

Chamada Virtual

Equipe:
Eduardo costa
Davi wesley
Neto castro

1.0 Introdução

Este documento especifica os requisitos do sistema *ChamadaVirtual*, fornecendo aos desenvolvedores as informações necessárias para o projeto e implementação, assim como para a realização dos testes e homologação do sistema.

1.1 Visão geral do documento

1.1

Além desta seção introdutória, as seções seguintes estão organizadas como descrito abaixo.

Seção 1.2 – Descrição geral do sistema: Apresenta uma visão geral do sistema, caracterizando qual é o seu escopo e descrevendo seus usuários.

Seção 1.3 – Requisitos funcionais (casos de uso): especifica as funcionalidades em geral do sistema.

Seção 1.4 – Requisitos não-funcionais: Especifica todos os requisitos não funcionais do sistema, divididos em requisitos de usabilidade, confiabilidade, desempenho, segurança, distribuição, adequação a padrões e requisitos de hardware e software.

Seção 2.1 – Especificação: Define as técnicas, critérios e metodologias usadas no projeto.

Seção 3.1 – Estratégia(s) de Teste: Define os cenários de testes das funcionalidades, as atividades de testes que serão conduzidas, as técnicas de testes que serão usadas, ferramentas e seus objetivos.

Seção 4.1 – Projeto de Casos de Teste: O Caso de Teste descreve uma condição particular a ser testada e é composto por valores de entrada, restrições para a sua execução e um resultado ou comportamento esperado.

Seção 5.1 – Execução: Define como os testes foram executados, as ferramentas que foram usadas e o ambiente de testes.

Seção 6.1 – Análise dos Resultados e Próximos Passos: Conclusão e considerações sobre os testes e lições aprendidas

1.2 - Descrição geral do sistema

1.2.1- Abrangência e sistemas relacionados

O programa *ChamadaVirtual* é um aplicativo para dispositivos mobile que auxilia a marcação de presença em sala de aula, onde o professor acessa o contato aplicativo através do smartphone para realizar a chamada da turma selecionada.

Diante da facilidade de se definir estratégias metodológicas, o *ChamadaVirtual* contribui de modo decisivo para melhorar a qualidade e a facilidade da marcação de presença.

1.3 - Requisitos funcionais

1.3.1 Cadastrar Turma

Descrição: Deve permitir o professor cadastrar uma turma com seus dados: nome, dias das aulas, horário, curso.

Cadastrar Aluno

Descrição: Deve permitir o professor cadastrar um aluno com seus dados: nome, matrícula, curso.

1.3.2 Gerar relatório de presença do aluno

Descrição: Deve permitir o professor verificar o número de faltas de um aluno.

1.3.3 Contar o número de presença do aluno

Descrição: O sistema deve contar o número de presença de um aluno em uma turma.

1.4 - Requisitos não-funcionais

1.4.1 Instalar

Descrição: O sistema está disponível para download na loja online de aplicativos oficial do sistema operacional em questão.

1.4.2 Segurança

Descrição: Apenas usuários com privilégios de acesso poderão manter dados sobre a turma e visualizar históricos.

1.4.3 Tempo de Resposta

Descrição: O sistema tem um tempo limite de resposta para autenticação do usuário de 3 a 4 segundos, para persistência de dados cadastrados 3 segundos e para validação das chamadas, 3 segundos.

1.4.4 Disponibilidade

Descrição: O sistema estará disponível para o aluno durante o período de tempo da aula cadastrado pelo professor responsável.

1.4.5 Tipo de Interface

Descrição: O sistema deverá ser acessado completamente via aplicação móvel.

2.1 – Especificação

2.2 – Testes usados

No sistema ChamadaVirtuais foram usadas os seguintes testes: Teste de Unidade, Casos de Testes e Teste de Integração, Teste de Segurança, Teste de Estresse, Teste Estrutural (Cobertura) e Análise automatizada do programa

2.2 – Testes não usados

2.2.1 Verificação Formal

Dado que muitos engenheiros de software não têm experiência em matemática discreta e em lógica e o tempo curto dos integrantes da equipe esse teste não foi usado

2.2.3 Verificação de modelos

Problema da explosão do espaço de estado (quando o sistema sendo verificado tem vários componentes de interação, ou estruturas de dados com vários possíveis valores

2.2.4 Revisões e Inspeções

Técnicas de inspeção podem somente verificar a correspondência entre o software e sua especificação não permite validar com o cliente(professor) as entregas uma vez que houve somente duas entregas, as revisões em pares foi impossibilitada pela alocação diferente dos membros da equipe e pela a inexperiência da equipe das tecnologias usadas

3.1 – Estratégia(s) de Teste

Ferramentas

Foram utilizadas ferramentas de testes para a linguagem de programação python, dentre elas estão os pacotes: unittest, modelmommy

3.2 Teste de unidade

É toda a aplicação de teste nas assinaturas de entrada e saída de um sistema. Uma unidade é a menor parte testável de um programa. de computador podendo ser uma unidade ou uma função individual ou um procedimento.

TU001 - Calcular falta

Calcula a quantidade de faltas de um aluno específico previamente com faltas cadastradas, retorna-se às faltas computadas segundo a seguinte fórmula: total de faltas = faltas * 2

3.3 Teste de Performance

Testes de performance são testes nos quais uma aplicação é submetida a uma carga de trabalho dentro de condições específicas por um tempo determinado com o objetivo de verificar os comportamentos diferentes que essas condições e cargas podem proporcionar. Com estes procedimentos, podemos observar o comportamento de todo o ambiente da aplicação, desde os defeitos de desempenho.

3.4 – Análise Automatizada de Programa

É o uso de software para controlar a execução de teste de software através da aplicação de estratégias e ferramentas, comparando os resultados esperados com os resultados reais. Seus objetivos são a redução do envolvimento humano em atividades manuais, de tempo demandado e de custo final.

3.5 Caso de teste

É um conjunto de condições usadas para teste de software ele pode ser elaborado para identificar defeitos na estrutura interna do software por meio de situações que exercitem os caso de teste deve especificar a saída esperada e os resultados esperados do processamento.

```
class TestDatabase(TestCase):
    # fixtures = ['datadumped']

    def setUp(self):
        self.student = mommy.make('api.Student', name='Mikasa',
                                   id_subscription=381097)
        self.client = APIClient()

    def test_student_creation(self):
        self.assertTrue(isinstance(self.student, Student))
        self.assertEqual(self.student.__str__(), self.student.name)

    def test_student_name(self):
        # Arrange
        student = self.student
        # Act
        name = student.name
        # Assert
        self.assertEqual(name, "Mikasa")
```

Figura 1 - Implementação do CT001

```

class Student(models.Model):
    COURSES = (('Engenharia de Software', 'Engenharia de Software'),
               ('Ciências da Computação', 'Ciências da Computação'),
               ('Engenharia da Produção', 'Engenharia da Produção'),
               ('Engenharia Civil', 'Engenharia Civil'),
               ('Engenharia Mecânica', 'Engenharia Mecânica'),)

    name = models.CharField(max_length=55, verbose_name='nome', unique=True)
    id_subscription = models.IntegerField(primary_key=True,
                                         verbose_name='matricula')
    course = models.CharField(max_length=30, choices=COURSES,
                             verbose_name="Curso")
    owner = models.ForeignKey('auth.User', on_delete=models.CASCADE)

    class Meta:
        verbose_name = 'Aluno'

    def __str__(self):
        return self.name

```

Figura 2 - Classe que o CT001 testa

CT001 cadastro de um aluno

Ele cadastra um aluno em banco de dados é verificar se o cadastro foi realizado corretamente no banco de dados é verificando a existência do próprio e realizar o cadastro.

CT002 verificação de nome de um aluno

Dado um aluno existente após ser feita uma pesquisa por nome. caso exista um registro desse aluno o sistema deve informar a existência do próprio e realizar o cadastro .

CT003 Cadastro de disciplina

Ele cadastra um disciplina em banco de dados é verificar se o cadastro foi realizado corretamente no banco de dados é verificando a existência do próprio e realizar o cadastro

CT004 Cadastro de professor

Ele cadastra um professor em banco de dados é verificar se o cadastro foi realizado corretamente no banco de dados é verificando a existência do próprio e realizar o cadastro.

CT005 Cadastro de faltas

Dado um aluno existente no banco de dados, informa a disciplina que o aluno está cadastrado e o número de faltas e realizar o cadastro.

CT006 Cadastro de usuário no sistema

informar o nome do usuário, email, senha e confirmação de senha e realizar o cadastro.

3.6 Teste de integração

O propósito do teste de integração é verificar os requisitos funcionais, de desempenho e de confiabilidade na modelagem do sistema. Com ele é possível descobrir erros de interface entre os componentes do sistema.

```
def test_api_create_student(self):
    # Arrange
    url = '/api/alunos'
    data = {
        "name": "Annie Leonheart",
        "id_subscription": 432634,
        "course": "Engenharia de Software"
    }
    # Act
    response = self.client.post(url, data, format='json')
    # Assert
    self.assertEqual(response.status_code, 201)
```

Figura 3 - Implementação para o TI001

```
class StudentSerializer(serializers.ModelSerializer):
    """ Serialzize Student model."""
    owner = serializers.ReadOnlyField(source='owner.username')

    class Meta:
        model = Student
        fields = ('name', 'id_subscription', 'course', 'owner')
```

Figura 4 - Classe que o TI001 está testando

TI001 Cadastro de aluno usando API

Dado um acesso com credenciais informados para cadastrar de aluno no sistema usando a API REST com as informações requisitadas do sistema realiza-se o cadastro.

TI002 Cadastro de disciplina usando API

Dado um acesso com credenciais informados para cadastrar uma disciplina no sistema usando a API REST com as informações requisitadas do sistema realiza-se o cadastro.

TI003 Cadastro de faltas usando API

Dado um acesso com credenciais informados para cadastrar as faltas no sistema usando a

API REST com as informações requisitadas do sistema realiza-se o cadastro.

3.7 Teste de segurança

Tem como meta garantir que o funcionamento da aplicação esteja exatamente como especificado. Verifica também se o software se comporta adequadamente mediante as mais diversas tentativas ilegais de acesso, visando possíveis vulnerabilidades.

TS001 Requisição de alunos cadastrado sem autenticação

Dado um acesso sem credenciais informados para requisição de alunos cadastrado o sistema informar que não existe um usuário logado.

TS002 Requisição de um aluno informando um ID sem autenticação.

Dado um acesso sem credenciais informados para requisição de alunos por um ID o sistema informar que não existe um usuário logado.

TS002 Requisição de faltas lançadas sem autenticação.

Dado um acesso sem credenciais informados para requisição de faltas cadastrado o sistema informar que não existe um usuário logado.

TS003 Requisição de faltas lançadas por ID sem autenticação.

Dado um acesso sem credenciais informados para requisição de faltas por um ID o sistema informar que não existe um usuário logado.

TS004 Requisição de disciplina sem autenticação.

Dado um acesso sem credenciais informados para requisição de disciplina cadastrado o sistema informar que não existe um usuário logado.

TS005 Requisição de disciplina por ID sem autenticação.

Dado um acesso sem credenciais informados para requisição de disciplina por um ID o sistema informar que não existe um usuário logado.

TS006 Requisição de aluno com autenticação.

Dado um acesso com credenciais informados para requisição de um alunos cadastrado no sistema informar acesso autorizado.

TS007 Requisição de um aluno informando um ID com autenticação.

Dado acesso com as credenciais informados para requisição de alunos por um ID o sistema informar que esse aluno tem acesso autorizado.

TS008 Requisição de faltas lançadas com autenticação.

Dado acesso com as credenciais informados para requisição de faltas cadastrado o sistema informar as faltas que constam no aluno desejado,.

TS009 Requisição de faltas lançadas por ID com autenticação.

Dado acesso com as credenciais informados para requisição de faltas por um ID o sistema informar as faltas que constam no aluno desejado,.

TS010 Requisição de disciplina com autenticação.

Dado um acesso com credenciais informados para requisição de disciplina cadastrado o sistema informar as que existe um usuário logado.

TS011 Requisição de disciplina por ID com autenticação.

Dado um acesso com credenciais informados para requisição de disciplina por um ID o sistema informar que existe um usuário logado.

3.8 Teste de estresse

Ele se baseia em testar os limites do software e avaliar seu comportamento. Assim, avalia-se até quando o software pode ser exigido e quais as falhas (se existirem) decorrentes do teste.

TE001 Carga de requisições simultâneas.

Dado o sistema no ar deve suportar 100.000. mil requisições simultaneamente o sistema deve responder a todas as requisições normalmente com sucesso.

3.9 Teste de cobertura

É baseada em fluxo de controle, o objetivo é testar linhas de código, condições de ramificação, caminhos que percorrem o código ou outros elementos do fluxo de controle do software. Na cobertura baseada em fluxo de dados, o objetivo é testar se os estados dos dados permanecem válidos durante a operação do software,

4.0 - Projeto de Casos de Teste

São todos os elementos essenciais para o sucesso das atividades de teste em um projeto de software.

4.1 - Teste de integridade no banco de dados

ID	Cenário	Etapas	Resultado esperado
CT001	/admin	informa-se os dados cadastrais de um aluno e realiza o cadastro	Instância da classe Aluno
CT002	/admin	insere-se um nome de um aluno	Instância QuerySet
CT003	/admin	insere-se dados cadastrais da	Instância da classe Disciplina

		disciplina	
CT004	/admin	insere-se dados cadastrais do professor	Instância da classe Professor
CT005	/admin	insere-se os dados cadastrais de uma falta	Instância da classe Falta


4.2 - Testes de Segurança

ID	Cenário	Etapas	Verbo HTTP	Resultado esperado
TS001	/api/alunos	-	GET	403 FORBIDDEN
TS002	/api/alunos/1	-	GET	403 FORBIDDEN
TS003	/api/faltas	-	GET	403 FORBIDDEN
TS004	/api/faltas/1	-	GET	403 FORBIDDEN
TS005	/api/disciplinas	-	GET	403 FORBIDDEN
TS006	/api/disciplinas/1	-	GET	403 FORBIDDEN
TS007	/api/alunos	Credenciais fornecidas	GET	200 OK
TS008	/api/alunos/1	Credenciais fornecidas	GET	200 OK
TS009	/api/faltas	Credenciais fornecidas	GET	200 OK
TS010	/api/faltas/1	Credenciais fornecidas	GET	200 OK
TS011	/api/disciplinas	Credenciais fornecidas	GET	200 OK
TS012	/api/disciplinas/1	Credenciais fornecidas	GET	200 OK


Teste de Integração

ID	Cenário	Etapas	Verbo HTTP	Resultado esperado
TI001	/api/alunos	Informa-se os dados do aluno em formato JSON	POST	200 CREATED
TI002	/api/disciplinas	Informa-se os dados da disciplina em formato JSON	POST	200 CREATED
TI003	/api/faltas	informa-se os dados da falta em formato JSON	POST	200 CREATED

5. Execução

 master
GitHub

CRON Merge pull request #5 from daviwesley/d

 #148 passed
2F22285


 1 min 22 sec
 24 days ago

Figura 5. Resultado de um teste no Travis CI

```
497 Applying authtoken.0001_initial... OK
498 Applying authtoken.0002_auto_20160226_1747... OK
499 Applying sessions.0001_initial... OK
500 System check identified no issues (0 silenced).
501 test_api_create_a_fault (test_integration.TestAPIPost) ... ok
502 test_api_create_attendance (test_integration.TestAPIPost) ... ok
503 test_api_create_student (test_integration.TestAPIPost) ... ok
504 test_api_create_subject (test_integration.TestAPIPost) ... ok
505 test_api_create_teacher (test_integration.TestAPIPost) ... ok
506 test_api_create_turma_with_existing_teacher_and_absent_turma_name (test_integration.TestAPIPost)
507 it should create a turma because we have an existing teacher in DB ... ok
508 test_api_create_turma_without_absent_teacher_and_student_name (test_integration.TestAPIPost) ... ok
509 test_api_update_student (test_integration.TestAPIPost) ... ok
510 test_api_update_student_fault (test_integration.TestAPIPost) ... ok
511 test_api_update_subject (test_integration.TestAPIPost) ... ok
512 test_create_attendance_log (tests.TestDatabase) ... ok
513 test_create_fault (tests.TestDatabase) ... ok
514 test_create_subject (tests.TestDatabase) ... ok
515 test_create_teacher (tests.TestDatabase) ... ok
516 test_create_user (tests.TestDatabase) ... ok
517 test_student_creation (tests.TestDatabase) ... ok
518 test_student_name (tests.TestDatabase) ... ok
519 test_tem_usuario_joão (tests.TestDatabase) ... ok
520 test_url_api_attendance_get (tests.TestURLsemToken) ... ok
521 test_url_api_faults_get (tests.TestURLsemToken) ... ok
522 test_url_api_faults_post (tests.TestURLsemToken) ... ok
523 test_url_api_student_post (tests.TestURLsemToken) ... ok
524 test_url_api_students_get (tests.TestURLsemToken) ... ok
525 test_url_api_students_id_get (tests.TestURLsemToken) ... ok
526 test_url_api_subject_post (tests.TestURLsemToken) ... ok
527 test_url_api_teacher_post (tests.TestURLsemToken) ... ok
528 test_url_attendance_post (tests.TestURLsemToken) ... ok
529 test_received_token (tests.TestaURLcomToken) ... ok
530 test_url_api_faults_with_token (tests.TestaURLcomToken) ... ok
531 test_url_api_students_id_with_token (tests.TestaURLcomToken) ... ok
532 test_url_api_students_with_token (tests.TestaURLcomToken) ... ok
533 test_url_professores_deve_retorna_200 (tests.TestaURLcomToken) ... ok
534
535 -----
536 Ran 32 tests in 2.081s
537
538 OK
539 Destroying test database for alias 'default' ('file:memorydb_default?mode=memory&cache=shared')...
540 The command "python manage.py test api/test -v 2" exited with 0.
541
542
543
544 Done. Your build exited with 0.
```

Figura 6 - Resultado de todos os testes rodando no Travis CI

Os testes foram executados em um servidor dedicado de integração contínua e entrega contínua chamado Travis CI que é disponibilizado gratuitamente para projetos open source no github, oferece configurações de ambientes para vários sistema operacionais e linguagens de programação, nos teste realizados foram configurados para rodar em uma máquina Linux com o python na versão 3.6, a realização dos testes foram configuradas para rodarem a cada *pull request* github, *commit* realizado e teste agendado para rodar diariamente uma vez por dia, ao desenrolar do desenvolvimento foram corrigidos os erros que apareceram, os teste

realizados retornaram os mesmo resultados dos teste rodados localmente no ambiente de desenvolvimento.

6. Análise dos Resultados e Próximos Passos

Observou-se a praticidade de usar um servidor de integração contínua e entrega contínua para os testes propostos, agilizando a verificação de testes após mudanças do projeto pela equipe de desenvolvimento, notou-se a necessidade de seguir o padrão TDD no projeto motivado pela longa demora de se criar os testes depois de funcionalidades implementadas, acarretando em erros que poderiam ser corrigidos durante a implementação.e não depois da implementação, devido o projeto ser um trabalho de uma disciplina do curso de Engenharia de Software, a equipe de desenvolvimento concordou em deixar as futuras modificações para o desenvolvedor principal, uma vez que a equipe não precisa continuar o desenvolvimento do projeto pela finalização e conclusão da disciplina.