelmente a quantidade de dados armazenados num espaço igualmente usado para um código de barra 1D.

Apenas em 2002 cameras de celulares foram comercialmente vendidas com dispositivos reconhecedores de Qr-codes e desde então uma vasta variedade de aplicativos começou a surgir no Japão e vem tomando o mundo.



Em 2006, como parte do Windows Live service, a Microsoft lançou o Windows Live Barcode , onde Qr-codes eram usados como ponte para troca de informações entre aparelhos mobile e outras medias como PCs, Tv de plasma e revistas.

As Qr-Codes geradas para esse projeto passarão para o usuário o endereço de uma página web de busca dos livros da biblioteca em que ele se encontra. Junto nessa URL tem um id do Qr-Code, que mostra para o programa onde no grafo do mapa da biblioteca se encontra o visitante, para dessa maneira poder criar o caminho que o mesmo deve traçar.

2.4.2 Grid Graphs

Um grafo G é uma par (V, E) , onde V é um conjunto finito e E é uma relação binária de V . O conjunto V é chamado de o *conjunto de vértices* de G e seus elementos são chamados de *vértices*. O conjunto E é chamado de *conjunto de arestas* de G e seus elementos são chamados de *arestas*. Existem dois tipos de grafos - grafo direcionados e grafos não-direcionados. [2]

Uma aresta num *grafo não-direcionado* é um conjunto $\{u, v\}$ onde $u, v \in V$ e $u \neq v$ que por convenção é definida como (u, v) ou (v, u) e é possível ir de u para v e de v para u .

Já num *grafo direcionado* (ou *dígrafo*) uma aresta é identificada por setas e $(u, v) \neq (v, u)$.

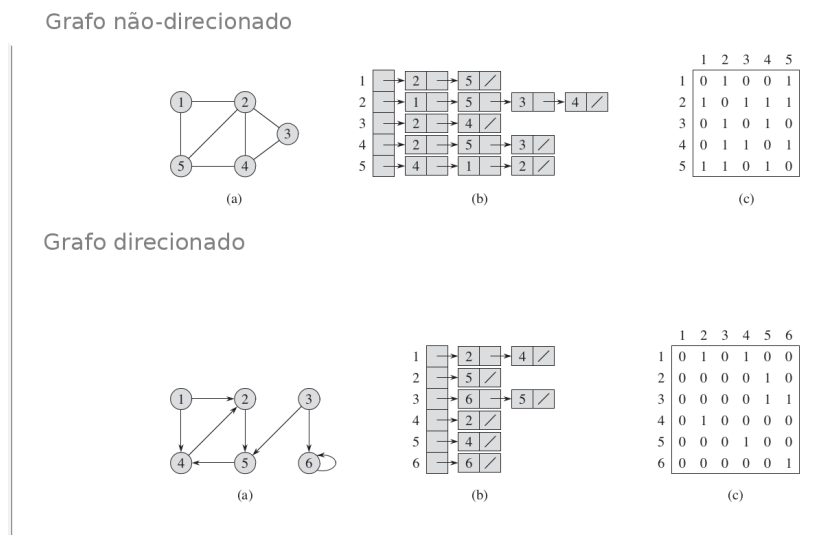


Figura 2: Grafo retirado do livro [2]

Em ambos os casos uma aresta (u,v) pode possuir um *peso* que indica o custo por passar do vértice u para o v .

Um Grid Graph (grafo do tipo Grid) é um grafo não-direcionado; uma grade de nós (vértices) $n \times n$ sendo que cada nó tem exatamente 4 arestas com exceção dos nós das extremidades.

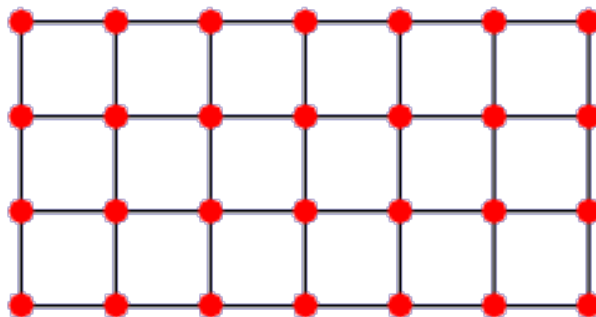


Figura 3: Representação de um Grid

Para o LibraryMapper será usado um grafo não-direcionado do tipo Grid alocado numa matriz, sendo que cada vértice do grafo será um nó da classe Node. Essa possui os seguintes atributos:

```
1 package library.domain;
2
3 public class Node {
4
5     private Integer idNode;
6     private Integer positionX;
7     private Integer positionY;
8     private Integer contentId;
9     private Integer idLibrary;
10    private String codeIdInitialShelf;
11    private String codeIdFinalShelf;
12    private String contentType; //Free, QrCode, Forbidden, Shelf
13    private Node parentFromBeginNode;
14    private Node parentFromEndNode;
15    public int isInUse;
16    private Semaphore semaphore;
17    private String color;
18    private String whoMarkedThisNode;
19    /*... continua ...*/
```

idNode: o id de cada novo Node.

positionX: coluna j na matriz $\text{Mapa}(i,j)$ ¹.

positionY: linha i na matriz $\text{Mapa}(i,j)$.

contentType: Tipo do Node, podendo variar entre Free, Forbidden, QrCode ou Shelf.

contentId: Uma vez definido qual o tipo do Node é necessário saber qual o seu id na tabela relacionada. Caso

¹A matriz que se refere ao mapa da biblioteca

não possua uma tabela, como é o caso de Free e Forbidden, o ContentsId é um valor desnecessário
idLibrary: o id da biblioteca a qual pertence.
codeIdInitialShelf: o identificador inicial de uma estante
codeIdFinalShelf: o identificador final de uma estante
parentFromBeginNode: o nó pai do nó atual na busca feita partindo do ponto inicial definido
parentFromEndNode: o nó pai do nó atual na busca feita partindo do ponto final definido
isInUse: flag que define se o nó em questão está sendo usado por outro processo de busca.
semaphore: semáforo do nó.
color: cor do nó definida no esquema da BFS.
whoMarkedThisNode: valor que define qual busca já passou por esse nó.

Alguns atributos serão melhores descritos nas próximas seções.

2.4.3 Breadth-First Search (BFS)

Dado um grafo $G=(V,E)$ com um vértice s distinto, Breadth-first search explora sistematicamente as arestas de G para descobrir todo vértice que pode ser alcançado por s . Ele computa a distância (menor número de arestas) do vértice s até todos os vértices que ele possa alcançar. Para manter o controle do progresso da busca, BFS procura cores em cada vértice - branco, cinza e preto. Todos os vértices começam com cor branca, eventualmente passam para cinza e por fim ficam pretos. [2]

Uma vez que um vértice é descoberto ele se torna não-branco. Um vértice só se tornará preto quando todos os vértices que são adjacentes a ele tenham sido descobertos, enquanto isso ele será cinza. Isso representa a fronteira entre vértices descobertos e não-descobertos como mostrado na Figura 4

Como no pior caso, a busca tem que ser feita em todos os (m) nós de um grafo e passar por todas suas (a) arestas. Como a BFS não volta nos caminhos que já traçou, a complexidade desse algoritmo é $O(a)$.

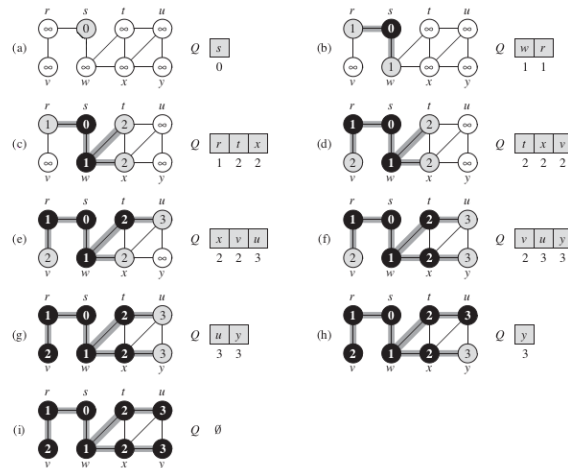


Figura 4: BFS retirada do livro [2]

Além das cores, nesse projeto essa BFS distingue entre nós (vértices) do tipo Free, Bookshelf, Qr-Code e Forbidden e será aplicada de uma maneira Bi-direcional.

2.4.4 Bi-directional BFS

A idéia por trás de uma busca Bi-direcional é fazer duas buscas rodarem simultaneamente - Uma "para frente", saindo do estado inicial (a origem) da busca inteira e outra "para trás", saindo do estado final (a chegada) da busca inteira esperando que ambas se encontrem no meio. [13].

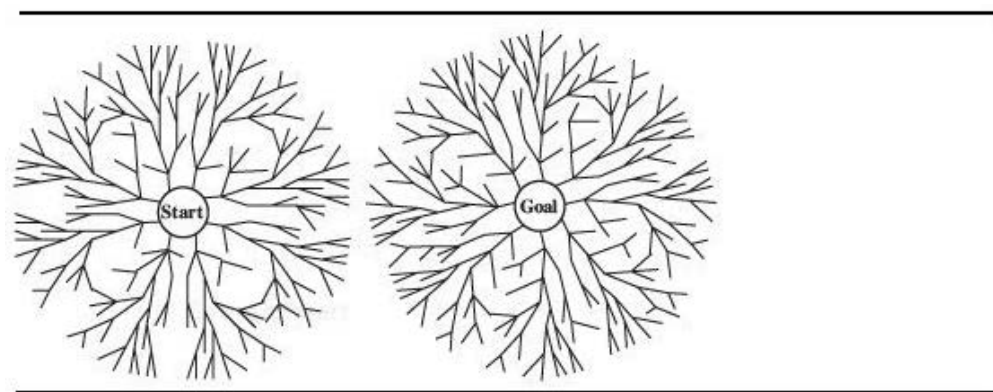


Figura 5: BFS bi-direcional retirada do livro [13]

Nesse projeto a busca Bi-direcional será realizada através de métodos concorrentes de programação. Cada pólo iniciará sua busca paralelamente liberando assim duas threads. Caso uma thread chegue em um nó que já foi visitado ela define aquele nó como o "meio do caminho" e a partir dele gera a rota inteira do mapa. Mas caso o nó que chegue esteja sendo visitado, ela para e abandona o que estiver fazendo para depois ser reciclada num outro novo processo. Nas próximas seções serão demonstradas com maiores detalhes, como essa busca foi implementada no Library Mapper.

Pela profundidade da busca ter caído pela metade, ou seja, no máximo uma busca tem que ir até a metade do grafo, a complexidade do algoritmo é $O(a/2)$.

2.4.5 pgAdmin - Interface gráfica para gerenciamento do banco de dados PostgreSQL

pgAdmin é o recurso de administração Open-Source para o banco de dados PostgreSQL. O aplicativo pode ser usado no Linux, FreeBSD, Solaris, Mac OSX e Windows para gerenciar o PostgreSQL 7.3 e acima executado em qualquer plataforma, bem como versões comerciais e derivados do PostgreSQL como Postgres Plus avançado Server e banco de dados Greenplum.

*pgAdmin é projetado para atender as necessidades de todos os usuários, desde escrever consultas SQL simples para o desenvolvimento de bancos de dados complexos. A interface gráfica suporta todos os recursos do PostgreSQL e facilita a administração. O aplicativo também inclui um destaque de sintaxe SQL editor, um editor de código do lado do servidor, um agente de agendamento de tarefas SQL/batch/shell, etc. Conexão com o servidor pode ser feita usando TCP / IP ou soquetes de domínio Unix (nas plataformas * nix), e pode ser criptografado SSL para a segurança. Drivers adicionais não são necessários para se comunicar com o servidor de banco de dados. [12]*

Com o pgAdmin não é possível iniciar um servidor de Banco de dados no linux. É necessário um terminal para criar e configurar um servidor de banco de dados e referenciá-lo no pgAdmin. Contudo uma vez isso feito, fica fácil criar banco de dados e também editar, deletar tabelas e colunas desse banco. Como exemplo,

a figura 6 mostra como é a tela de administração dos bancos de dados em conjunto de uma das tabelas do banco do Library Mapper.

No Library Mapper foi criado um banco de dados com o nome libraryMapperBD que utiliza a porta padrão do servidor de dados para se comunicar com o programa. Os detalhes desse banco serão vistos logo em seguida.

2.4.6 PostgreSQL

O PostgreSQL é um poderoso sistema gerenciador de banco de dados objeto-relacional de código aberto. Tem mais de 15 anos de desenvolvimento ativo e uma arquitetura que comprovadamente ganhou forte reputação de confiabilidade, integridade de dados e conformidade a padrões. Roda em todos os grandes sistemas operacionais, incluindo GNU/Linux, Unix (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), e MS Windows. É totalmente compatível com ACID, tem suporte completo a chaves estrangeiras, junções (JOINS), visões, gatilhos e procedimentos armazenados (em múltiplas linguagens). Inclui a maior parte dos tipos de dados do ISO SQL:1999, incluindo INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, e TIMESTAMP. Suporta também o armazenamento de objetos binários, incluindo figuras, sons ou vídeos. Possui interfaces nativas de programação para C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, entre outros, e uma excepcional documentação.

The screenshot displays the pgAdmin 3 interface. On the left, the 'Servers' tree shows the 'libraryMapperBD' database. The main pane shows the 'Properties' tab for the 'bookshelf' table, detailing its columns and constraints. The 'Columns' tab shows the table structure, and the 'Data' tab shows the table's contents.

Table Structure (Columns):

- id_bookshelf**: integer, NOT NULL, Primary Key
- id_library**: integer, NOT NULL, Foreign Key to library (id_library)
- prefix_code_id_initial**: character varying(500)
- prefix_code_id_final**: character varying(500)
- version_code_id_initial**: character varying(500)
- version_code_id_final**: character varying(500)
- code_id_initial**: integer, NOT NULL
- code_id_final**: integer, NOT NULL

Table Data (21 rows):

id_bookshelf	id_library	prefix_code_id_initial	prefix_code_id_final	version_code_id_initial	version_code_id_final	code_id_initial	code_id_final
1	1	ldsdv	fdg	dfsdf	dfsdf	213	123
2	1	sdfd	dd	dd	dd	12	122
3	1	dgd	df	df	df	411	14
4	2	prel	pref	ver11	ver11	1	1
5	2	prel	pref	ver11	ver11	1	1
6	2	prel	pref	ver11	ver11	1	1
7	2	prel	pref	ver11	ver11	1	1
8	2	prel	pref	ver11	ver11	1	1
9	2	prel	pref	ver12	ver12	2	2
10	2	prel	pref	ver12	ver12	2	2
11	2	prel	pref	ver12	ver12	2	2
12	2	prel	pref	ver12	ver12	2	2
13	2	prel	pref	ver12	ver12	2	2
14	2	prel	pref	ver12	ver12	2	2
15	2	prel	pref	ver12	ver12	2	2
16	3	REF. Q	REF. Q	F381i	F381i	180	180
17	3	REF. Q	REF. Q	F381i	F381i	180	180
18	3	REF. Q	REF. Q	F381i	F381i	180	180
19	3	REF. Q	REF. Q	F3	F3	180	180
20	3	REF. Q	REF. Q	F3	F3	180	180
21	3	REF. Q	REF. Q	F3	F3	180	180

Figura 6: tela de administração pgAdmin e tabela de dados

Como um banco de dados de nível corporativo, o PostgreSQL possui funcionalidades sofisticadas como o controle de concorrência multiversionado (MVCC, em inglês), recuperação em um ponto no tempo (PITR em inglês), tablespaces, replicação assíncrona, transações agrupadas (savepoints), cópias de segurança a quente (online/hot backup), um sofisticado planejador de consultas (otimizador) e registrador de transações sequencial (WAL) para tolerância a falhas. Suporta conjuntos de caracteres internacionais, codificação de caracteres multibyte, Unicode e sua ordenação por localização, sensibilidade a caixa (maiúsculas e minúsculas) e formatação. É altamente escalável, tanto na quantidade enorme de dados que pode gerenciar, quanto no número de usuários concorrentes que pode acomodar. Existem sistemas ativos com o PostgreSQL em ambiente de produção que gerenciam mais de 4TB de dados. Alguns limites do PostgreSQL estão incluídos na tabela abaixo. [1]

Tamanho Máximo do Banco de Dados	ilimitado
Tamanho máximo de uma Tabela	32 TB
Tamanho Máximo de uma Linha	1.6 TB
Tamanho Máximo de um Campo	1 GB
Máximo de Linhas por Tabela	Ilimitado
Máximo de Colunas por Tabela	250–1600 dependendo do tipo de coluna
Máximo de Índices por Tabela	Ilimitado

Tabela 1: Dados retirados do site PostgreSQL

O banco de dados usado pelo Library Mapper é criado para armazenar a matriz que se refere ao mapa da biblioteca, representada aqui por Mapa(i,j). O banco contém 4 tabelas:

- library

Library
+idLibrary: int
+size_X: int
+size_Y: int

Essa tabela foi criada para o cliente ter liberdade de criar inúmeras versões de bibliotecas e também carregá-las quando queira.

idLibrary: o id de cada nova biblioteca criada.
size_X: número de colunas na matriz Mapa(i,j).
size_Y: número de linhas na matriz Mapa(i,j).

- node

Node
+idNode: int
+positionX: int
+positionY: int
+contentsType: String
+contentsId: int
+idLibrary: int

Cada Node equivale a uma posição na matriz Mapa(i,j)

idNode: o id de cada novo Node.
 positionX: coluna j na matriz Mapa(i,j).
 positionY: linha i na matriz Mapa(i,j).
 contentsType: Tipo do Node, podendo variar entre Free, Forbidden, QRCode ou Shelf.
 contentsId: Uma vez definido qual o tipo do Node é necessário saber qual o seu id na tabela relacionada. Caso não possua uma tabela, como é o caso de Free e Forbidden, o ContentsId é um valor desnecessário
 idLibrary: o id da biblioteca a qual pertence.

- bookshelf

BookShelf
+idBookShelf: int
+idLibrary: int
+codeIdInitial: int
+codeIdFinal: int
+prefixCodeIdInitial: String
+prefixCodeIdFinal: String
+versionCodeIdInitial: String
+versionCodeIdFinal: String

Quando uma bookshelf é criada ela recebe dois identificadores, sendo um referente ao primeiro livro contido nela e o outro o último. Dessa maneira, é possível identificar o range de livros contido nela. Mas para ser possível fazer uma busca no banco de dados nesse range, é necessário retirar dois valores inteiros, sendo um o início e o outro o fim do range. Por isso a tabela bookshelf divide os identificadores em 3 partes: prefixo, codeId e versão. Dessa maneira, com a identificação do prefixo mais a verificação no range da estante, é possível retornar uma consulta precisa da estante que se encontra o livro. A figura 7 mostra a maneira como uma das duas identificações é dividida para ser guardada no banco de dados.

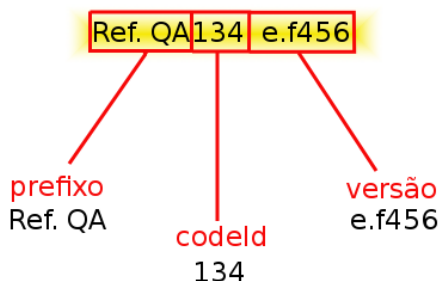
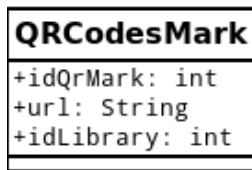


Figura 7: Identificação da estante parseada.

idBookShelf: o id de cada nova estante
 idLibrary: o id da biblioteca a qual pertence.
 prefixCodeIdInitial: prefixo inicial da estante
 codeIdInitial: range inicial da estante
 versionCodeIdInitial: versão inicial da estante
 prefixCodeIdFinal: prefixo final da estante

codeIdFinal:range final da estante
versionCodeIdFinal:versão final da estante

- qr_cod_mark



idQrMark: id de cada novo QRCode
url: a url contida no QRCode.
idLibrary: o id da biblioteca a qual esse QRCode pertence

A imagem abaixo ilustra o banco de dados completo.

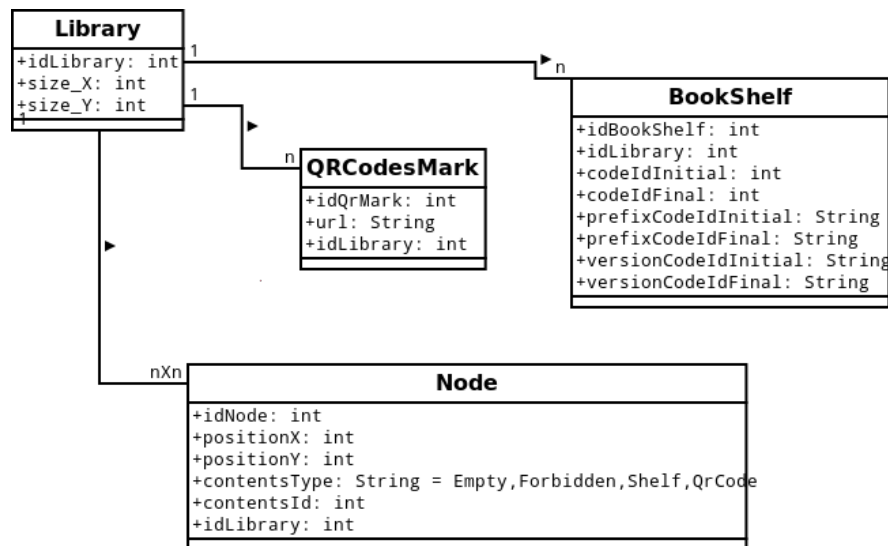


Figura 8: Identificação da estante parseada.

2.4.7 Mybatis - Mapeando SQL para JAVA

Uma vez criado o banco de dados, é necessário criar métodos que permitam operar nesse BD. Contudo, para isso ser possível é necessário fazer consultas em SQL no BD da aplicação, a qual está toda escrita em linguagem JAVA. Isso não é um grande problema, mas consome tempo e paciência do programador que terá que fazer vários ajustes e códigos extras em JDBC. (JDBC ou Java Database Connectivity é um conjunto de Classes e interfaces em JAVA, que conversam com qualquer Banco de dados através de envio de instruções SQL.)

MyBatis é um framework de persistência de primeira classe, com suporte para SQL personalizado, procedimentos armazenados e mapeamentos avançados. MyBatis elimina quase todo o código JDBC e ajuste manual dos parâmetros e recuperação de resultados. MyBatis pode usar XML simples ou Anotações para

configuração e mapa primitivas, interfaces e POJOs Mapa Java (Plain Old Java Objects) para registros de dados. [8]

No Library Mapper ainda foi usado uma ferramenta do Eclipse chamada *MyBatis Generator* que monta todo o mapeamento feito pelo Mybatis direto no projeto. Ao configurar dois arquivos, `generatorConfig.xml` e `MapperConfig.xml`, setando quais as tabelas deveriam ser mapeadas para suas futuras classes no Library Mapper, tudo é gerado automaticamente pela ferramenta.

O mapeamento consiste em três partes:

- As Classes Domínio

Essas são as classes correspondentes aos atributos do Banco de Dados. Necessariamente todas as colunas de um BD têm que estar relacionadas a um atributo dessas classes, mas o contrário não é verdade. Quando um select é feito com as interfaces Mapas descritas abaixo, um objeto de um dos domínios é retornado.

`Library.java`, `BookShelf.java`, `Node.java` e `QRCodeMark.java` são as classes domínio no Library Mapper.

- As interfaces Mapas

Essas são as interfaces JAVA com o métodos de consultas SQL e são elas que serão usadas pelo programador para fazer uma consulta no banco de dados. Essas interfaces chamam os métodos implementados nos XMLs, descritos abaixo.

`LibraryMapper.java`, `BookShelfMapper.java`, `NodeMapper.java` e `QRCodeMarkMapper.java` são as interfaces Mapas no Library Mapper.

- Os XMLs

Essas são as consultas com linguagem parcialmente SQL. Essas são as consultas mais próximas a uma linguagem SQL e são elas que são usadas nos Bancos de Dados. Todas as consultas desses arquivos têm o mesmo nome que os métodos das interfaces Mapas.

`LibraryMapper.xml`, `BookShelfMapper.xml`, `NodeMapper.xml` e `QRCodeMarkMapper.xml` são os XMLs de consulta do Library Mapper.

Uma vez gerado todo o mapeamento, basta criar uma nova seção e começar a usar os métodos da interface Mapas.

2.4.8 JSP, Servlets e Apache Tomcat

A tecnologia JavaServer Pages (JSP) permite que os desenvolvedores da Web e designers desenvolvam e mantenham de uma maneira fácil e rápida páginas Web dinâmicas. Como parte da família da tecnologia Java, a tecnologia JSP permite o rápido desenvolvimento de aplicações baseadas na Web que são independentes de plataforma. A tecnologia JSP separa a interface do usuário a partir de geração de conteúdo, permitindo aos designers para alterar o layout geral da página sem alterar o conteúdo dinâmico subjacente. JavaServer Pages é uma extensão da tecnologia Java Servlet.

Servlets são independentes de plataformas, são módulos que se encaixam perfeitamente em uma estrutura de servidor Web e podem ser usados para estender as capacidades de um servidor Web com mínima sobrecarga, manutenção e suporte. Ao contrário de outras linguagens de script, servlets não envolvem nenhuma

consideração específica da plataforma ou modificações, que são componentes de aplicativos que são baixados, a pedido, para a parte do sistema que precisa deles. [9]

Apache Tomcat é uma implementação open source do Java Servlet e JavaServer Pages, responsável por manter as páginas do Library Mapper na internet.

2.4.9 JSF2 - Managed Beans

O JavaService Facelets 2 oferece muitos recursos para o desenvolvimento de uma aplicação Web Java, sendo um dos principais os Managed Beans que são objetos utilizados para:

- *Receber os dados enviados pelos usuários através das telas Web da aplicação.*
- *Executar as lógicas para tratar as requisições dos usuários.*
- *Disponibilizar os dados que devem ser apresentados nas telas da aplicação Web.* [14]

No Library Mapper existe dois grandes Managed Beans que cuidam das interfaces Web, o MapBean.java e o SearchBean.java, sendo que o primeiro é responsável por toda a parte de criação da biblioteca e o segundo por todo o processo de busca, desde a requisição ao Colméia até a geração do Mapa com o caminho.

2.4.10 JavaScript, jQuery e JSON

JavaScript é uma linguagem de script baseada no padrão ECMAScript, aprovado como um padrão internacional ISO/IEC 16262, em abril de 1998, utilizada para programação client-side em navegadores web. Foi inventada por Brendan Eich na Netscape e primeiro apareceu no navegador Navigator 2,0 da empresa. Ele já apareceu em todos os browsers Netscape subsequentes e em todos os navegadores da Microsoft começando com Internet Explorer 3.0. [3]

jQuery é uma biblioteca JavaScript rápida e concisa que simplifica num documento HTML o tratamento da manipulação de eventos, animação e interações Ajax para desenvolvimento web rápido. [4]

JSON (JavaScript Object Notation) é um formato de troca de dados leve. É fácil para os seres humanos lerem e escreverem. É fácil para máquinas analisar e gerar. É baseado em um subconjunto da linguagem de programação JavaScript. JSON é um formato de texto que é completamente independente do idioma, mas usa convenções que são familiares aos programadores da família-C de línguas, incluindo C, C++, C#, Java, JavaScript, Perl, Python, e muitos outros. Essas propriedades fazem JSON uma linguagem de troca de dados ideal. [6]

No Library Mapper o JavaScript foi usado para criar métodos que ajudassem na lógica das interfaces e também para criar objetos semelhantes aos usados na parte JAVA, como no caso das estantes, qrCode e blocos livres e proibidos ao usuário passar.

O jQuery teve papel fundamental na parte responsável pela edição via Web do mapa da biblioteca. Métodos como *resize*, *drag* e *drop* foram utilizados para possibilitar que um desses objetos JavaScript descritos anteriormente se tornassem modeláveis através da interface Web.

O JSON foi responsável por fazer a conversão dos objetos criados na interface Web em um parâmetro compreensível para o Library Mapper.

2.4.11 HTML5 - Canvas

Hypertext Markup Language ou HTML é a linguagem de publicação da World Wide Web. [10] Atualmente na versão 5, o HTML possui novas funcionalidades e consegue realizar tarefas que antes só conseguia realizar

através de outras ferramentas, como por exemplo carregar arquivos locais.

Funcionando como um container de processamento gráfico (gráfico de jogos, arte ou imagens) provê scripts para a renderização desses gráficos. Pode gerar qualquer tipo de forma, com qualquer tipo de cor, textura, etc e por isso, no Library Mapper é usado para gerar os mapas já com os caminhos mapeados, para o visitante que procuram por um livro.

A classe responsável por esse script canvas é a `CanvasMap.java` que divide a criação do mapa em duas partes: Uma do carregamento do mapa sem o caminho traçado da busca e outra para o desenho do resultado da busca. O primeiro caso é resolvido pelo método `loadNodesFromBD(StringBuilder out)` que verifica se a matriz `Mapa(i,j)` já foi carregada pelo Library Mapper, carregando ou não do banco de dados (dependendo da resposta) e transformando cada Node dessa matriz num quadrado colorido no mapa mostrado ao usuário. O parametro `out` passado é uma `StringBuilder` do JAVA, responsável por guardar o que será escrito no arquivo do mapa desenhado `.xhtml`.

O segundo caso é resolvido pelo método `loadSearch(StringBuilder out, Book selectedBook)` que fica responsável por extrair as posições de saída (QRCode - retirado da URL que iniciou a seção) e chegada (estante do livro - retirada do livro `selectedBook`) da procura pelo livro e por chamar a busca dessas posições.

2.4.12 Desenvolvimento mobile

As pessoas apreciam aplicativos iOS que sentem como se eles fossem projetados especificamente para o dispositivo. Por exemplo, quando um aplicativo se encaixa bem na tela do dispositivo e responde aos gestos que as pessoas conhecem, que fornece grande parte da experiência as quais as pessoas estão procurando. E, embora as pessoas possam não estar cientes dos princípios de design de interface humana, como a manipulação direta ou consistência, esses usuários podem dizer quando os aplicativos seguem esses princípios e quando não. Ao começar a projetar um aplicativo iOS, certifique-se de compreender e aprender como incorporar os princípios de design HI de modo que você possa oferecer ao usuário uma experiência pessoal que ele irá apreciar. [11]

Inicialmente o Library Mapper foi desenvolvido apenas para se encaixar nos padrões IOS, especificamente para iPhones. Por isso toda sua interface e a maneira como o usuário opera o Library Mapper foi pensada para proporcionar uma experiência rápida e de fácil compreensão. Um exemplo é a página inicial de busca semelhante a usada pelo Google, que permite que o usuário reconheça facilmente que se trata de uma página de busca, pois já está familiarizado com o padrão. Outro exemplo é a idéia de um mapa ao invés de coordenadas descritivas como vire a esquerda na estante x, depois a direita, etc. Os botões estilizados para o toque e a lista de livros ocupando a tela inteira são outros exemplos também.

2.4.13 Integração Contínua

Integração Contínua é uma prática de desenvolvimento de software onde os membros de uma equipe integram seu trabalho frequentemente, geralmente cada pessoa integra pelo menos diariamente - levando a múltiplas integrações por dia. Cada integração é verificada por uma compilação automatizada (incluindo teste) para detectar erros de integração, o mais rapidamente possível. [7]

Por mais estranho que pareça falar de Integração Contínua para um time de uma única pessoa, chegou um ponto durante a criação do Library Mapper que essa técnica, ou parte dela, tornou-se essencial pois eu estava trabalhando nele em 3 lugares diferentes, com horários completamente diferentes de um dia para outro. Comecei a comitar tudo que eu fazia, indiferente se estava certo ou errado, pois assim conseguia continuar do ponto certo de onde parei. Também comecei a me policiar a tentar comitar algo todo dia, porque estava com o tempo muito restrito para desenvolver o projeto, afinal tinha que dividi-lo entre trabalho, casa recém comprada, faculdade e médico para meu futuro filho.

2.5 Library Mapper, o cliente

Essa seção é reservada para explicar toda a parte relacionada ao cliente da aplicação e, nesse caso, o cliente é o responsável pela biblioteca. O único trabalho desse cliente é montar a biblioteca pela interface Web, identificando cada estante corretamente e onde ficarão os QR-Codes. Nessa primeira versão do Library Mapper, os qr-codes criados no programa são armazenados no banco de dados da aplicação e não há como identificá-los sem ter que olhar no BD pela ordem que foram criados.

2.5.1 Construindo o seu mapa Web

Para a construção do mapa da biblioteca para o Library Mapper, existe uma interface Web que auxilia todo o processo de construção e identificação das estantes. Acessando <http://host/LibraryMapper/faces/WebLibraryMapper/mapCreation.xhtml> uma tela com um espaço definido para construção junto de todos acessórios para montagem do mapa aparecerão, como mostrado na figura 9.

Desses acessórios existe um botão Save para salvar o mapa no Banco de dados e quatro objetos para montagem da biblioteca:

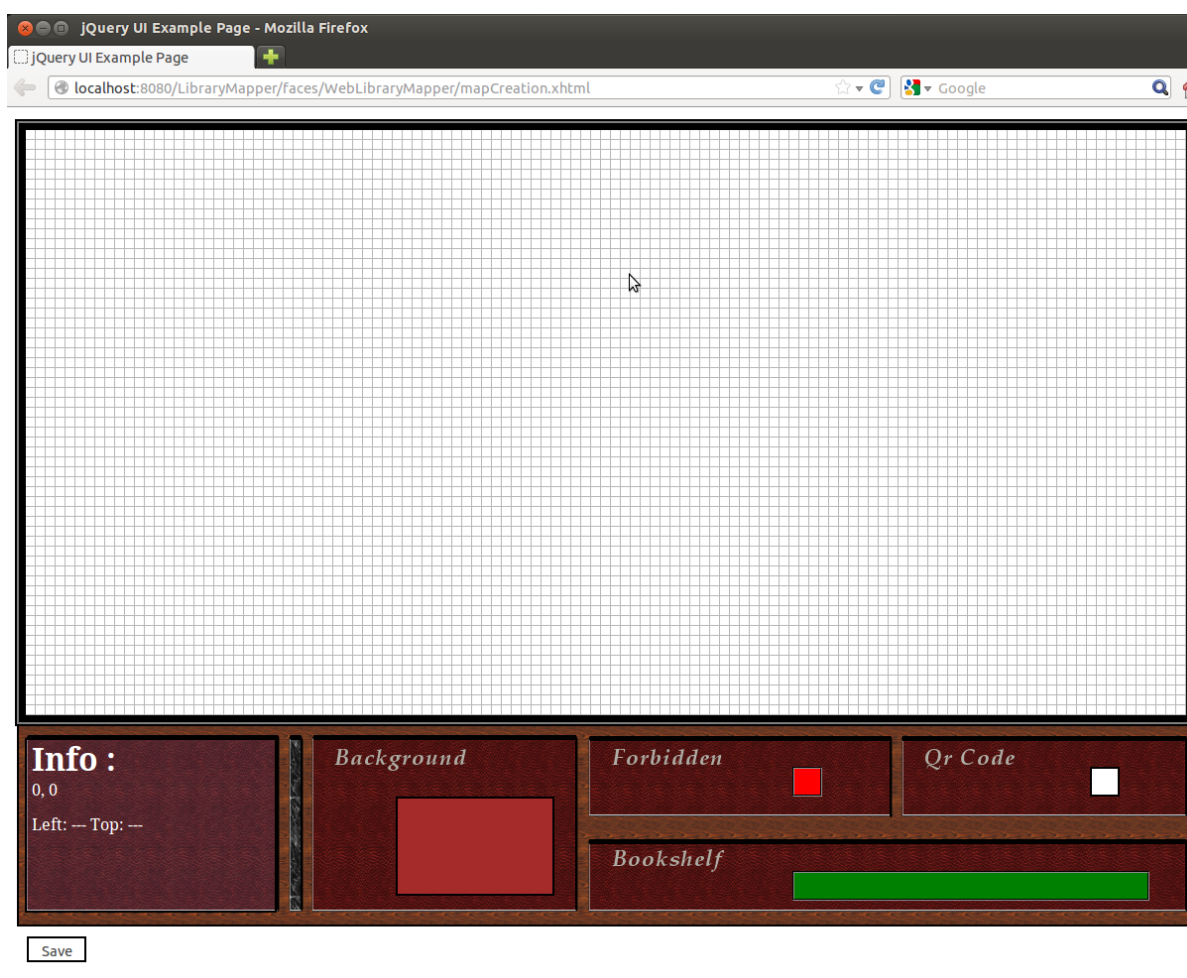


Figura 9: Interface Web para criação de mapas.

- **Background**

O objeto background deve ser utilizado para montar o chão da biblioteca, ou seja, é a base da biblioteca e tudo deve ser montado em cima dele. Uma vez o mapa transformado na matriz utilizada pelo Library Mapper, a parte do Background que não possua nada em cima será definida como um Node do tipo Free.

- **Qr Code**

O objeto Qr Code deve ser colocado na mesma posição onde será colocado na biblioteca física, pois senão o usuário encontrará problemas para se orientar no mapa. Uma vez o mapa transformado na matriz utilizada pelo Library Mapper, cada parte do QrCode gerado será um Node do tipo QrCode.

- **Forbidden**

O objeto Forbidden deve ser interpretado, com exceção das estantes, como qualquer obstáculo existente na biblioteca: vasos, colunas, esculturas, ou seja, qualquer coisa que impeça o usuário de passar por aquele espaço específico. Uma vez o mapa transformado na matriz utilizada pelo Library Mapper, cada parte do Forbidden gerado será um Node do tipo Forbidden.

- **BookShelf**

O objeto BookShelf deve ser interpretado como **uma parte da estante** física, ou a estante inteira, dependendo dos livros que ela possui. Uma estante é definida por dois identificadores, sendo um de início referente ao primeiro livro da estante e um de fim, referente ao último livro da estante. No caso da biblioteca do IME-USP a identificação é feita dessa maneira e começa no topo esquerdo da estante e termina no piso direito da mesma, como mostrado na figura 10.

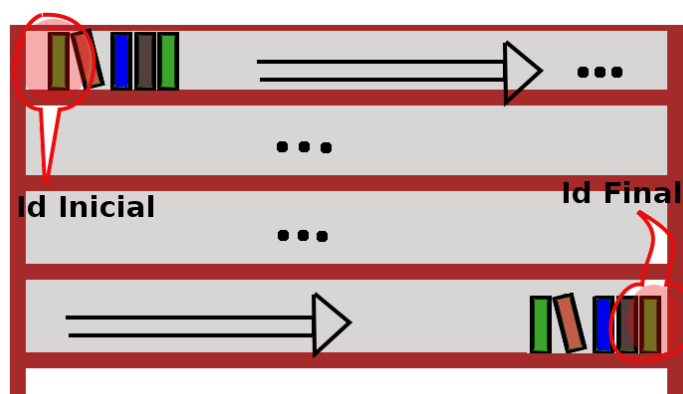


Figura 10: Referencia a organização de uma estante na biblioteca IME-USP

Caso uma estante seja um conjunto de livros do mesmo prefixo, então tem-se uma estante inteira igual a estante física como na figura 11.b, porém caso existam vários tipos de conjuntos de prefixos devem ser criadas n estantes do Library Mapper, onde n é o número de conjuntos contidos na estante física, como mostrado na figura 11.a.

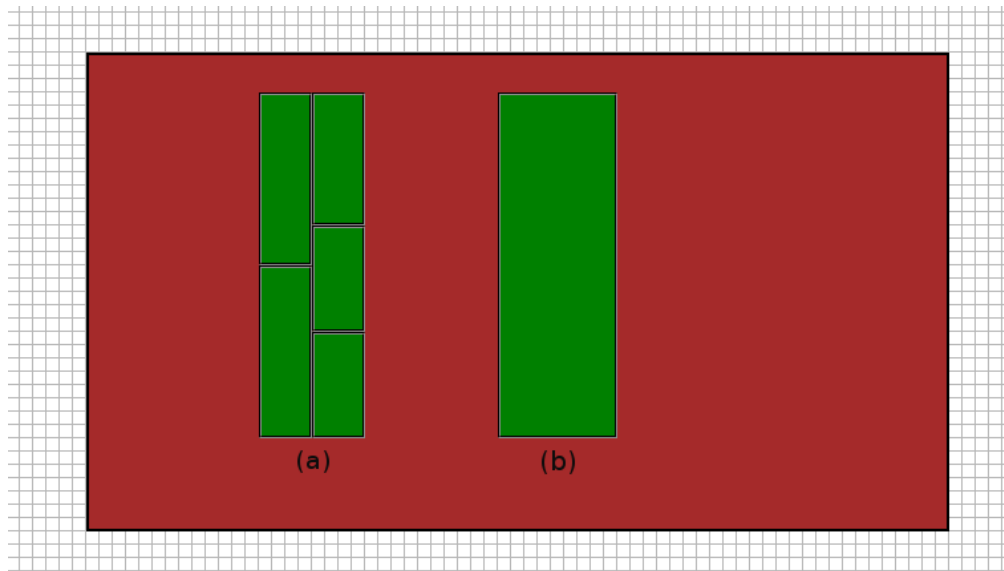


Figura 11: A figura (a) mostra uma estante física dividida em várias estantes do Library Mapper enquanto a figura (b) mostra apenas uma única estante.

Essa interpretação é importante porque dado que o mapa é definido numa superfície 2D não existe outra maneira de separar grupos diferentes de livros numa mesma estante, a não ser colocando uns do lado dos outros.

Outro aspecto desses objetos é que eles podem ter seu tamanho e sua posição mudados dentro da área reservada de desenho do mapa, apenas clicando no objeto e arrastando. Essa propriedade se dá graças ao jQuery e seus métodos **draggable** e **resizable**. Importante também é o fato de tudo isso acontecer apenas dentro dessa área reservada para o desenho do mapa, porque senão a tela ficaria completamente desordenada. Essa área é a única que permite que objetos sejam *dropados* para a criação de um mapa e isso é possível graças ao método **droppable** do jQuery também.

2.5.2 JavaScript para Nodes e Nodes para o Banco de Dados

Uma vez definido os objetos que formarão o mapa da biblioteca, o usuário clica no botão Save para salvar no banco de dados da aplicação. No meio desse processo existe um passo importante, que é a transformação da informação Web para algo compreensível para o Library Mapper poder usar.

A maneira de descrever as partes do mapa como objetos foi proposital, dado que cada uma dessas partes é um objeto do JavaScript, como mostrado no exemplo abaixo na criação de um objeto do tipo Shelf.

```

1  mapObject = new Object();
2  var tempType = $(".shelf").eq(i).attr("class").split(' ');
3  mapObject.type = tempType[0];
4  mapObject.id = $(".shelf").eq(i).attr("id");
5  mapObject.top = $(".shelf").eq(i).position().top;
6  mapObject.left = $(".shelf").eq(i).position().left;
7  mapObject.height = $(".shelf").eq(i).height();
8  mapObject.width = $(".shelf").eq(i).width();
9  mapObject.shelfId = IdEstantes[shelf];

```

Nesse caso em particular, existe um atributo a mais que é o shelfId, pois o tipo estante é o único com identificador físico. Cada objeto obtém suas informações das classes definidas nos botões HTML (bloco, forbidden, qrCode e shelf), que ao serem arrastados geram um clone com suas propriedades as quais serão atribuídas a um objeto no momento que o botão Save for clicado.

Esse objeto será adicionado numa pilha junto com todos os que foram criados e, uma vez terminado o empilhamento, a biblioteca JSON se encarrega de transformar toda a pilha numa grande String, a qual será passada para o Managed Bean MapBean através de um hidden input do HTML.

Uma vez que essa String é pega pelo MapBean, ele parseia cada item com o auxílio do JSON e transforma em objetos JsonMap, montando um array com cada objeto criado. Após criar esse array de objetos JsonMap ele os divide de acordo com seus tipos em diferentes listas de Nodes, ou seja, o tipo bloco vai para uma lista de Node com contentType Free, shelf para uma lista de Node com contentType Shelf, qrCode uma lista de Node com contentType QrCode e forbidden uma lista de Node com contentType Forbidden. Essa divisão é necessária devida a particularidade de alguns tipos.

Durante o processo de transformação toda a área reservada da interface web é ajustada para o padrão do Library Mapper e é montada uma matriz de Nodes. Um exemplo é número de posições que serão ocupadas por uma estante na matriz de Nodes, que é baseado no seu comprimento e largura na interface Web. Essa matriz é mandada para o banco de dados que grava cada nó numa tabela Node, verificando o tipo de cada um para poder também gravar nas tabelas BookShelf e QrCodeMark os respectivos tipos Shelf e QrCode desse Node.

2.6 Library Mapper, o usuário

Essa seção é reservada para explicar toda a parte relacionada ao usuário da aplicação, que para essa versão do Library Mapper são os visitantes de uma biblioteca.

2.6.1 Identificando o usuário

O primeiro contato que o visitante tem com o Library Mapper é através de um QrCode com o endereço da aplicação e, para acessá-lo ele precisa estar perto desse Qr-Code para poder identificá-lo. Portanto, o uso da posição desse Qr-Code parece bastante razoável como ponto de partida para a busca do livro que o usuário procura.

A URL contida no Qr-Code possui o endereço para a aplicação com o id desse Qr-Code. A página extrai esse id e manda para o Managed Bean SearchBean.java junto da consulta que o usuário digitar. Esse id é a coordenada XY do QR-Code que o SearchBean guarda durante toda essa seção de busca. Quando esse id é gerado, ele multiplica a posição X do objeto JavaScript por 10 elevado ao número de casas máximo de Y e soma o valor de Y ao resultado. Logo, para obter o inverso é só dividir pela mesma potência de 10, atribuindo esse resultado para X e o resto para Y. No exemplo da figura 12, o qual refere-se ao Qr-Code da biblioteca do IME, o número de casas máximo foi 2 pois o maior valor da posição Y é 57.

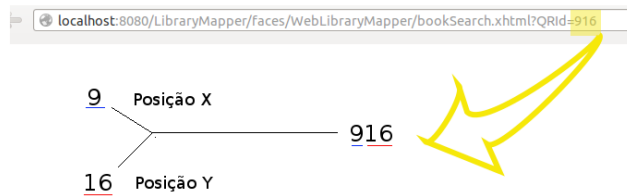


Figura 12: Exemplo de id do Qr-Code passado pela URL

2.6.2 A consulta e a ligação com a biblioteca

Como descrito na seção anterior, o Qr-Code selecionado pelo visitante possui um id com a posição dele no mapa do Library Mapper e uma URL para uma página de busca de livros da biblioteca. Nessa página o visitante deve escrever um nome de um livro, ou o autor, ou editora, algo que torne possível reconhecer o livro desejado, para que uma consulta possa ser realizada no banco de dados dessa biblioteca.

A consulta é enviada para o SearchBean que chama a classe ServiceWeb.java para montar e enviar a requisição ao serviço da biblioteca.

Como essa busca será realizada no banco de dados da biblioteca, o Library Mapper se restringe a apenas enviar uma consulta para um serviço dessa biblioteca e esperar por uma resposta desse serviço, pois dessa maneira nenhuma quebra de protocolo de segurança ou ameaça de alteração de dados poderá ser feita. No caso da biblioteca do IME o serviço chamado pertence ao Colméia, onde a consulta é enviada através de uma requisição POST e é retornado um XML contendo uma lista de possíveis livros, como mostrado na figura 13.

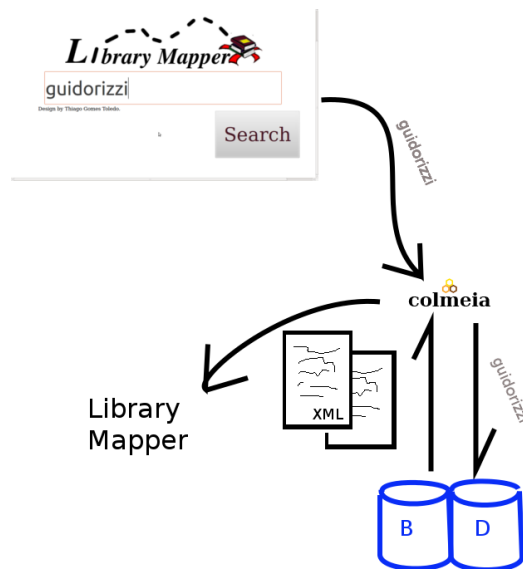


Figura 13: Conversa entre Colméia e Library Mapper

2.6.3 Lista de livros

A lista retornada pelo serviço da biblioteca deve ser *tratada* para que o Library Mapper possa mostrar uma lista compreensível para o usuário. Como o Colméia retorna um arquivo XML, esse arquivo é parseado com o objetivo de retirar cada livro dessa lista.

Ao receber a lista como resposta ao POST realizado pelo ServiceWeb, o SearchBean envia essa lista para a classe ParseColmeiaXML.java poder transformar todo o conteúdo parseado do XML em uma lista de Objetos Books. Caso outra biblioteca não retorne como resposta um XML, basta criar uma classe no mesmo esquema da ParseColmeiaXML.java, sendo que esta compreenda a resposta enviada pelo serviço da biblioteca e fazer o SearchBean chamá-la.

Com os objetos books colocados numa lista, o SearchBean guarda essa lista para ser usada na próxima página que aparecerá para o usuário, a bookSelection.xhtml. Nessa página aparecerá essa lista de Books com apenas os títulos de cada livro e um botão Back, para realizar uma nova busca, como mostrado na figura.14 a.



Figura 14: Exemplo da lista de livros e a página de informações mostradas para o visitante

Quando um livro é selecionado, uma nova página com informações sobre o livro aparecerá o qual será guardado pelo SearchBean (figura.14 b), para caso o usuário realmente queira traçar a rota até esse livro.

2.6.4 Montando o mapa com HTML5 e canvas

Quando o usuário escolhe traçar o mapa até o livro, o SearchBean cria uma nova instância da classe CanvasMap.java que irá gerar o mapa com a rota até o livro em duas etapas:

- Desenhando todo o Mapa sem o resultado da busca
- Desenhando o resultado da busca por cima do mapa

Cada Node da matriz da biblioteca é interpretada pelo CanvasMap como um quadrado que varia de cor de acordo com seu ContentType. Essa interpretação é feita pelo método *createJavaScriptForImpression*, que cria blocos de códigos script CANVAS. O CanvasMap irá gerar muitos blocos de linhas de código script CANVAS numa StringBuilder que será escrita no arquivo .xhtml referente ao mapa final do usuário, pois cada um dos Nodes será um bloco a ser transformado. Abaixo segue um exemplo do bloco de um Node com o contentType Forbidden.

```

1   ctx.beginPath();
2   ctx.fillStyle = "black";
3   ctx.fillRect(0,0,10,10);
4   ctx.closePath();

```

No GlobalUtils é definido qual o tamanho que cada quadrado deve ter no mapa e as cores escolhidas para esse mapa foram :

- Free = white
- Forbidden e Shelf(não escolhida)= black
- QRCode = yellow
- Shelf(escolhida) = blue
- Caminho retornado pela busca = red

Como o script do CANVAS desenhado é lido de cima para baixo, foi mais conveniente sobreescrever antigos blocos criados do que verificar se a posição onde esse novo bloco iria ser desenhado já existia para ser editado e por isso, o desenho do mapa é dividido em duas partes, onde a primeira desenha todo o mapa e a segunda apenas sobreescreve os blocos existentes com os da rota de busca.

Quando a primeira busca do Library Mapper é realizada, todo o mapa é buscado do banco de dados, guardado numa matriz no GlobalUtils e só então é gerado o script CANVAS da matriz de Nodes do GlobalUtils. Todo esse processo toma muito tempo, pois carregar todos os dados do Banco de dados é uma operação demorada. Contudo a partir dessa busca, todo o mapa desenhado é baseado na matriz que se encontra no GlobalUtils, tornando a criação do mapa .xhtml muito mais rápida.

2.6.5 A busca pela estante

O CanvasMap recebe do SearchBean três parâmetros: A StringBuilder que será escrita no arquivo .xhtml, o livro selecionado e o próprio SearchBean. Após esse passo, o CanvasMap cria blocos script CANVAS para as estantes do livro escolhido e os grava na StringBuilder, para finalmente chamar a classe MonitorSearch.java responsável por cuidar da busca.

O MonitorSearch criará duas Threads SearchGrid.java que realizarão a busca bi-direcional, sendo que uma busca sairá da posição da estante do livro e outra da do QR-Code. As buscas feitas por essas duas Threads só pararão quando a flag *stopAllOtherTasks* do monitorSearch for setada.

Cada Thread SearchGrid fará uma busca em largura com algumas condições a mais. Da mesma maneira que na BFS clássica, os nós do grafo (o grafo nesse caso é a matriz de Nodes) mudam de cor conforme vão sendo visitados e são representados pelo atributo *color* da classe Node, seguindo o padrão de cores estabelecida no [13].

Como agora um nó do grafo pode ser alterado por Threads diferentes, podem ocorrer casos das duas Threads estarem vendo o mesmo nó ao mesmo tempo. Para isso, cada nó Node possui um semáforo *semaphore* e uma flag *estaEmUsoParalelo* que cuidam dessa situação, impedindo deadlocks e garantindo exclusão mútua.

Para determinar o final da busca, cada Thread tem três opções:

- A Thread A encontrou o nó destino e a Thread B não chegou a executar.
- A Thread A encontrou um nó preto que foi marcado pela Thread B

- A Thread A encontrou um nó cinza que foi marcado pela Thread B

Devido ao segundo e do terceiro caso, durante a busca foi necessário identificar quem tinha marcado cada nó e por isso atribuiu-se a cada Node a variável *whoMarkedThisNode*.

Caso uma das três situações for atingida, a flag *stopAllOtherTasks* é setada e é desenhado o mapa do ponto de encontro das duas Threads até os pontos de origem de cada Thread de busca (Qr-Code e estante). Isso só é possível porque cada Node possui dois atributos, *parentFromBeginNode* e *parentFromEndNode* que definem os parents de cada nó vindo da Thread do início ou do fim da busca. A não ser no caso do nó de encontro, nunca ambos valores estarão preenchidos e portanto é possível montar a primeira metade, depois a segunda metade e por fim juntar ambas formando o caminho completo.

Esse caminho é uma lista de Nodes que é retornada para o CanvasMap, o qual se encarrega de criar blocos script CANVAS para desenhar a rota do Qr-Code até a estante onde se encontra o livro.

2.7 Caso de Uso

O caso de uso para o Library Mapper foi a biblioteca do Instituto de Matemática e Estatística em parceria com o sistema de banco de dados Colméia. Foi realizada uma catalogação de cada estante e desenhada em proporção todos os itens incluindo estantes, colunas e qualquer obstáculo do primeiro andar dessa biblioteca.

2.8 Resultados e produtos obtidos

Com o programa finalizado todas as buscas realizadas por livros do primeiro andar, retornaram com sucesso o caminho certo. Com exceção da primeira busca realizada pelo Library Mapper, foi possível encontrar o livro buscado em menos de 2 minutos.

2.9 Conclusões

Infelizmente ele não resolve por completo o problema de uma busca por um livro numa biblioteca, pois ainda existe uma grande parcela desse problema atrelado a má organização feita pela biblioteca. Contudo como esse é um problema enfrentado por qualquer sistema de busca em ambientes internos, o Library Mapper por não apenas encontrar o livro, mas também mostrar o caminho de onde está o visitante até o item que procura é uma ferramenta poderosa e muito eficiente.

O Library Mapper cumpre com excelência o mapeamento de um ambiente interno e auxilia o visitante desse lugar a encontrar o que procura, levando esse direto ao livro que busca sem perda de tempo ou paciência, tornando a visita muito mais agradável e proveitosa.

3 Parte Subjetiva

3.1 Desafios e frustrações

A in experiência com alguma ferramentas do projeto me tomaram muito tempo durante esse projeto. Em particular:

- o jQuery com o métodos droppable, draggable e resizable que foram muito difíceis de configurar e acertar os css corretos;
- o timeout do Apache-Tomcat. Para montar uma biblioteca como a do IME, levava-se cerca de uma hora e meia para completar tudo e nesse tempo a seção expirava. Mas colocando no web.xml do projeto

```
1 <session-config>
2 <session-timeout>QUANTIDADE EM MINUTOS DE CADA SECAO </session-timeout>
3 </session-config>
```

o problema foi resolvido. Contudo, isso só pode ser setado durante a criação do Mapa.

- o synchronized do JAVA não estava funcionando para minha busca paralela e tive que montar meu próprio esquema de paralelismo.

Um problema frustrante foi a maneira como a biblioteca cataloga alguma das suas estantes. Em alguns casos existiam mais de 15 conjuntos de livros diferentes, tornando quase impossível nomear uma estante. A solução foram criar várias exceções tratando esses casos, o que aumentou a distância para um modelo mais geral de classificação de estantes numa biblioteca.

Mas o maior desafio foi desenvolver o projeto nos últimos seis meses em meio a casamento, pintura do apartamento (o qual eu praticamente fiz sozinho e foram 2 mãos e em algumas até 3!!), procura de coisas para a casa nova, trabalho de 40 horas semanais, outra matéria da faculdade, médico para esposa e arrumação da casa, pois a esposa estava grávida e ainda trabalhando. Esse problema eu ainda não sei bem como foi resolvido, mas tomaram algumas madrugadas e todos finais de semana.

3.2 Relações entre Disciplinas

- **Introdução à Computação**

Eu posso até não saber essa linguagem de programação, mas me dê o manual dela que em uma semana eu te ensino como ela funciona.

Essa foi a frase que eu ouvi no meu primeiro dia no IME pelo professor Roberto Hirata e desde então é dessa maneira que tento conduzir minha profissão, aceitando cada desafio que possa me fazer crescer.

Introdução a Computação foi a matéria mais importante que tive no IME, porque foi o primeiro passo para um mundo sem limites, foi onde aprendi uma nova linguagem chamada programação, onde eu era capaz de criar tudo que minha imaginação me permitisse por enquanto ela me permitiu o Library Mapper.

- **Princípios de Desenvolvimento de Algoritmos**

Além de aprender conceitos essenciais para desenvolver qualquer programa, como listas ligadas, vetores, matrizes, etc.. foi onde aprendi com o Professor Arnaldo Mandel que o conhecimento de verdade só chega para aqueles que estudam muito.

- **Laboratório de programação I e II**

Em ambas as matérias foi onde comecei a aprender a desenvolver um projeto grande e como dividir esse projeto em pedaços para que ele se torne passível de ser resolvido. Mas foi em Laboratório de programação onde conheci o conceito do QR-Code e foi onde trabalhei pela primeira vez com essa tecnologia.

- **Algoritmos em Grafos**
- **Análise de Algoritmos**
- **Conceitos de Programação**
- **Banco de Dados**
- **Programação Concorrente, Paralela e Distribuída**
- **XP - eXtreme Programing**
- **Física I e II**

3.3 Os próximos passos

Referências

- [1] Postgresql Brasil community page. <http://www.postgresql.org.br/sobre>.
- [2] H. Thomas Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1 edition, 2009.
- [3] ECMA International. *ECMAScript Language Specification*. ECMA International, 5.1 rev.6 edition, 2011.
- [4] jQuery official page. <http://jquery.com/>.
- [5] Hiroko Kato, Keng T. Tan, and Douglas Chai. *Barcodes for Mobile Devices*. Cambridge, 1 edition, 2010.
- [6] JSON official page. <http://www.json.org/>.
- [7] Martin Fowler official page. <http://martinfowler.com/articles/continuousIntegration.html#PracticesOfContinuousIntegration>.
- [8] Mybatis official page. <http://www.mybatis.org/core/index.html>.
- [9] Oracle official page. <http://www.oracle.com/technetwork/java/overview-138580.html>.
- [10] W3C official page. <http://www.w3.org/html/>.
- [11] IOS Human Interface Guideline page. <http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html>.
- [12] pgAdmin official page. <http://www.pgadmin.org/index.php>.
- [13] Stuart Russel and Peter Norvig. *Artificial Inteligence, A Modern Approach*. Prentice Hall, 3 edition, 2009.
- [14] K19 Treinamentos. *Desenvolvimento Web com JSF2 e JPA2*. K19 Treinamentos, 2011.