Sistemas de interoperabilidade entre hospitais e seguradoras

João Gomes,[1]|[2][A74033] Joel Morais[1]|[3][A70841] Miguel Cunha[1]|[4][a78478] Nadine Oliveira[1]|[5][A75614]

[1] Universidade do Minho, Braga, Portugal
[2] a74033@alunos.uminho.pt
[3] a70841@alunos.uminho.pt
[4] a78478@alunos.uminho.pt
[5] A75614@alunos.uminho.pt

Resumo Existem muitos padrões de interoperabilidade na saúde, mas existem poucos sistemas de informação utilizados para a comunicação entre os prestadores de serviços de saúde e empresas de seguros.
Com o crescimento do número de conceções torna-se impossível integrar individualmente todos os sistemas, não apresentando sequer viabilidade técnica ou económica. Para tal é necessário que ocorra uma evolução nos modelos de dados semânticos comuns.
No *HL7* registam-se algumas evoluções importantes, como é o caso do *HL7 FHIR* que abandona as arquiteturas SOA (*Service-oriented architecture*), passando a utilizar arquiteturas *RESTful*, oferecendo assim mais variedades de respostas tais como XML,JSON,HTML,etc. Além de oferecer simplicidade de implementação e recursos capazes de responder a todos os pedidos de uma forma mais eficiente.

Chapter 1

Introdução

Com o aumento da quantidade de equipamentos informáticos, presentes num ambiente hospitalar, que recolhem diferentes tipos de dados acerca dos pacientes, a necessidade da interoperabilidade com estas fontes de dados, bem como fontes externas ao hospital, é algo cada vez mais vital para o bom funcionamento do sistema de saúde.

Neste segundo trabalho prático, vai ser focado o processo de interoperabilidade entre o sistema de gestão de saúde com as seguradoras, visto que a troca de informação entre eles ainda é algo que carece de bastante atenção, ainda não estando totalmente otimizado.

É verdade que existem bastantes standards de interoperabilidade na saúde, mas poucos são usados nos sistemas de informação de modo a que exista uma comunicação entre os serviços de saúde e as seguradoras.

Tendo também em conta que existe um mercado crescente e cada vez mais elevado de seguradoras, principalmente no setor privado, é necessário existir uma evolução o mais rapidamente possível, passando de uma integração individual destes sistemas, para uma integração de um modelo de dados semântico comum. Para isto é necessário um sistema de conversão de todas as mensagens. O objetivo é que exista apenas uma interface dos serviços de saúde que comunique com diferentes interfaces das seguradoras.

Neste trabalho vamos proceder a implementação destes dois sistemas, bem como, efetuar a comunicação entre eles tendo por base a comunicação do *HL7 FHIR*, com respostas em *JSON*.

Chapter 2

Conceção e implementação do sistema

1 Arquiteturas do Sistema

Para a realização deste trabalho resolvemos criar duas aplicações independentes entre si e estabeleciam comunicação entre elas quando necessário.

Desta forma cada aplicação possuí a sua própria base de dados, a sua própria API de dados e a sua interface.

Estamos a usar o MySQL para guardar todos os dados referentes as nossas duas bases de dados (veremos estas duas em detalhe mais á frente) e estamos a usar o REST para comunicar entre as duas aplicações (veremos estas duas em detalhe mais á frente).

2 Base de Dados

2.1 Base de Dados Hospital

A nossa base de dados referente aos hospitais contem 8 tabelas diferentes: *factura, tratamento, medico, utente, diagnóstico, hospital, pedido e utilizador.*

A tabela *utente* guarda as informações de cada utente dos vários hospitais (nif, nome, data de nascimento e telemóvel).

A tabela *fatura* guarda os dados das faturas de cada tratamento (id, descrição e o preço).

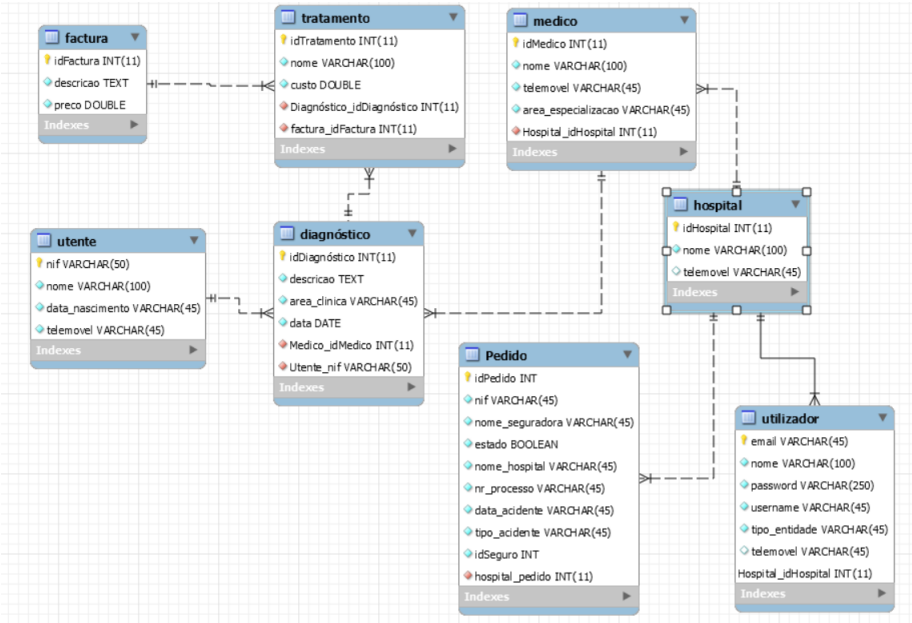
A tabela *utilizador* guarda as informações dos diversos utilizador que trabalham nos vários hospitais (email, nome, password, username, entidade e telemóvel).

A tabela *hospital* guarda os dados dos vários hospitais (id, nome e telemóvel).

A tabela *médico* guarda as informações dos diversos médicos e tem uma chave estrangeira que nos diz em qual hospital trabalha (id, nome, telemóvel e especialização).

A tabela *diagnostico* guarda os dados referentes aos vários diagnósticos, usa também uma chave estrangeira que guarda o médico que fez o diagnóstico e outra para guardar o utente a qual o diagnostico foi feito (id, descrição, área clínica e data).

A tabela *tratamento* guarda informações referentes aos diversos tratamentos, cada tratamento varia dependendo do diagnostico, logo é obrigatório termos uma chave estrangeira para ligarmos cada tratamento a um diagnostico e outra chave estrangeira para ligar o tratamento a uma fatura (id, nome e custo).



2.2 Base de dados Seguradora

A nossa base de dados que guarda todos os dados referentes as seguradoras contem 8 tabelas: *participação, relatório médico, despesa tratamento, seguro, seguro sinistrado, utilizador, seguradora e sinistrado.*

A tabela *seguradora* guarda as informações das distintas seguradoras usadas para fazer os diversos seguros (id, nome e telemóvel).

A tabela *utilizador* guarda as informações do diversos utilizadores que são trabalhadores de cada seguradora, com isso é necessário ter uma chave estrangeira para identificar de qual seguradora este utilizador trabalha (email, nome, password, username, tipo entidade e telemóvel).

A tabela *participacao* guarda as informações referente a cada participação feita á seguradora, com isto é necessário guardar chaves estrangeiras para ser possível aceder seguro em questão (numero, data do acidente, tipo do acidente, estado, nif, nome da seguradora e nome do hospital).

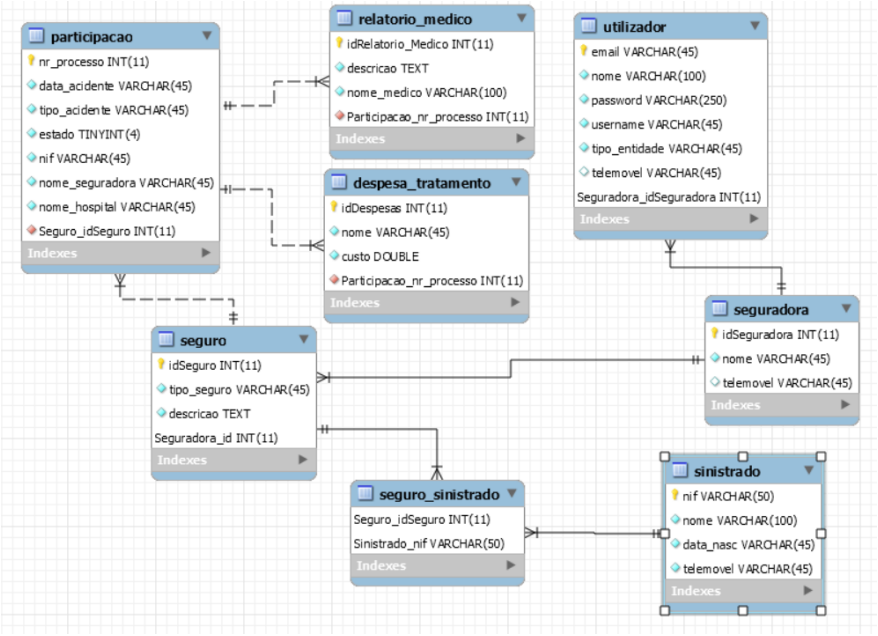
A tabela *relatório medico* guarda os dados dos diversos relatórios médicos feitos e uma chave estrangeira para conseguir ligar este relatório a uma deter-

minada participação, para mais tarde conseguir analisar, com maior detalhe, a participação (id, descrição e nome do médico).

A tabela *despesa tratamento* guarda os dados das despesas necessárias para os tratamentos derivados de cada acidente (participação), logo necessita de guardar uma chave estrangeira para ser possível associar cada despesa a uma determinada participação (id, nome e custo).

A tabela *seguro* guarda as informações dos diversos tipos de seguros presentes numa certa seguradora, logo, para alem dos dados, esta tabela também tem uma chave estrangeira para associar cada seguro á sua, respetiva, seguradora (id, tipo seguro e descrição).

A tabela *sinistrado* guarda os dados dos vários cliente que tem seguros numa segura qualquer. Logo, existe uma tabela, *seguro sinistrado*, que guarda todas as ligações entre seguros e os utilizador que os tem (nif, nome, data de nascimento e telemóvel).



Chapter 3

API REST

Neste capitulo vamos abordar de uma forma detalhada as API's REST criadas para os dois sistemas, explicando todos os models criados, controllers, as API's de dados e as interfaces das duas aplicações.

3 API REST Hospital

A API REST criada para o Hospital utiliza a base de dados do Hospital, e foi desenvolvida segundo o padrão MVC Model-View-Controller. Nesta aplicação possuímos um modulo de autenticação em que o utilizador registado na base de dados, poderá aceder efetuando um login com email e password.

Para efetuar este modulo usamos um pacote do *npm*(instalador de pacotes do *NodeJS*) chamado **Passport**. Este módulo possuí várias estratégias como por exemplo login com *facebook, google,etc.* Nós usamos a estratégia local para o primeiro acesso a aplicação e depois a estratégia *JWT*(JSON Web Token) para proteger as rotas principais e a API de dados.

3.1 Models

Os models representam a estrutura da base da base de dados utilizada. São definidos todas as tabelas, atributos e os seus tipos e as respectivas chaves. Estes models, vão ser utilizados para definir as respectivas associações entre as tabelas, com recurso à framework **sequelize**.

Figura 1. Exemplo de model criado

3.2 Controllers

Nos controllers forem definidos todos os métodos de acesso à base de dados recorrendo à framework **sequelize**. Para tal, começou-se por definir as associações entre os diferentes models, permitindo assim uma implementação de querys mais abstrata, evitando recorrer a *raw querys*.

```
/*
    Hospital 1-N Pedido
    Pedido 1-1 Hospital
*/

Hospital.hasMany(Pedido, { foreignKey: "hospital_pedido" });
Pedido.belongsTo(Hospital, { foreignKey: "hospital_pedido" });
```

Figura 2. Exemplo de associação criada

De seguida, e de modo a responder aos pedidos da API, foram definidos os seguintes métodos, para os seguintes controllers:

Controller	Método	Descrição
Hospital	getHospitais	Retorna uma lista de todos os hospitais presentes na base de dados
Hospital	getHospitaisById	Retorna um hospital com um dado id
Diagnóstico	getAllDiagnosticosByUtente	Retorna todos os diagnósticos de um determinado utente
Diagnósticos	getAllDiagnosticosByData	Retorna todos os diagnósticos de uma dada data e um dado nif de utente
Medico	getAllMedicos	Retorna uma lista com todos os medicos de um dado hospital
Medico	addMedico	Adciona um novo médico a um hospital
Medico	getMedicoById	Retorna toda a informação acerca de um dado médico
Utente	getAllUtentes	Retorna toda a informação acerca de todos os utentes de um hospital
Utente	addUtente	Adiciona novo utente a um hospital
Utente	getUtenteById	Retorna toda a informação acerca de um dado utente
Utilizador	getAllUtilizadores	Retorna toda a informação realativa a todos os utilizadores registados
Utilizador	addUtilizador	Adiciona novo utilizador
Utilizador	getUtilizador ID	Retorna toda a informação relativa a um dado utilizador
Pedido	getAllPedidos	Retorna toda a informação relativa a todos os pedidos registados
Utilizador	addPedido	Adiciona novo pedido
Utilizador	getPedidoByID	Retorna toda a informação relativa a um dado pedido
Utilizador	updatePedidoById	Atualiza o estado do pedido na base de dados

3.3 API de dados

A API de dados hospital está separada consoante os acessos que tem de fazer aos controllers especificados acima. De modo a conseguirmos obter toda a informação precisa foram criadas as seguintes rotas:

- **hospitais**:
  - *GET /api/hospitais*: Obtêm a lista de todos os hospitais presentes na base de dados.
- **diagnosticos**:
  - *GET /api/diagnosticos/tratamentos/:id*: Obtêm a lista de todos os tratamentos de um diagnostico com base no id de diagnostico.
- **medicos**:
  - *GET /api/medicos*: Obtêm a lista de todos os médicos presentes na base de dados.
- **utentes**:
  - *GET /api/utentes*: Obtêm a lista de todos os utentes presentes na base de dados.
  - *GET /api/utentes/diagnosticos/:id*: Obtêm a informação relativa aos tratamentos de um utente.
- **utilizadores**:
  - *GET /api/utilizadores*: Obtêm a lista de todos os utilizadores presentes na base de dados.
- **pedidos**:
  - *GET /api/pedidos/:nome*: Obtêm a lista de todos os pedidos de um hospital.
  - *POST /api/pedidos/:nr_processo*: Grava um novo pedido na base de dados.
  - *POST /api/pedidos/resposta/estado*: Cria resposta a enviar a seguradora e atualiza estado de um pedido de *Em Processo* para *Concluído*.

4 API Rest Seguradora

A API REST da seguradora segue os mesmos principios da API REST do hospital, visto serem similares no processo de criação, acede no entanto a uma base de dados diferente e com informações diferentes tendo impacto nos models e controllers.

O módulo de autenticação usa as mesmas duas estratégias do pacote *npm Passport*,sendo implemetado da mesma forma.

4.1 Models

Os models representam a estrutura da base da base de dados utilizada. São definidos todas as tabelas, atributos e os seus tipos e as respectivas chaves. Estes models, vão ser utilizados para definir as respectivas associações entre as tabelas, com recurso à framework **sequelize**.

4.2 Controllers

Identico ao que aconteceu para os controllers do hospital, também aqui foram definidas as devidas associações, bem como os devidos métodos, necessários para responder aos pedidos da API.

Controller	Método	Descrição
Seguradora	getSeguradoraID	Retorna a informação relativa a uma dada seguradora dado um id
Seguradora	getSeguradoraByNome	Retorna a informação relativa a uma dada seguradora dado um nome
Sinistrado	getAllSinistrados	Retorna uma lista com todos os clientes da seguradora
Seguros	getAllSeguros	Retorna uma lista com todos os seguros da seguradora
Utilizador	getAllUtilizadores	Retorna toda a informação realativa a todos os utilizadores registados
Utilizador	addUtilizador	Adiciona novo utilizador
Utilizador	getUtilizador ID	Retorna toda a informação relativa a um dado utilizador
Participacao	getAllParticipacoes	Retorna toda a informação relativa a todas as participações registados
Participacao	addParticipação	Adiciona nova participação
Participacao	validaParticipacao	Altera o estado da participacao

4.3 API de dados

A API de dados seguradora tais como as do hospital estão separadas consoante os acessos que tem de fazer aos controllers especificados acima. Para tal foram criadas as seguintes rotas:

- **seguradoras**:
  - *GET /api/seguradoras/:id*: Obtêm a informação relativa a uma seguradora com base no id.
- **sinistrados**:
  - *GET /api/sinistrados*: Obtêm a lista de todos os sinistrados presentes na base de dados.
- **seguros**:
  - *GET /api/seguros*: Obtêm a lista de todos os seguros presentes na base de dados.
- **utilizadores**:
  - *GET /api/utilizadores*: Obtêm a lista de todos os utilizadores presentes na base de dados.
- **participacoes**:
  - *GET /api/participacoes/:nome*: Obtêm a lista de todos as participações de uma seguradora.
  - *POST /api/participacoes/:nome*: Grava uma participação na base de dados
  - *POST /api/participacoes/resposta/estado*: Altera o estado da participação de *Em Processo* para *Concluído*

5 Interfaces

As duas aplicações apresentam o mesmo estilo de interface em seguida vamos apresentar algumas páginas da interface das duas aplicações.

5.1 Login:

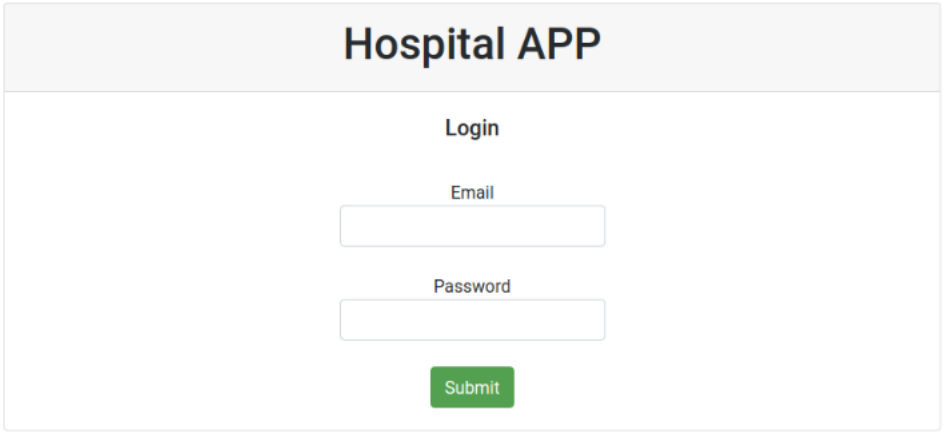


Figura 3. Login

5.2 Inicio Hospital:

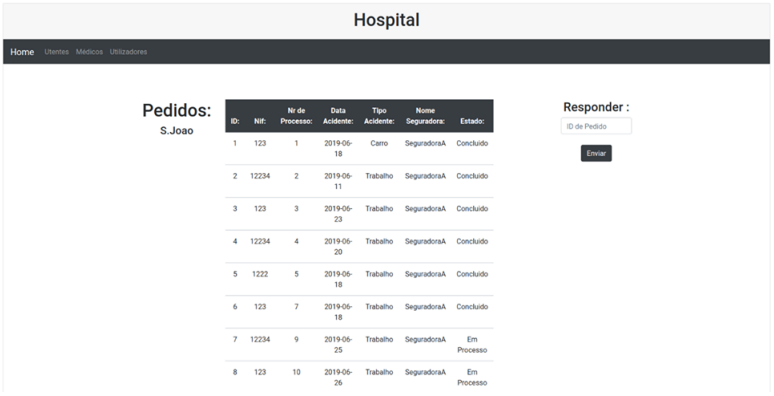


Figura 4. Hospital

5.3 Inicio Seguradora

Figura 5. Seguradora

6 Comunicação Hospital-Seguradora

Tal como foi dito anteriormente a comunicação é baseada na versão do protocolo HL7, denominada HL7FHIR.

A comunicação é feita quando uma seguradora faz uma nova participação quando isto acontece, a seguradora grava a participação e envia ao hospital que consta na participação. Posto isto pode acontecer as seguintes situações:

1. **O número do processo criado é repetido:**
   Quando isto acontece o hospital não regista o pedido e envia a seguinte mensagem:

```
{
    headers:{},
    pedido:{},
    erros: ['Pedido com número de processo recebido já existente.'],
    erro: true,
    totalRecebidos: 1,
    totalRegistados: 0,
    totalRejeitados: 1,
    totalValidados: 0
}
```

   Recebe os headers da mensagem o pedido feito pela seguradora recebe uma mensagem de erro, a flag de erro vem a true. Recebendo o número de pedidos rejeitados a 1.
2. **O número de participações do hospital no estado Em Processo excede o valor 10:**
   Quando isto acontece recebe uma mensagem semelhante a anterior mas com uma mensagem de erro diferente.

```
{
    headers:{},
    pedido:{},
    erros: ['Número de Pedidos Em processamento excedido.'],
    erro: true,
    totalRecebidos: 1,
    totalRegistados: 0,
    totalRejeitados: 1,
    totalValidados: 0
}
```

3. **Em caso de sucesso:**
   Quando tudo corre bem, recebe a mensagem:

```
{
    headers:{},
    pedido:{},
    erros: [],
    erro: false,
    totalRecebidos: 1,
    totalRegistados: 1,
    totalRejeitados: 0,
    totalValidados: 0
}
```

Nesta mensagem a flag erro encontra-se a false, as mensagens de erro não existem e informa que quantos pedidos foram registados.

Esta comunicação é efetuada novamente quando um utilizador do hospital valida o pedido, isto é, quando o pedido anteriormente no estado *Em Processo* passa ao estado *Concluído*. O hospital atualiza a sua base de dados, e manda uma mensagem a seguradora de modo a atualizar também a sua base de dados.

Neste caso podem existir duas mensagens distintas:

1. **O estado não é atualizado**:

   Quando isto acontece o hospital não consegue atualizar o estado e envia uma mensagem a seguradora que impede que o estado seja atualizado do lado deles também:

   ```
   {
       headers:{},
       pedido:{},
       erros: ['Não conseguimos efetuar Pedido!'],
       erro: true,
       totalRecebidos: 9,
       totalRegistados: 0,
       totalRejeitados: 0,
       totalValidados: 0
   }
   ```

   Recebe os mesmos parâmetros anteriores mas com uma mensagem diferente.
2. **O estado é atualizado**:

   Neste caso o estado dos dois lados são atualizados.

   ```
   {
       headers:{},
       pedido:{},
       erros: [],
   ```

```
        erro: false,
        totalRecebidos: 0,
        totalRegistados: 0,
        totalRejeitados: 0,
        totalValidados: 1
    }
```

A seguradora recebe uma mensagem com o número de pedidos validados.

Chapter 4

Recursos Computacionais

Para realizar este projeto usamos 4 recursos computacionais: *MySQL*, *NodeJS*, *PugJS* e *HL7FHIR*.

Para guardar os dados de ambas as nossas aplicações decidimos usar MySQL. Com isto foi nos possível criar 2 base de dados de forma simples e eficaz que guardassem todos os dados que são necessários de forma bastante eficiente.

Para fazer o back-end decidimos usar o NodeJS, pois o NodeJS ajuda-nos a aumentar a eficiência das nossas aplicações, pois o facto de o nodeJS não necessita de esperar que outras task acabem para começar uma nova, isto faz com que o sistema seja muito mais rápido do que se usássemos outra ferramenta. E para alem disto, NodeJS, como indica o nome, usa a linguagem JavaScript, e como já era familiarizados com a linguagem, o processo de criar o back-end tornou-se bastante mais simples.

Para o front-end vamos usar o pugJS, porque o PugJS é uma *template engine* para o NodeJS, ou seja, faz bastante sentido usar estas ferramentas juntas já que são desenhadas para auxiliar uma á outra. No PugJS utilizamos a linguagem HTML, CSS e JavaScript. Para alem disso, o PugJS apresenta uma sintaxe que torna a leitura e entendimento do código bastante simplista, mas com isto, é necessário dedicar um pouco de tempo para aprender e entender com funciona a maneira de programar no PugJS.

Para as trocas de mensagens entre as aplicações decidimos usar o HL7FHIR uma versão do HL7 que para alem de ser o protocolo abordado nas aulas, é um conjunto de normas para a transferência de dados clínicos entre sistemas de informação de saúde, e como o nosso se trabalho se assemelha muito a estes sistemas decidimos usar este protocolo para a troca de mensagens entre as aplicações.

7 Instalação

De modo a ter as duas aplicações em funcionamento é preciso criar as duas bases de dados *mysql* e povoar devidamente a base de dados, e correr **npm install** para cada aplicação, e assim terá tudo o que precisa para usar ambas as aplicações.

Chapter 5

Conclusão e Trabalho Futuro

8 Trabalho Futuro

Como Trabalho futuro teremos de melhorar as mensagens trocadas, bem como aumentar a quantidade de informação trocada, com isto, queremos dizer que em vez de só registar participações e atualizar estados, podemos acrescentar receber informação de faturação ligada a tratamentos de utentes dos dois lados.

Além disso permitir acesso da rede de hospitais a informações de todas as seguradoras na rede de hospitais permitindo aumentar o nível de interoperabilidade semântica entre estes sistemas.

9 Conclusão

Em suma, o projeto foi concluído com sucesso. Tivemos alguma dificuldades com o protocolo HL7FHIR, devido ao facto de este protocolo ser bastante complexo, mas, após um estudo aprofundado deste, conseguimos ultrapassar as dificuldades, encontradas inicialmente, e implementar o protocolo HL7FKIR com sucesso.

O resto do projeto foi feito sem dificuldade, pois, após estudar as várias opções para as ferramentas do back-end e do front-end, escolhemos as melhores, no nosso ponto de vista, e aquelas que éramos mais familiarizados, de forma a tornar esta fase do projeto simples.

Com isto, podemos concluir que o projeto foi feito com sucesso.