10 Documentação de Arquiteturas

Para projetos não triviais e empresas de médio e grande porte, a gestão do conhecimento técnico é fundamental. Um dos passos nesta direção é documentar, de forma leve e disciplinada, as decisões arquiteturais que foram discutidas nas fases iniciais de um determinado projeto. Ao longo deste capítulo, um exemplo é apresentado e os passos de sua organização e documentação são realizados.

O caso exemplo deste capítulo é inspirado em uma faculdade que possui um sistema acadêmico para os seus processos de negócio dos seus cursos de graduação. Esse sistema é baseado em tecnologia COBOL/CICS e usa banco de dados Oracle. Como consequência, o sistema é acessado pela equipe administrativa. Professores e alunos não tem acesso ao sistema e isso gera diversos problemas como sobrecarga de trabalho para a área administrativa e insatisfação para o corpo docente (professores) e corpo discente (alunos). Um projeto financiado pelo diretor de tecnologia tem como objetivo realizar uma modernização arquitetural nesse sistema, i.e., criar um portal Web de última geração para alunos e professores e permitir a migração gradativa de todo o código COBOL para uma nova plataforma ao longo de uma década. Uma reescrita completa do código antigo para uma nova plataforma seria caro e foi descartado.

Alguns desejos iniciais desse novo produto foram compilados em um pequeno texto e incluem:

- O novo sistema será implantado em módulos, com publicação semestral para manter o compasso de novas turmas.
- O sistema deve operar em vários navegadores e em telefones celulares dos alunos e professores.
- O desempenho e a escalabilidade são preocupações, devido aos períodos de pico de atividades como lançamento de notas, matrícula ou consultas de notas no fim do semestre.
- O sistema deve apresentar manutenção e testes facilitados pois a diretoria técnica que ter o menor custo de propriedade com o produto após a sua operação.
- O sistema deve se comunicar com o antigo sistema já desenvolvido com tecnologia COBOL/CICS.
- O sistema deve operar em qualquer período do dia e da noite.
- Sistemas acadêmicos mantém dados sensíveis e, portanto, a segurança é um aspecto crítico.

A partir do caso exemplo, iremos desenvolver uma documentação arquitetural que registre as decisões técnicas e o racional arquitetural.

10.1 Passo 1 - Visualização de Negócio

Recomendação Arquitetural: Comece o seu trabalho arquitetural com um desenho de alto nível, não técnico. Esse desenho é chamado de "marketecture" ou "visualização de negócio". O objetivo desse desenho é facilitar o seu discurso e ancorar a sua comunicação com gerentes, clientes, analistas de teste e desenvolvedores. Não usamos a UML ou outras linguagens técnicas nesse momento. Estamos mais preocupados aqui com comunicação visual que precisão técnica.

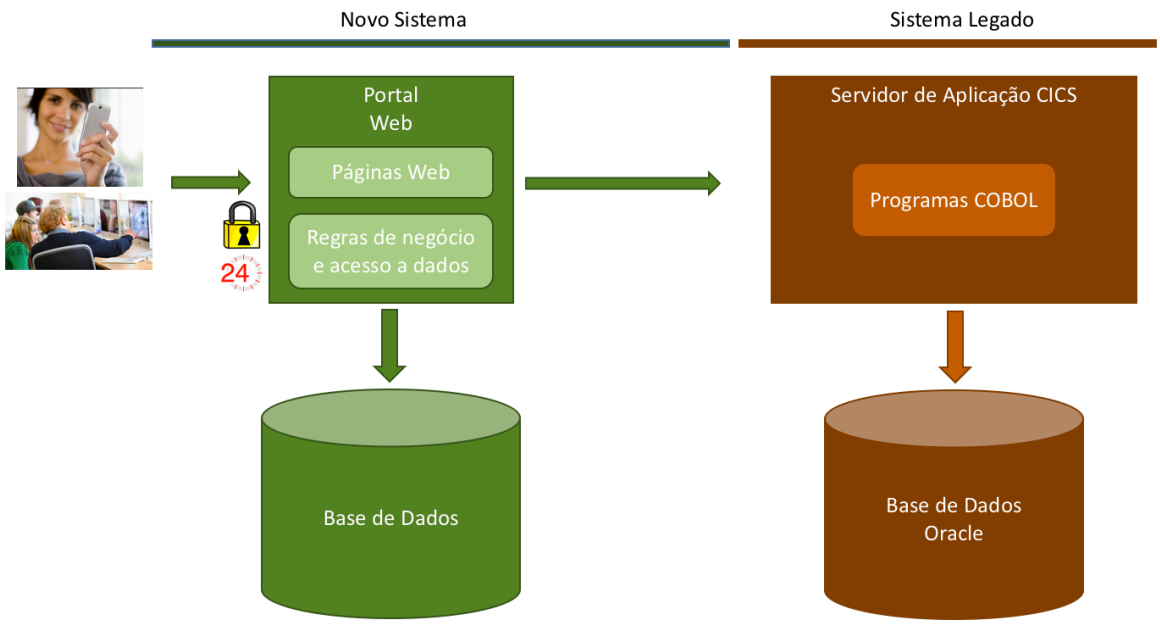

Figura 59: Visualização de Negócio do Sistema Acadêmico

Alguns pontos importantes que devem ser destacados em uma boa visualização de negócio incluem:

- Demarcar o que será construído (indicado em verde no desenho) e o que será integrado (indicando em marrom);
- Usar formas simples e humanizadas, como por exemplo os desenhos dos bancos de dados, o cadeado para indicar segurança da informação, o símbolo do relógio 24 horas para indicar alta disponibilidade e as fotos com alunos com computadores e celulares;
- Não introduzir tecnologias e decisões tecnológicas que ainda devem ser discutidas e deliberadas. Veja que o COBOL e Oracle aqui aparecem por serem informações do sistema já existente.

10.2 Passo 2 – Condutores Arquiteturais

O próximo passo é capturar e descrever os condutores arquiteturais, ou seja, qual a agenda técnica que deve guiar o trabalho do time de arquitetura. No contexto apresentado alguns condutores podem incluir:

10.2.1 Acessibilidade e Usabilidade

Racional: A acessibilidade é o atributo de qualidade que possibilita que um sistema seja universalizado e acessado pelo máximo de pessoas no maior número possível de dispositivos. Além disso, alunos entre 17 e 30 anos (gerações Y e Z) já se acostumaram a usar sistemas "bonitos" e com isso a usabilidade também se torna uma preocupação técnica.

10.2.2 Interoperabilidade

Racional: Durante toda uma década o sistema na nova tecnologia e o sistema legado irão conviver e, portanto, a interoperabilidade é um aspecto crítico. Além disso, o time de arquitetura também precisa interoperar com outros sistemas, como por exemplo:

- O sistema do Ministério da Educação que realiza o censo acadêmico;
- O sistema de segurança corporativa da faculdade baseado em Microsoft Active Directory;
- O sistema ERP que cuida do processamento da folha de pagamento e geração de boletos para os alunos.

10.2.3 Segurança

Racional: Sistemas acadêmicos são desafiados por *hackers*. Além disso, existe um conjunto de dados sigilosos como informações de milhares de alunos que irão circular em ambiente Web e dados como notas, faltas e comunicação entre professores. A agenda de autenticação, autorização, auditoria e transporte seguro ganha importância e deve ser trabalhada pelo time de arquitetura.

10.2.4 Escalabilidade

Racional: Faculdades possuem um calendário acadêmico rígido e os sistemas devem respeitar essa questão. Por exemplo, os períodos de lançamento de notas, consultas de notas e matrículas em disciplinas são críticos e geram sobrecargas imensas aos servidores. Dessa forma, a escalabilidade se torna um aspecto crítico e traz uma agenda de preocupação técnica para o time de arquitetura.

10.2.5 Manutenibilidade

Racional: O diretor da faculdade quer minimizar os custos de operação do produto. Além disso, sistemas acadêmicos tem regras de negócio bastante complicadas que requerem uma gestão bastante cuidadosa.

10.3 Passo 3 – Estilos Arquiteturais Web

No Capítulo 4 apresentamos os principais estilos arquiteturais Web (Web 1.0, Web 2.0, SPA, API Web e Micro Serviços Web). Cada estilo traz implicações distintas e é mais apropriado para um certo contexto.
No exemplo trabalhado nesse capítulo, como existe uma necessidade de modernização tecnológica e desafios intensos de acessibilidade, o estilo Web 1.0 seria já descartado. O estilo Web 2.0 é um candidato natural, sendo que até um estilo SPA poderia ser usado. O uso de API Web pode ser usado para a exposição dos serviços legados, mas devido à arquitetura monolítica do CICS o uso de micro serviços para essa API não é viável.
Em resumo, uma recomendação do uso de Web 2.0 para o portal e uso de uma API Web REST para os serviços legados COBOL pode ser uma alternativa viável para o nosso problema.

10.4 Passo 4 – Escolha da Plataforma Tecnológica

Com os condutores e estilo arquitetural Web definidos, o arquiteto e seu time devem definir a plataforma a ser usada. Uma lista de opções de plataformas Web inclui:

- Java EE Web
- ASP.NET
- Centrada em JavaScript
- LAMP (PHP, Python ou Ruby)
- Híbridas (JavaScript MVVM + Java/ASP.NET/LAMP)

Os capítulos 6, 7 e 8 apresentaram em detalhes três das plataformas mais comuns para trabalharmos com arquiteturas na Web. No mundo real, recomendamos que o arquiteto não tome uma decisão baseada apenas na sua preferência pessoal. Ao invés, ele deve pesar os seguintes aspectos:

- Habilidade do time;
- Produtividade na tecnologia pelo time;
- Restrições para a organização alvo;
- Facilidade de encontrarmos profissionais com conhecimento da tecnologia para manter o produto;
- Custo de profissionais;
- Maturidade das tecnologias;
- Risco de descontinuidade e obsolescência das tecnologias.

Recomendação Arquitetural: Sempre use critérios racionais para selecionar a sua plataforma e documente essa decisão. Como essa é a decisão mais crítica na sua escolha arquitetural, você poderá ser cobrado por ela daqui a 2, 5 ou 10 anos. Explique porque você tomou essa decisão.

No exemplo aqui trabalhado, não temos essas variáveis contextuais para fazer uma escolha e iremos apresentar as soluções técnicas nas três plataformas apresentadas: Java EE Web, ASP.NET e JavaScript.

10.5 Passo 5 – Visualização Lógica

Após capturar os condutores, o arquiteto pode começar a expressar a sua solução em desenhos técnicos. Para isso ele pode usar desenhos UML e um primeiro diagrama a ser produzido é o de pacotes. Ele fornece uma visão de pássaro da arquitetura, mas permite que os módulos centrais do produto sejam especificados. O diagrama da
Figura **60** apresenta essa visualização.
Note que esse diagrama é representado em uma linguagem formal e os módulos que irão capturar as principais preocupações arquiteturais estão aqui desenhados.

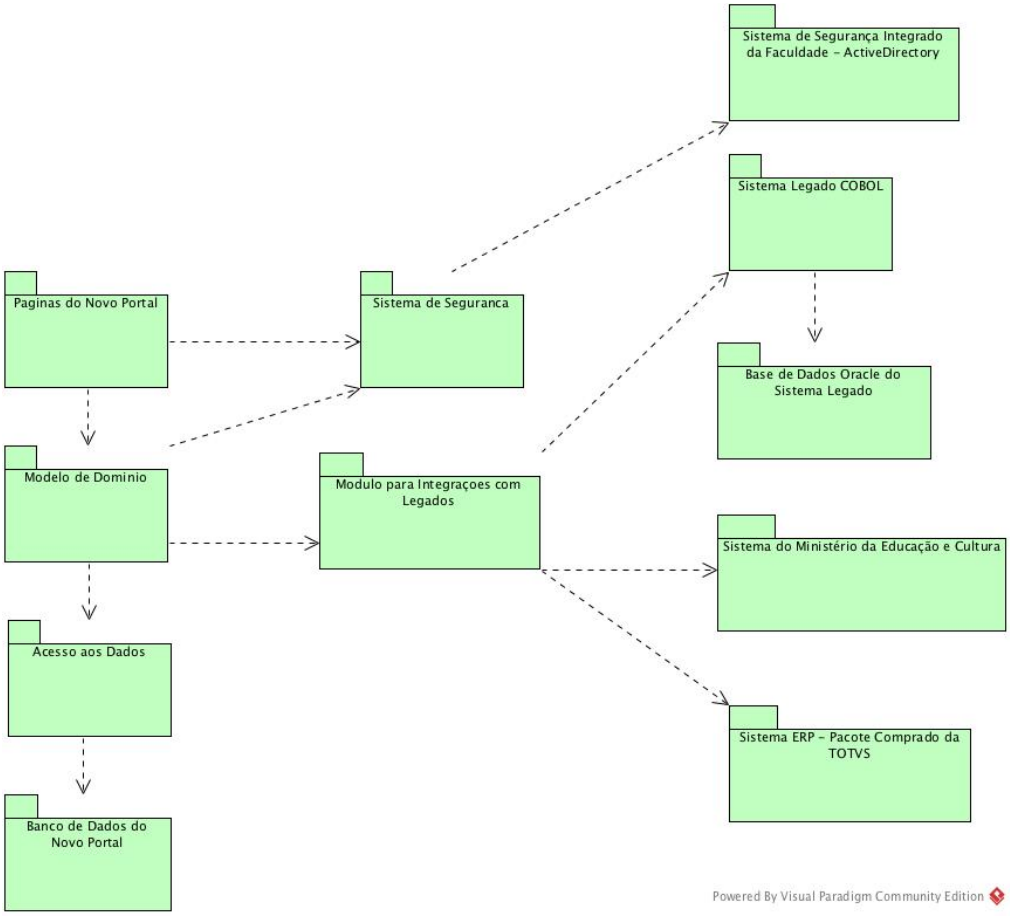

Figura 60: Visualização Lógica do Sistema Acadêmico

Algumas regras para um bom diagrama de pacotes incluem:

- Buscar a visão de amplitude;
- Mostrar as conexões entre os módulos arquiteturais;
- Não introduzir tecnologias;

Alguns arquitetos, para buscar uma melhor comunicação de negócio, também representam os módulos de negócio em um pacote especial chamado de modelo de domínio. Esse pacote pode ser refinado conforme mostrado a Figura 61. Em termos técnicos, cada pacote aqui mostrado se materializará como *package* em Java, um *namespace* em C# ou um *module* em JavaScript.

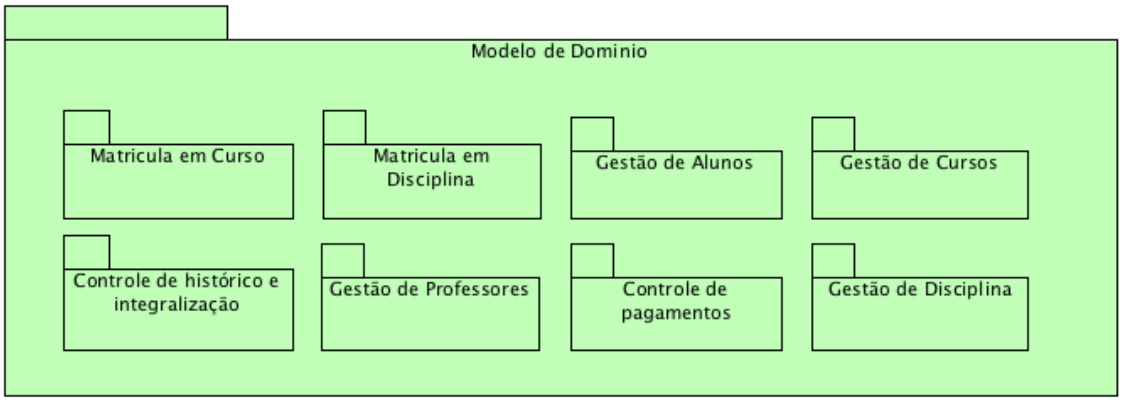

Figura 61: Modelo de Domínio do Sistema Acadêmico

10.6 Passo 6 – Visualização da Topologia Física

Tendo a visualização lógica em mãos, o arquiteto deve agora estruturar como a sua arquitetura será representada em termos físicos. A isso chamamos de topologia, ou representação da estrutura física de máquinas servidoras que irá hospedar a nossa aplicação.

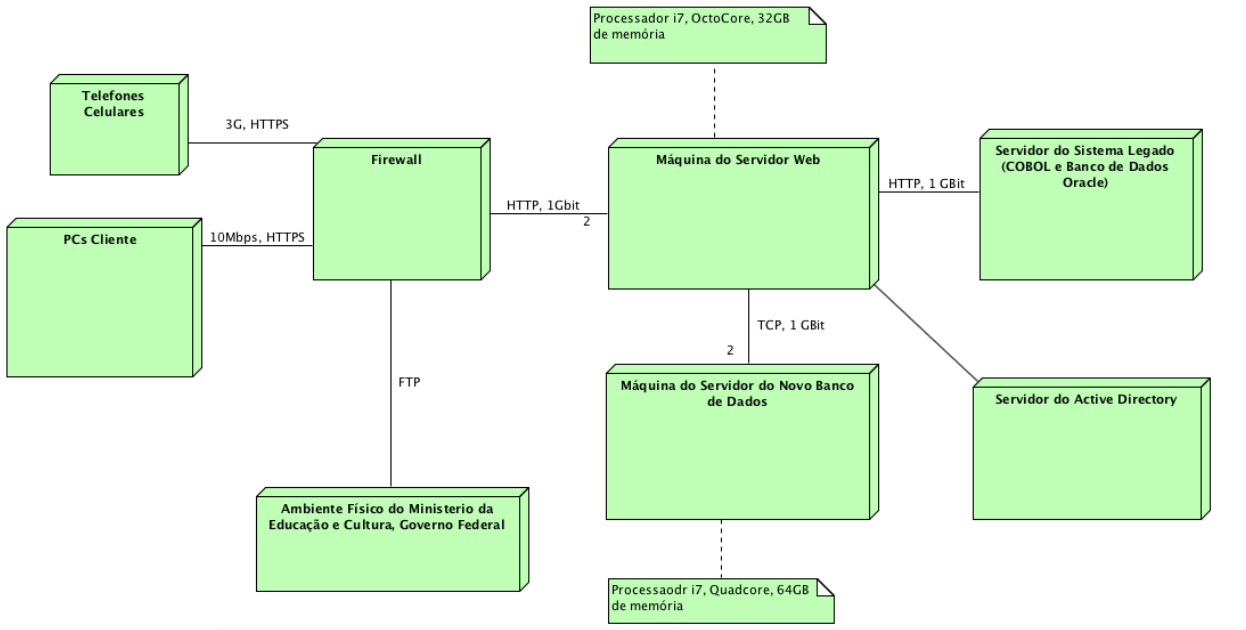

Figura 62: Topologia do Sistema Acadêmico

Neste diagrama, podemos representar:

- As máquinas que irão fazer parte da topologia, sejam elas clientes (ex. Telefones celulares) ou do ambiente da universidade (máquina do servidor Web);
- Protocolos de rede que a aplicação requer (ex. HTTPS, HTTP ou TCP);
- Bandas mínimas de passagem (ex. 3G ou 10Mbps);
- Clusters (note o número 2 na figura, indicado ao lado na máquina do servidor Web e do banco de dados);
- Máquinas externas à sua rede (ex. servidor do MEC);
- Máquinas já existentes (ex. Máquina do ActiveDirectory ou o servidor do sistema legado).

10.7 Passo 7 – Visualização da Persistência de Dados

Como a persistência é um aspecto crítico em sistemas Web, é recomendado que o arquiteto indique como

essa parte será resolvida pela arquitetura. Para isso, iremos precisar de um detalhamento técnico com o uso do diagrama de componentes UML.

Nesse passo, vamos assumir que a universidade já possui uma diretriz da arquitetura corporativa do uso de Oracle 12c e, portanto, o novo sistema de banco de dados também será Oracle. Quando esse cenário surge, isso se chama uma restrição arquitetural.

Como a persistência pode ser realizada de várias plataformas distintas, vamos fazer desenhos com o uso da plataforma Java EE, .NET e Node.JS. Os desenhos estão na Figura 63, Figura 64 e Figura 65. Cada caixa nesse diagrama representa um componente arquitetural, que em termos práticos é uma DLL, assembly, .JAR, biblioteca ou um módulo JavaScript.

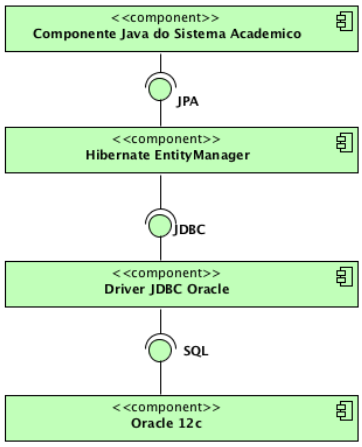

Figura 63: Visualização de Persistência em Java EE

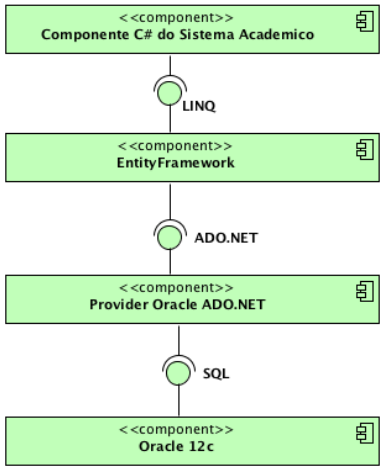

Figura 64: Visualização de Persistência em .NET

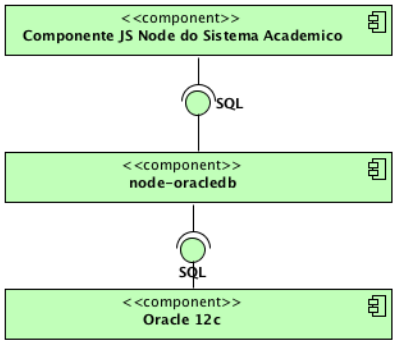

Figura 65: Visualização de Persistência em Node.JS

Uma boa visualização de persistência deve capturar o seguinte:

- Banco de dados que serão usados (ex. Oracle 12c);

- Drivers de acesso a banco de dados (ex. node-oracledb ou Oracle JDBC Driver);
- Presença ou não de frameworks de mapeamento objeto relacional (ex. Entity Framework).

Quando necessário, especifique já até a versão dos drivers e frameworks usados e sítios de acesso para download. Por exemplo, em Junho 2016 uma especificação possível da Figura 63 poderia indicar os componentes banco de dados versão Oracle 12.1.0.2.0, driver JDBC versão ojdbc7 [218] e versão do Hibernate ORM 5.2.0[219]. É sempre válido testar se essas combinações interoperam com um teste rápido ou uma prova de conceito mais elaborada se o risco é maior.

10.8 Passo 8 – Visualização da Apresentação (Usabilidade e Acessibilidade)

Agora o arquiteto deve representar que bibliotecas e frameworks para a camada de apresentação. As figuras seguintes apresentam possibilidades para Java EE, .NET e AngularJS/Node.JS.

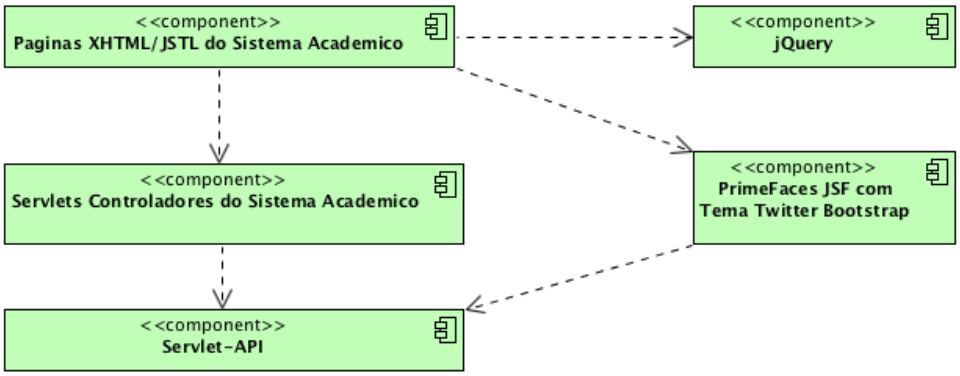

Figura 66: Visualização de Apresentação em Java EE

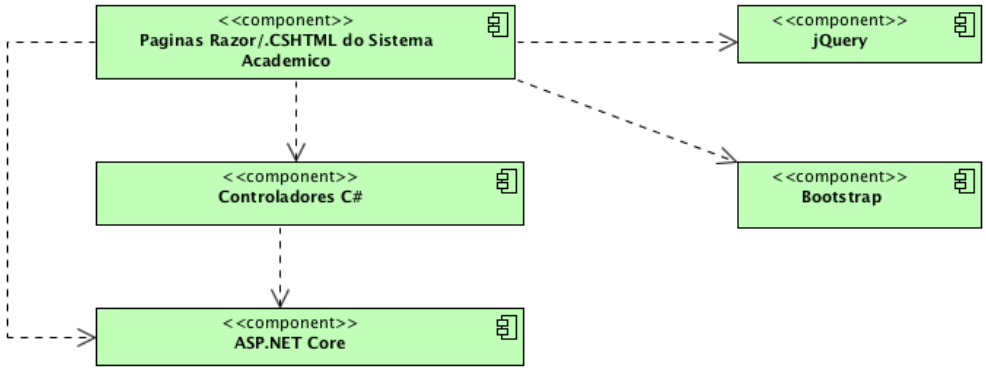

Figura 67: Visualização de Apresentação em ASP.NET Core

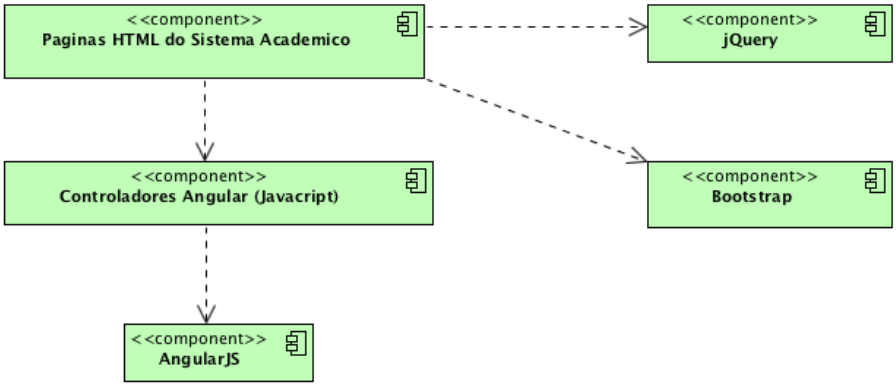

Figura 68: Visualização de Apresentação em AngularJS/Node.JS

[218] http://www.oracle.com/technetwork/database/features/jdbc/jdbc-drivers-12c-download-1958347.html
[219] http://hibernate.org/orm/downloads/

Uma boa visualização de apresentação deve indicar:

- Frameworks a serem usados e se necessário a versão de cada framework; (ex. Angular ou ASP.NET Core)
- Componentes da aplicação (ex. Páginas e controladores).

Como opção, arquitetos podem comunicar isso de forma alternativa à UML com o desenho da sua estrutura de projetos. Um exemplo nesse sentido é fornecido para um projeto ASP.NET Core com Bootstrap e jQuery.

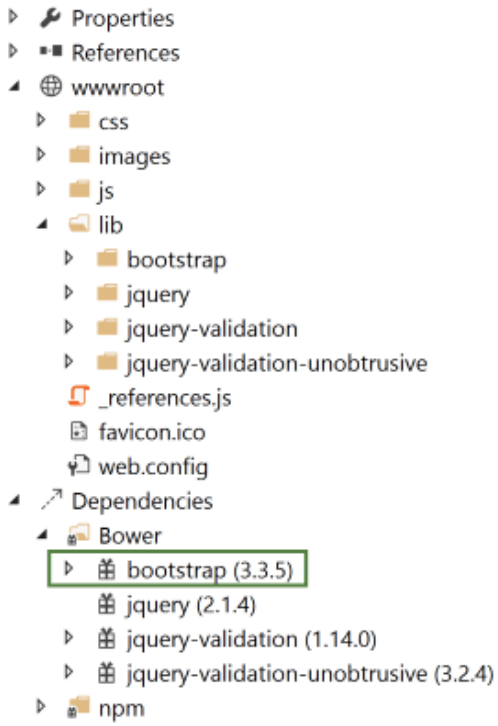

Figura 69: Visualização de Componentes de Apresentação em ASP.NET Core

Observe que a escolha de Bootstrap e jQuery foi arbitrária e usada como exemplo apenas. No mundo real, o arquiteto deve reservar tempo para buscar, comparar e selecionar as bibliotecas mais apropriadas para o seu contexto.

10.9 Passo 9 – Visualização de Segurança

O arquiteto deve também investir tempo para representar como a segurança será resolvida em sistemas Web. No exemplo sendo aqui trabalhado, em particular, a segurança foi colocada como um aspecto crítico e usaremos como premissa que a autenticação de usuários deve ser realizada sob controle de um ambiente controlado da faculdade, que é um servidor LDAP da Microsoft, o Active Directory.

Algumas soluções para Java EE, ASP.NET e Node.JS são mostradas nas figuras a seguir.

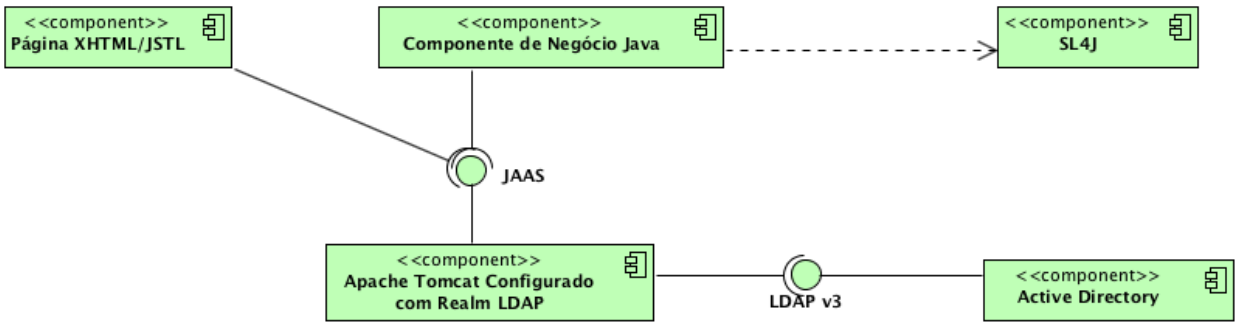

Figura 70: Visualização de Segurança em Java EE

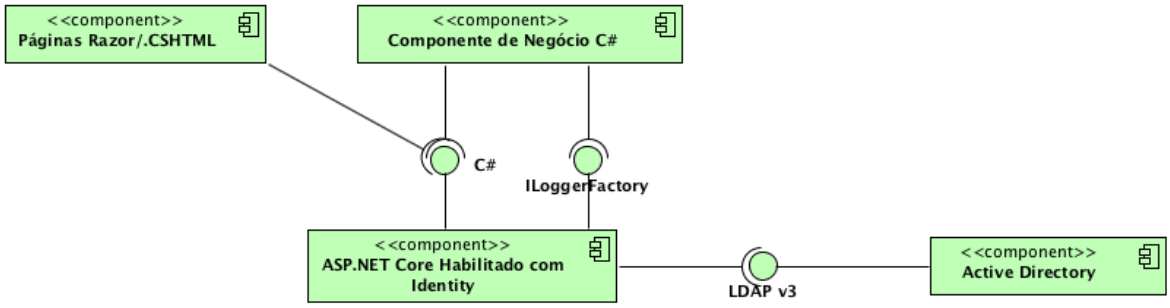

Figura 71: Visualização de Segurança em .NET

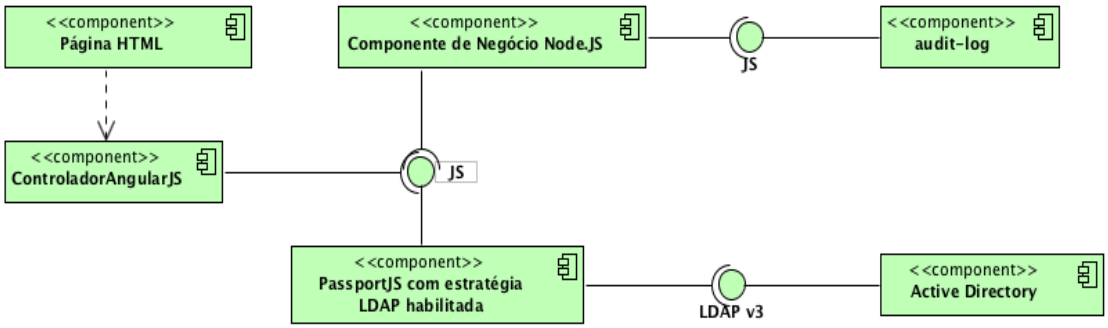

Figura 72: Visualização de Segurança em Node.JS

Alguns pontos importantes que o arquiteto deve considerar nessa visualização incluem:

- Componentes extras a serem usados (ex. No exemplo do Node.js temos os componentes passport e audit-log);
- Configurações a serem realizadas (ex. ASP.NET Core habilitado como Identity ou Apache Tomcat configurado com LDAP)
- Protocolos relevantes (ex. LDAP ou JAAS)

Como aspectos de segurança podem envolver configurações específicas, é bom que a documentação arquitetural indique sítios auxiliares que contenham exemplos de como fazer essas configurações. Exemplos:

- Configuração do LDAP com AD no ApacheTomcat[220] [221].
- Configuração e uso do Node Passport com AD[222]

[220] https://tomcat.apache.org/tomcat-7.0-doc/realm-howto.html#JNDIRealm
[221] http://stackoverflow.com/questions/267869/configuring-tomcat-to-authenticate-using-windows-active-directory
[222] https://www.npmjs.com/package/passport-ldapauth

10.10 Passo 10 – Visualização de Interoperabilidade

Em sistemas não triviais ou em grandes empresas, a integração é sempre um ponto de atenção no desenho de arquitetural. Neste aspecto, é importante primeiro conhecermos os estilos mais comuns de integração antes de discutirmos opções tecnológicas nas plataformas Java, .NET ou Node.js. A Figura 73 apresenta estes estilos.

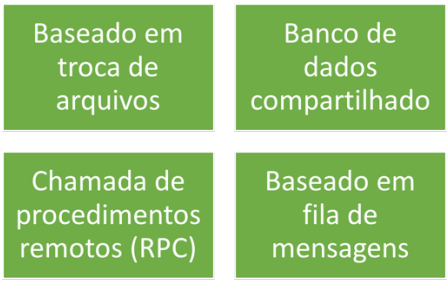

Figura 73: Estilos de integração

Embora o assunto de integração seja extenso e complexo, podemos dizer que:

- A troca de arquivos tende a ser a solução técnica mais simples, mas em sistemas complexos ela sofre de problemas de falta de controle transacional, falhas humanas e ambientes não confiáveis.
- Bancos de dados compartilhados tendem a ser simples também, mas sem uma governança adequada o seu uso indiscriminado pode trazer efeitos ruins com mudanças em uma aplicação gerando efeitos colaterais indesejados em dezenas de outras aplicações.
- Chamada de procedimentos remotos e filas de mensagens são mais complexos de serem entendidos pelo time, mas fornecem escala corporativa mais robusta. Ao mesmo tempo, também requerem governança técnica.
- Soluções de filas de mensagens são recomendadas quando buscamos tolerância a falhas e escalabilidade com baixo custo de hardware.

Vamos assumir no nosso cenário que o time que mantém o ambiente legado CICS/COBOL já conheça como expor programas COBOL através de uma API REST[223]. Com essa premissa, podemos escolher a chamada de procedimentos remotos como paradigma de integração. Isso levaria à seguinte solução genérica, que envolve que o time COBOL crie uma fachada de serviços de negócio que será chamada pelos componentes de negócio do novo portal Web.

Vamos assumir também que o time do ministério da educação em Brasília mantém um servidor FTP seguro para receber os arquivos solicitados da faculdade, que é chamado de censo acadêmico. E vamos assumir também que o fornecedor do ERP interopere através de arquivos colocados em diretório e isso seja uma restrição pois não pode ser modificado em um grande custo como o fornecedor.

[223] https://www.ibm.com/developerworks/community/blogs/e4210f90-a515-41c9-a487-8fc7d79d7f61/entry/cics_atom_support_restful_service

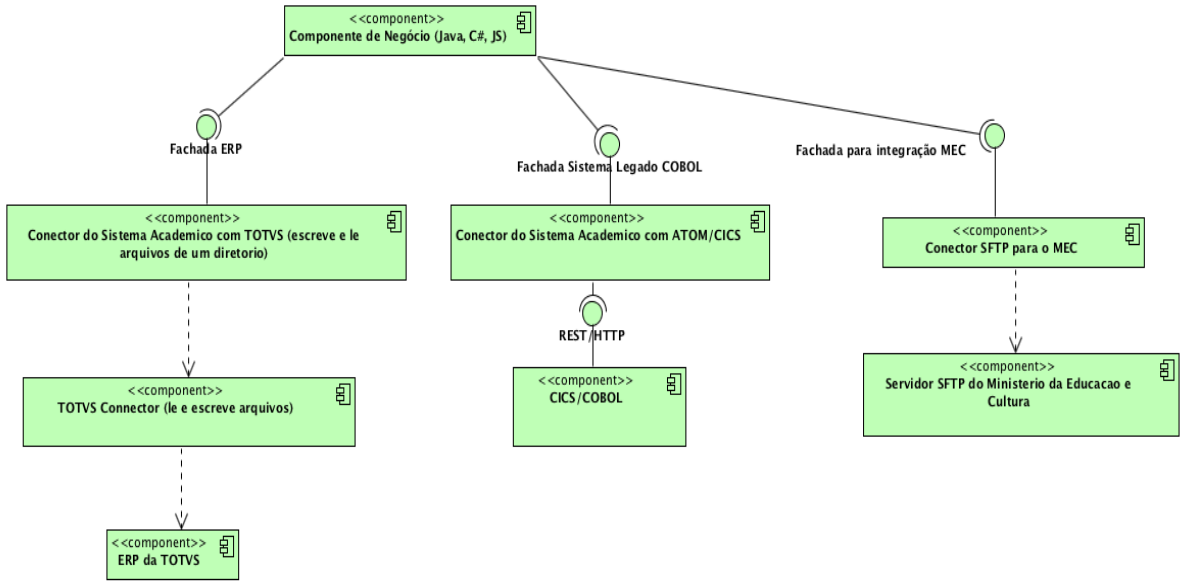

Figura 74: Visualização de Integração

Na solução apresentada, temos um conector (código Java, C# ou JS) específico por tecnologia. O primeiro conector usa as APIs de leitura e escrita de arquivos para fazer a comunicação. O segundo conector faz a chamada aos serviços REST/HTTP usando as APIs de programação dessas linguagens. O terceiro e último conector faz chamada via FTP ao servidor SFTP do governo federal.

10.11 Passo 11 – Visualização de Manutenibilidade

Dado que no nosso contexto a manutenibilidade e a testabilidade foram citados como aspectos críticos, queremos também demonstrar como isso poderá ser resolvido em cada plataforma. Isso leva a diagramas distintos nas várias tecnologias, conforme mostrado nas figuras.

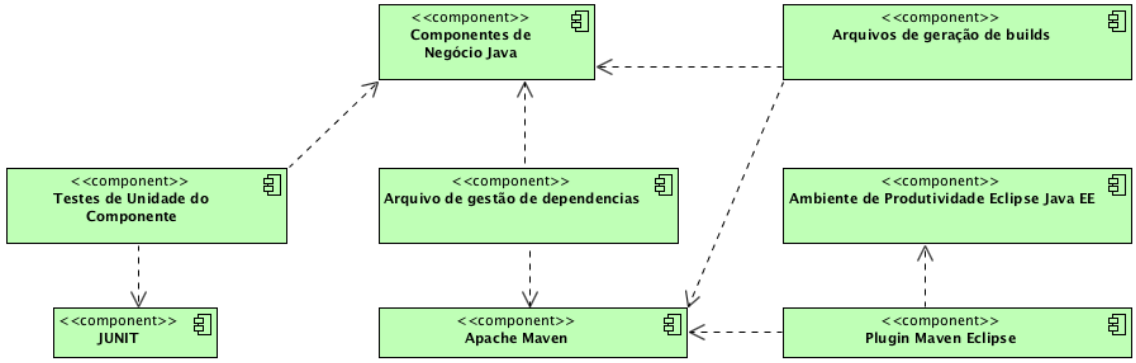

Figura 75: Visualização de Manutenibilidade em Java EE

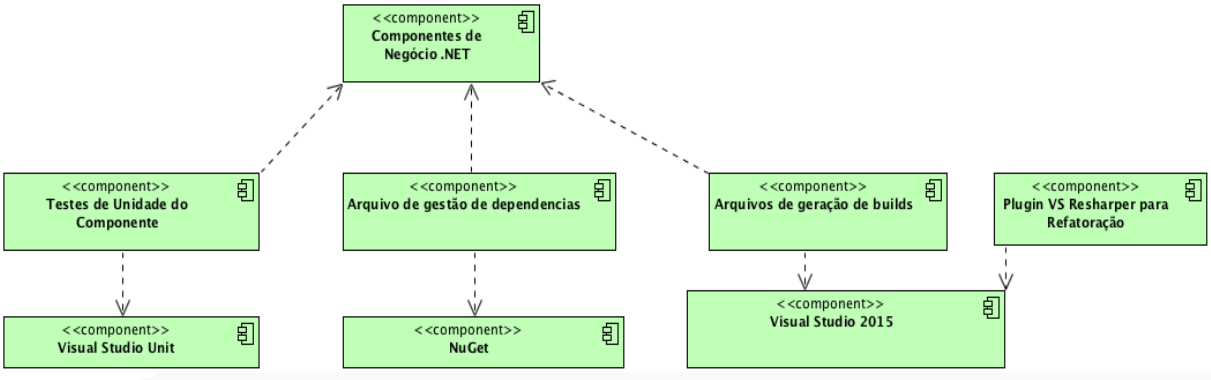

Figura 76: Visualização de Manutenibilidade em ASP.NET

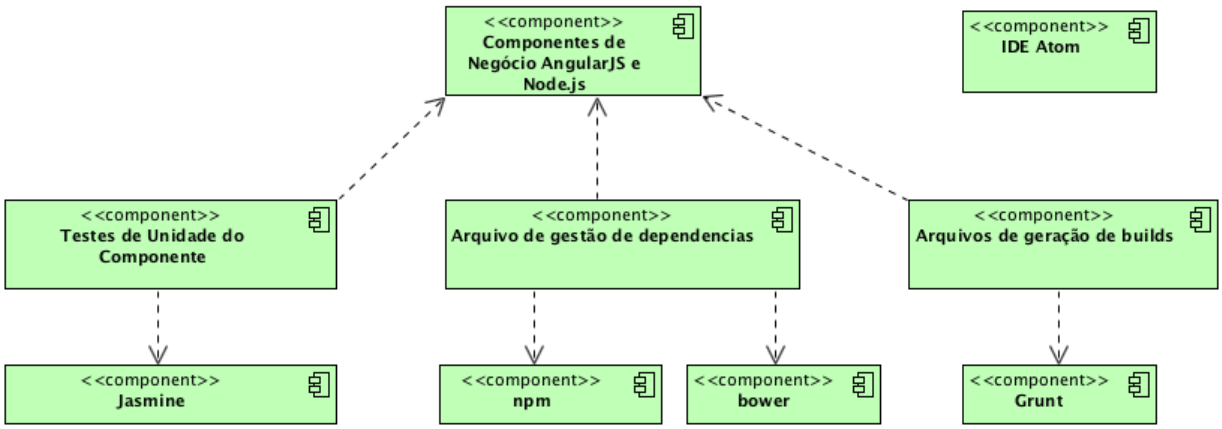

Figura 77: Visualização de Manutenibilidade em Node.js

Uma boa visão de manutenibilidade deve indicar que componentes de infraestrutura o arquiteto e seu time irá usar para apoiar os seus processos de manter e testar o seu código. Esses componentes não importam para o cliente, mas são críticos para o time de desenvolvimento em termos do seu processo técnico de desenvolvimento.

10.12 Passo 12 – Alocação de Componentes aos Nodos Físicos

O diagrama da Figura 62 mostra a topologia física. É possível também estendê-lo e mostrar como alguns componentes de software podem ser alocados a cada um desses hardwares. Embora isso seja mais útil em topologias bem mais complexas do que a trabalhada nesse exemplo, um exemplo é mostrado aqui para referência para o leitor.

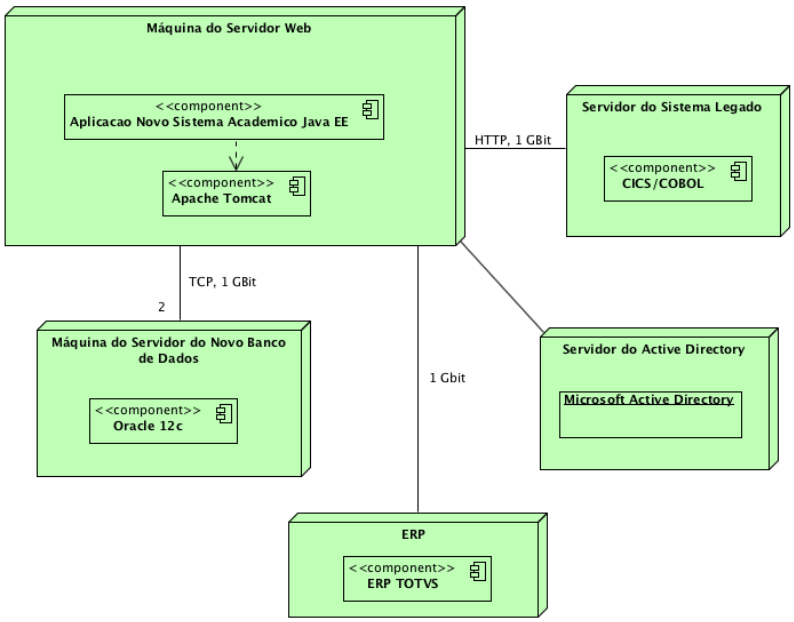

Figura 78: Topologia Física com Componentes – Visão Java EE

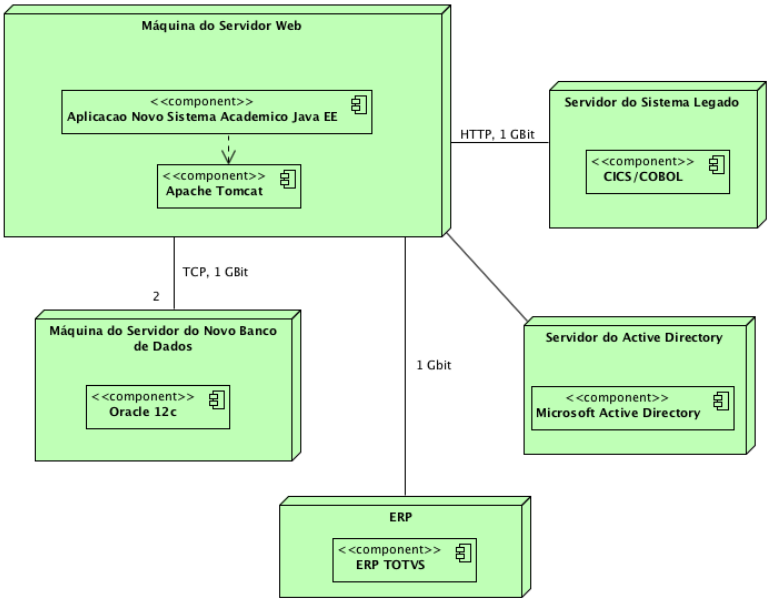

Figura 79: Topologia Física com Componentes – Visão .NET

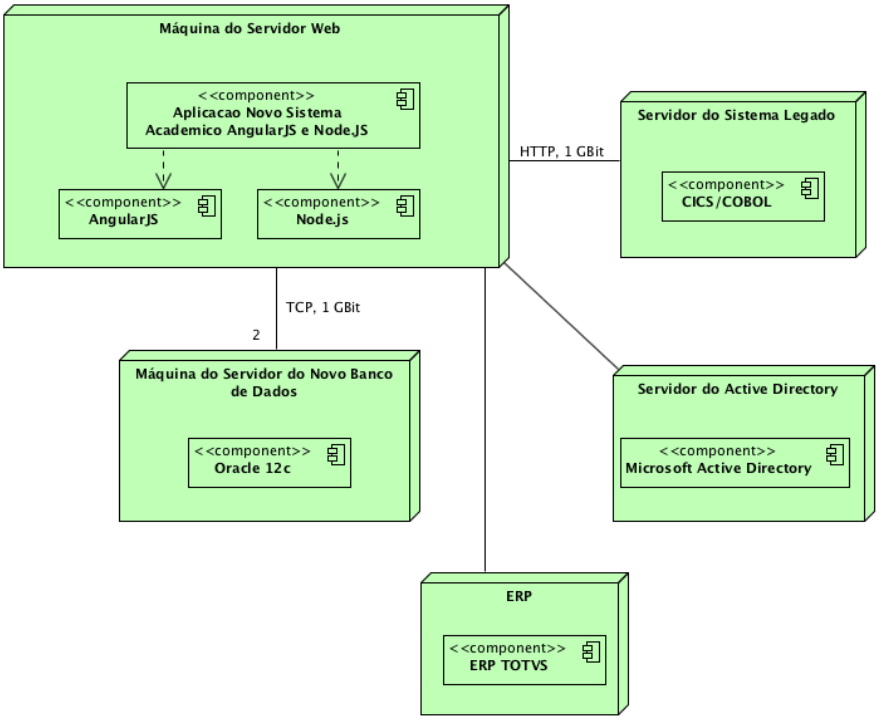

Figura 80: Topologia Física com Componentes – Visão Angular e Node.js

Recomendação Arquitetural: Utilize o conceito de pontos de vista para fazer múltiplos modelos sobre a sua arquitetura. Como pode ser observado nos passos 3-10, cada preocupação relevante do sistema acadêmico foi representada a partir de um modelo distinto. O uso de vários modelos, cada um dele pequeno e de fácil comunicação, é uma melhor prática arquitetural. Não exagere. Produza apenas documentação que tenha propósito e seja esperada pelo seu time.

10.13 Passo 13 – Determinar Provas de Conceito

Quando a arquitetura traz riscos técnicos para o time, é prudente que o arquiteto organize provas de conceito. Quais as provas de conceito vão ser realizadas é uma informação contextual pois envolve a criticidade da tecnologia para o produto, conhecimento do time e momento do projeto.

Vamos assumir, por exemplo, que a plataforma do sistema acadêmico seja desenvolvida em Node.js e a equipe ainda não tenha experiência com o uso do modulo Passport com o servidor Microsoft AD. Nesse caso, podemos ter uma prova de conceito cujo objetivo é fazer um código mínimo em Node.js que realize a autenticação de usuários com o uso do Microsoft AD, conforme a estrutura da árvore LDAP já utilizada pela equipe de segurança da informação da faculdade.

10.14 Modelos de um Documento de Arquitetura de Software

Os processos de software como o RUP[224], OpenUP[225], SEI V&B (Views and Beyond)[226] trazem já bons modelos para a documentação de arquitetura de software.
Inspirados nesses e outros processos, uma sugestão de um documento de arquitetura de software leve e centrado em modelos visuais para sistemas Web poderia ter as seguintes seções.

- Visualização de Negócio
- Condutores Arquiteturais
- Estilo Arquitetural
- Plataforma Tecnológica
- Visualização Lógica
- Visualização Física
- Visualização de Persistência
- Visualização de Apresentação (Usabilidade e Acessibilidade)
- Visualização de Segurança
- Visualização de Interoperabilidade
- Visualização de Manutenibilidade
- Visualização dos Componentes Alocados aos Nodos Físicos
- Organização das Provas de Conceito

Ao mesmo tempo, se você e o seu time trabalham com métodos ágeis e não possuem necessidades formais de gestão documental, use quadro brancos e canetas para fazer os seus desenhos e gerar bons debates. Depois tire fotos com o seu telefone celular para capturar e disseminar o conhecimento.

[224] https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf
[225] http://epf.eclipse.org/wikis/openuppt/
[226] http://www.sei.cmu.edu/architecture/tools/document/viewsandbeyond.cfm