SMARTCLASS API – TORNANDO A COMUNICAÇÃO MAIS INTELIGENTE NO AMBIENTE ACADÊMICO

Antônio O. Junior – antonio.jr004@gmail.com
Cleverson V. Nahum – cleversonahum@ufpa.br
Derian F. A. Alencar – derian.sobral@gmail.com
Eduardo G. L. Silva – eduardolima.ufpa@gmail.com
Universidade Federal do Pará, Instituto de Tecnologia, Faculdade de Engenharia da Computação e Telecomunicações.
Rua Augusto Correa, Nº 01, Cidade Universitária Prof. José da Silveira Neto, Guamá.
66075-110 – Belém – Pará

Resumo: *O projeto SmartClass API visa driblar as dificuldades enfrentadas na comunicação do aluno dentro e fora do meio acadêmico, impossibilitada, muitas vezes pela ausência de tecnologias ou sistemas pouco eficazes. Para isso, o projeto consiste em uma API, que utiliza ferramentas open source presentes em muitas aplicações atuais, tais como, virtualização em forma de containers com o Docker, banco de dados com o PostgreSql e Framework NodeJS.*

Palavras-chave: *Api, Docker, Nodejs*

1 INTRODUÇÃO

O conceito de cidades inteligentes vem ganhando força nos últimos anos por suas características inovadoras com a proposta de melhorar a qualidade de vida de seus habitantes, através de rotinas que permitem as pessoas usar melhor seu tempo e facilitar a difusão de informações relevantes.

Existem várias ferramentas que podem ser usadas para a implementação eficiente de um "ambiente inteligente", uma que vem ganhando destaque nos últimos anos é a Internet das Coisas - IoT [Gubbi et al., 2013]). Para fazer isso na prática, torna-se necessário um sistema robusto capaz de armazenar essas informações e que pode ser facilmente acessado e manipulado por meio de um padrão conhecido por todos os que usarão sua interface. Um ambiente inteligente é definido como um sistema onde a computação é penetrante e usada para melhorar a experiência humana, como para a partilha de útil de dados ou algo mais como um sistema de tomada de decisão [Cicirelli F & Vinci, 2016].

Este trabalho propõe uma API de Sala de Aula Inteligente, que administra informações relacionadas a horários de aula, atividades, informações relevantes sobre cursos universitários, e quaisquer outros assuntos relacionados a um ambiente tanto escolar como acadêmico, foi escrita no Node.JS [Tilkov & Vinoski, 2010] que acessa um banco de dados construído no PostgreSQL, ambos armazenados em containers criados utilizando Docker [Airton Lastori, 2015]. Inicialmente, a API surge para atender as necessidades do projeto denominado SmartClass ou Sala de Aula Inteligente, dentro da disciplina de Redes de Computadores 2 do Curso de Engenharia da Computação da Universidade Federal do Pará, onde a API era responsável por fazer o gerenciamento dos dados por meio das requisições, juntamente com

uma rede MQTT (Message Queue Transporte de Telemetria) [Hunkeler et al., 2008] que utilizava sensores para coletar informações sobre o ambiente acadêmico e meteorológico e um aplicativo móvel que mostrava tais informações aos usuários, facilitando a comunicação entre o aluno e os assuntos acadêmicos.

Desse modo, o objetivo da SmartClass API é propor uma ferramenta adaptável que pode ser modificada e portada para qualquer outro ambiente de interesse com um mínimo necessário de codificação e configuração, em outras palavras, é uma ferramenta que pode ser implantada em qualquer meio estudantil. Dispondo do código livre para acesso na plataforma Github, cujo nome do projeto é SmartClass-UFPA/server.

Agora vamos explicar como a API foi desenvolvida. Este artigo está organizado da seguinte forma: a Seção 2 mostra uma visão geral sobre as ferramentas e conceitos utilizados no projeto. A Seção 3, descreve a construção do banco de dados e API de Sala de Aula de Inteligente. A Seção 4, mostra os resultados obtidos da API sendo utilizada no projeto SmartClass em uma turma na UFPA – Campus Belém, e na Seção 5, a conclusão obtida a partir deste projeto.

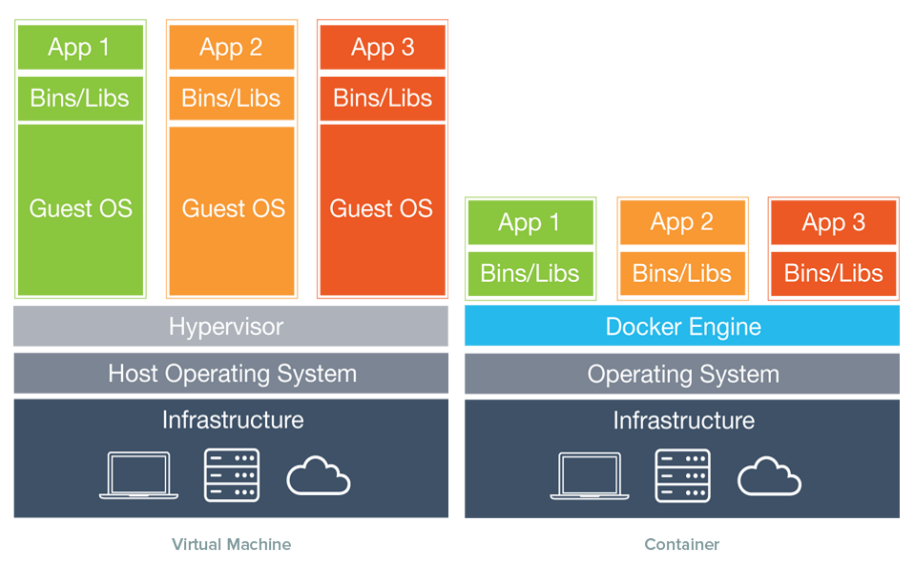
2 VISÃO GERAL

Antes da darmos início a abordagem de funcionamento da API, explicaremos melhor as ferramentas utilizadas no projeto de forma a deixar mais clara os seus funcionamentos e objetivos que serão discutidos nas demais subseções.

2.1 Container docker e sua rede

Os containers são um tipo de virtualização no sistema operacional para executar alguns sistemas diferentes dentro de um host de controle [Merkel, 2014]. Esse conceito é comumente confundido com o conceito de máquinas virtuais, de certa forma há semelhanças, mas eles têm grandes diferenças. Os containers utilizam o mesmo núcleo para todos os containers; é possível ganhar muito desempenho, comparando com uma máquina virtual (VM), por exemplo, onde precisa ter um kernel para a máquina e todas as estruturas de um sistema operacional em cada um, os containers têm um desempenho melhor do que as máquinas virtuais. Assim, o container não é uma máquina virtual, eles oferecem um ambiente virtual com CPU, memória, bloco de E / S, rede, por exemplo [Jaison 7 Kavitha, 2016]. Na "Figura 1", há um comparativo entre máquinas virtuais e containers, ela mostra no lado direito que um container possui um mecanismo docker sobre um sistema operacional do host que faz o controle sobre os containers, enquanto no lado esquerdo da figura, há uma ilustração de uma máquina virtual com um Hypervisor sobre o sistema operacional host controlando todos os componentes de outros sistemas operacionais completos, tornando o processo mais complexo.

Figura 1 - Comparação entre a arquitetura de uma máquina virtual e containers.



O Docker é o mecanismo mais utilizado e popular que proporciona ao usuário o controle de containers, possui um sistema completo para ter controle sobre diferentes aspectos dos containers, ele pode ser usado em Linux, Mac e Windows. Possui uma biblioteca com imagens de sistemas operacionais denominados Docker Hub, com essas imagens é possível iniciar um novo sistema e configurá-lo com base nessas imagens [F e S., 2016]. Este processo é muito importante, porque introduz um aumento no desempenho e flexibilidade no gerenciamento de servidores.

Com o docker é possível distribuir melhor os recursos em rede e gerencia-los. Docker usa o conceito “Container as a Service” (CaaS), cada container representa um serviço na rede, assim é mais simples controlar os recursos. A Docker Network é uma função do docker que faz com que as redes entre os containers se liguem tanto a outros containers quanto a outros hosts e hosts externos [Dusia A & M, 2015]. Com isso, cada container pode estabelecer comunicação com outros, porque eles estarão na mesma rede (uma rede de docker local). A rede docker oferece três tipos de redes, o primeiro é o modo Ponte (Bridge), ele cria uma sub-rede para o container e é acessível apenas para outros containers na rede ou no host. Outra opção é um modo de host, essa rede tem como objetivo entregar para o container todas as interfaces existentes no docker host, o container pode ser acessado para hosts externos. O último modo é o modo None que isola o container e não pode estabelecer comunicação com ninguém. Na base disso, o host docker usa NAT para tornar o acesso externo aos containers.

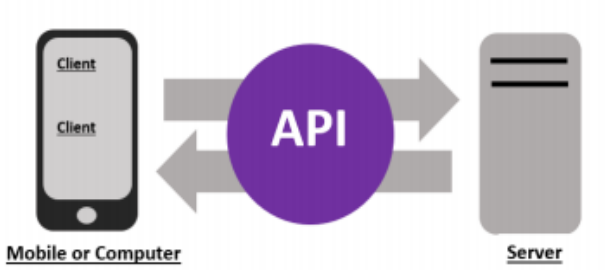
Outra ferramenta importante é o Docker-Compose, com ela, mais de um container pode ser iniciado ao mesmo tempo usando um comando, a configuração está em um arquivo docker-compose onde está relacionada as portas usadas, redes, volumes e outras características do ambiente virtual a ser criado. Assim, com o Docker-Compose, um sistema completo pode ser iniciado com todos os links e informações necessárias.

2.2 Api rest

A API é uma maneira de conectar o banco de dados da web com seu aplicativo de forma simples. O uso de APIs é bastante comum porque, depois de configurado, pode ser lido por qualquer tipo de aplicativo apenas fazendo requisições HTTP. O uso de APIs para tornar a comunicação entre o banco de dados e sua aplicação é muito eficaz, caso contrário, o banco

de dados seria vulnerável a qualquer tipo de invasão ou erro de código. Basicamente o seu programa funciona enviando pedidos para a API e ela lida com essas informações e as envia de volta ao banco de dados, ele envia a informação que é tratada pela API e sua aplicação, geralmente no formato JSON, que é leve e fácil de ler. A "Figura 2" representa o papel de uma API.

Figura 2 - Funcionamento de uma API.



A arquitetura REST define os princípios que os Serviços da Web precisam seguir. Concentra-se no sistema de conteúdo, incluindo a forma como os estados de recursos são abordados e transferidos através de HTTP por uma grande quantidade de clientes escritos em diferentes idiomas [Rodriguez, 2008]. Seguindo as definições REST, o serviço web torna-se mais flexível e versátil, pois todas as aplicações que entendem o REST podem manter uma comunicação.

Já o Node.js é um mecanismo Javascript que permite desenvolver aplicativos para servidores e tem uma biblioteca chamada Express para criar serviços web HTTP, as bibliotecas e dependências são instaladas pelo NPM (Node Package Manager) que é um gerente de dependências do Javascript [NodeBr, 2017], tornando o desenvolvimento mais simples e ao mesmo tempo acaba por deixar as configurações mais complexas, tais como o roteamento de páginas e a codificação. Atualmente, a maioria das aplicações está usando uma notação de objeto do Javascript chamada JSON para estabelecer comunicação entre diferentes tipos de aplicativos para compartilhar informações.

2.3 Postgresql

Structured Query Language - SQL, é uma linguagem projetada para gerenciar relacionamentos de bancos de dados. Consiste em uma linguagem de definição de dados, linguagem de manipulação de dados e linguagem de controle de dados, o SQL é baseado em álgebra relacional e cálculo relacional. Com o SQL, podemos incluir dados, atualizar, consultar, criar e modificar esquemas e controle de acesso a dados. Isso faz do SQL uma linguagem muito simples e poderosa e tão usada em todo o mundo.

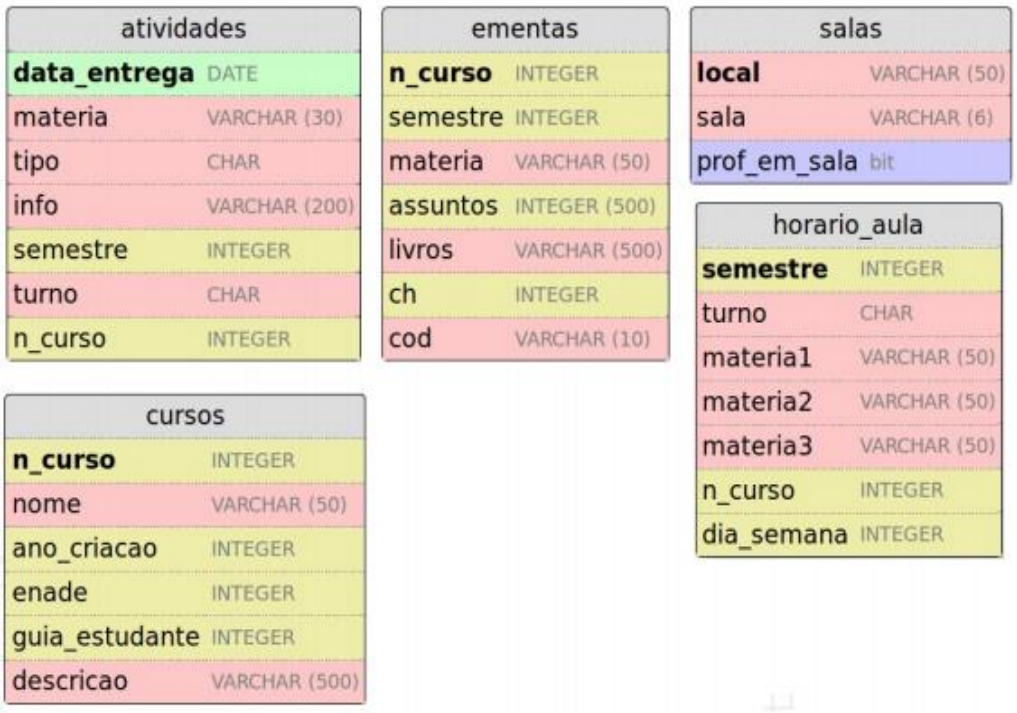
O PostgreSQL é um sistema de banco de dados objeto-relacional. PostgreSQL possui código fonte aberto e fornecer mais recursos, tais como; Pesquisas complexas, chaves estrangeiras, gatilhos e linguagem processual para tornar as adições de API mais simples. Essas melhorias atendem aos nossos requisitos.

3 A CONSTRUÇÃO DA API E BANCO DE DADOS

O servidor criado é dividido em duas partes que são o banco de dados Postgre e Node.js Server com API REST. O banco de dados possui cinco tabelas com informações sobre

atividades, notas, aulas, horários e cursos, as informações detalhadas sobre as tabelas estão na "Figura 3". O servidor tem a responsabilidade de estabelecer comunicação entre aplicativos e banco de dados de forma simples e padronizada para facilitar o acesso a informações por aplicativos externos que tenham autorização para o acesso. O servidor foi construído usando o mecanismo de JavaScript chamado Node.js, que permite que você desenvolva aplicativos no lado do servidor usando o Javascript. Foi utilizada a biblioteca Express para Node.js para criar a API REST para responder solicitações HTTP feitas pelos aplicativos externos.

Figura 3 - Ilustração das tabelas do banco de dados, com campos e tipos de dados.



Foi construído controladores específicos para cada tabela no banco de dados de acordo com as necessidades de aplicativos externos. Por exemplo, na tabela de informações sobre cursos (a tabela "Cursos") possui um controlador com funções para adicionar, atualizar, ler e excluir entradas na tabela através de solicitações HTTP feitas por agentes externos. A tabela 1 mostra alguns dos controladores utilizados no servidor e o caminho utilizado para fazer transações no banco de dados.

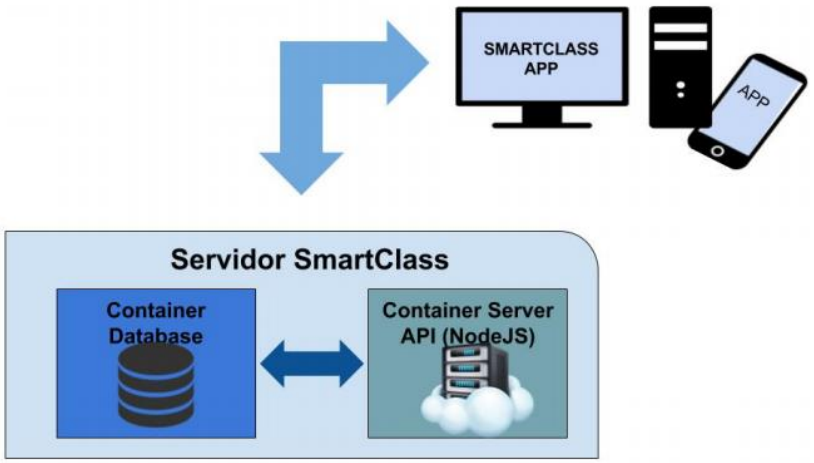
Tabela 1 - Alguns controladores utilizados na API na transação com o banco de dados.

Controladores de Sala de Aula		
/sala/	POST – Add uma sala de aula	
/sala/$local/$sala	GET – pega informações sobre uma sala	DELETE – Deleta a sala
/sala/$local/$sala/$status	POST – Muda o Status se o professor estiver na sala de aula	
Controladores de Atividades		
/atividades/$n_curso/$semestre/$turno	GET – Pega as atividades de um curso turno	DELETE – Deleta as atividades do curso turno

/atividades/$n_curso/$semestre/$turno/$materia	GET – Pega as atividades de uma matéria	POST – Add uma nova atividade na matéria
Controladores de Cursos		
/curso/	GET – Pega Informações sobre todos os cursos do banco	
/curso/$n curso/	GET – Pega informações de um curso	POST – Add um novo curso
Controladores de Horários de Aula		
/horarios/	POST –Add um horário de aula	GET – Pega as atividades de uma turma
/horarios/$n_curso/ $semestre/$turno	GET – Pega horário de aula de uma turma	PUT – Atualiza o horário de aula
	DELETE – Deleta o horário de aula de uma turma	

Tanto o servidor com API REST e o servidor PostgreSQL foram implementados usando contêiner Docker. O container com banco de dados usou a imagem oficial do PostgreSQL chamado "postgre" e o container com Node.js usou a imagem oficial de Node.js chamada "node", essas imagens estão disponíveis no Docker Hub. O uso de containers garante flexibilidade, escalabilidade e tornam o controle mais simples. Os links em container com servidor e banco de dados foram criados usando redes docker, que permitem estabelecer uma rede virtual deixando um grupo isolado em relação a outros containers de forma a melhorar a disponibilidade e a segurança. Na "Figura 4" há uma ilustração sobre as redes de containers, suas relações e alguns parâmetros do Docker, como rede, volumes e portas, por exemplo, tomando como base o projeto SmartClass-UFPA.

Figura 4 - Relação entre containers no projeto SmartClass-UFPA.



Para implantar toda a construção do sistema, foi utilizado a ferramenta Docker-Compose com um arquivo de configuração com informações sobre o cenário necessário para o servidor e os containers de banco de dados e informações sobre a rede, portas e volumes usados para eles. Desta forma, todo o sistema pode ser implantado usando apenas um comando no Docker Client, tornando o sistema mais simples para ser implantado. Depois de iniciar o sistema completamente, pod-se fazer solicitações ao servidor da API para fazer transações com o banco de dados usando solicitações HTTP.

4 A API E O PROJETO SMARTCLASS-UFPA

O sistema SmartClass foi implementado em uma sala de aula da UFPA (Universidade Federal do Para) no campus Belém, onde os alunos começaram a usar a aplicação da SmartClass na vida acadêmica para facilitar o acesso a informações sobre aulas e atividades, uma vez que um grande problema dos estudantes na universidade é a informação não localizada. Com o uso da aplicação SmartClass, o desempenho dos alunos foi melhorado, segundo pesquisa feito a alguns alunos, porque, com o uso, tornou-se mais simples conhecer informações sobre testes, trabalhos de casa, projetos, horários de aula e informações sobre o ambiente em torno das salas de aula, como temperatura, clima e se professor está presente ou ausente na sala de aula, pois agora essas informações podem ser encontradas no aplicativo móvel SmartClass.

O servidor com banco de dados e API foram implantados em um computador pessoal que funciona como servidor para fazer os testes, respondendo as requisições de aplicativos e armazenando informações em um banco de dados postgreSQL. Todos os códigos desenvolvidos e o sistema completo estão no GitHub em um repositório público localizado em [SmartClass-UFPA 2017]. As informações na base de dados são alimentadas por estudantes que podem melhorar os dados e adicionar novas informações no banco de dados através da API ou do uso próprio do aplicativo, melhorando a utilização da aplicação. Seguindo o conceito de Redes e Sistemas Colaborativos, uma interface de aplicativo móvel está em um “Figura 5” ilustra a primeira versão do aplicativo móvel integrado a rede MQTT e a API com o banco de dados. As informações fornecidas por sensores de temperatura e clima foram implementadas através de sensores conectados à rede MQTT que foram instalados na sala de aula escolhida para fazer os testes. Alguns dados já foram ou podem ter sido aprimorados por estudantes, já que o projeto é de código aberto, todos podem desenvolver novos recursos e funções no sistema SmartClass, assim como na API, contribuindo para tornar-se um sistema mais confiável e adaptável a cada necessidade.

Figura 5 - Aplicativo SmartClass-UFPA.

5 CONSIDERAÇÕES FINAIS

Considerando o cenário atual, onde há um número maciço de aplicações que resolvem diversos problemas do nosso cotidiano, surge a ideia de solucionar o problema da informação descentralizada em um campus universitário de grande extensão, composto por diversos alunos de diversas áreas. Contudo, muitas universidades e escolas distribuem as informações pertinentes aos alunos, porém, muitas vezes feitas em forma de aviso ou comunicado por e-mail. Em meio a esse cenário, a construção da SmartClass API visa amenizar tal problema, uma ferramenta que seria de fácil implementação em qualquer aplicação que visa solucionar os problemas de comunicação no ambiente acadêmico.

O projeto SmartClass-UFPA foi o primeiro a utilizar a API com o banco de dados, embora tenha sido aplicado a uma porção pequena de alunos da universidade, a pesquisa feita com cerca de 30 alunos e 4 professores, constatou que houve uma melhora significativa no desempenho e comunicação dos alunos, assim como o aumento no número de presenças nas aulas, já que o aplicativo indica a presença do professor em sala de aula, bem como a mudanças de sala.

Um grande problema apresentado se dá no mesmo das novas tecnologias e ideias, já que as pessoas ainda não estão acostumadas a usá-las e acabam por não as utilizar como deveria ou mesmo esquecer de usá-las. Mas considerando todas as vantagens que o sistema pode oferecer às universidades de grande extensão, espera-se que os alunos vejam o desenvolvimento e os benefícios que seus colegas que já utilizam o sistema têm e acabam usando o mesmo.

Dado o cenário tecnológico atual, tanto pela área de desenvolvimento quanto pela área de segurança da informação, pode-se dizer que o futuro da API e do banco de dados necessita de uma melhora significativa na área de segurança, usando, por exemplo, o protocolo SSL para solicitações da aplicação com a API e criptografia do código para evitar o roubo de dados, uma vez que existe um grande fluxo de informações através da aplicação. Além de expandir os métodos de compartilhamento de dados, com o objetivo de diminuir a necessidade de usar a Internet.

Agradecimentos

Agradecemos ao Professor Dr. Eduardo Coelho Cerqueira pela orientação e apoio na construção desse trabalho.

A equipe do projeto SmartClass-UFPA pelo apoio e utilização deste projeto em seu aplicativo android.

REFERÊNCIAS BIBLIOGRÁFICAS

Artigos de jornais:
MERKEL, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux Journal, 2014.p.239.

Internet:
IMASTERS, Airton Lastori (2015). **Utilizando Docker com MySQL.** Disponível em: < https://imasters.com.br/banco-de-dados/mysql/utilizando-docker-com-mysql/?trace=1519021197&source=single> Acesso em: 15 fev. 2017.

NodeBr (2017), Admin. **O que é o NPM do Node.JS**. Disponível em: < http://nodebr.com/o-que-e-a-npm-do-nodejs/> Acesso em: 20 fev. 2017.

SMARTCLASS-UFPA/SERVER. **Repositório Github**. Disponível em: < https://github.com/SmartClass-UFPA/server/> Acesso: 20 abril 2017.

Trabalhos em eventos
DUSIA A, Y. Y. and M, T. (2015). Network quality of service in docker containers - IEEE International Conference on Cluster Computing.

JAISON, A. and KAVITHA, N. (2016). Docker for optimization of cassandra nosql deployments on node limited clusters - International Conference on Emerging Technological Trends [ICETT].

TILKOV, S. and VINOSKI, S. (2010). Node. js: Using javascript to build high-performance network programs - IEEE Internet Computing, 14(6):80–83.

SMARTCLASS API – MAKING COMMUNICATION MORE INTELLIGENT IN ACADEMIC ENVIROMENT

Abstract: *The SmartClass API project aims to face difficulties around communication of the student inside and outside the academic environment, being hard confront by the absence of technologies or systems. Thus, the project consists in an API, that uses open source tools present in many current applications, such as, virtualization containers using Docker, ProstgreSQL database and NodeJS framework.*

Key-words: *Api, Docker, Nodejs.*