FUNDAÇÃO ESCOLA DE COMÉRCIO ÁLVARES PENTEADO

FECAP

CENTRO UNIVERSITÁRIO ÁLVARES PENTEADO

TESTES DE QUALIDADE DE SOFTWARE
APLICATIVO BEM VIVER CONNECT

LUCA SILVESTRE

MELISSA LEQUIPE

NAYAN PINHO DE OLIVEIRA

NICOLLE MARIA FIRMINIO

São Paulo

2024

LUCA SILVESTRE

MELISSA LEQUIPE

NAYAN PINHO DE OLIVEIRA

NICOLLE MARIA FIRMINIO

TESTES DE QUALIDADE DE SOFTWARE
APLICATIVO BEM VIVER CONNECT

Trabalho do curso de Análise de Desenvolvimento de Sistemas, da disciplina Teste de Qualidade de Software do Aplicativo Bem Viver Connect, apresentado à Fundação Escola de Comércio Álvares Penteado - FECAP,

Orientador: Prof. Aimar Martins Lopes

São Paulo

2024

Resumo

Este trabalho irá apresentar o aplicativo Bem Viver Connect, com apresentação de testes unitários, componentes. Além da mais demonstração dos atributos de qualidade de software utilizados no projeto do aplicativo Bem Viver Connect.

Palavras-chave: Testes, Qualidade, Aplicativo; Bem Viver Connect, Tarefas, Software.

Sumário

1 INTRODUÇÃO:

O aplicativo Bem Viver Connect, tem como objetivo o promover organização, agilidade na rotina dos seus usuários. Ele terá um checklist onde os usuários terão a oportunidade de inserir suas rotinas diárias, simplificando a gestão do tempo para hábitos saudáveis. Além do mais, o aplicativo visa aprimorar a qualidade de vida dos usuários.

O publico alvo do aplicativo são jovens interessado em gerenciar suas rotinas, em busca de uma vida mais organizada e saudável. O aplicativo tem o foco de proporcionar uma experiência acessível e personalizável para essa faixa etária, facilitando a incorporação de práticas saudáveis no seu cotidiano.

O aplicativo Bem Viver Connect é dedicado à transformação positiva de hábitos e qualidade de vida. Em resposta à crescente necessidade de uma abordagem abrangente, o aplicativo propõe uma solução integrada para a gestão diária, combinando funcionalidades como lembretes, checklists. Essa aplicativa busca não apenas preencher lacunas observadas na promoção da saúde, mas também oferecer uma experiência personalizável, incentivando os usuários a adotarem práticas saudáveis de forma eficaz.

2. TESTE DE SOFTAWARE

Os sistemas de softwares estão sujeitos a erros ou inconsistências no seu processo, pois isso é importante a realização de testes no desenvolvimento de um software. Os testes eram realizados deste os modelos de cascata, com o levantamento de requisitos, análise e codificação, naquele momento eram realizados testes manuais, com o objetivo de edificar bugs no sistema antes da entrada em produção.

No decorrer do tempo, os testes foram se aperfeiçoando e com a utilização de métodos ágeis, começaram a utilizar código para testar classes, se tornando auto testáveis, os testes ganharam novas funções, como verificar se uma classe continuará funcionando após um bug ser corrigido em uma outra parte do desenvolvimento do software. Os testes são divididos em três grupos, testes de unidade, teste de integração e testes de serviços.

2.1. Apresentação 2 Testes Unitários

Com a criação de uma classe checklistItem, foi realizado dois testes de unidade

1° Teste:

```
import org.junit.Test;
import static org.junit.Assert.*;

public class ChecklistItemTest {
  @Test
        public void testChecklistItem() {
            // Criar um item de checklist
            ChecklistItem item = new ChecklistItem("Item de teste");

            // Verificar se a descrição está correta
            assertEquals("Item de teste", item.getDescription());

            // Verificar se o item não está marcado inicialmente
            assertFalse(item.isChecked());
        }

        @Test
        public void testCheck() {
            // Criar um item de checklist
            ChecklistItem item = new ChecklistItem("Item de teste");

            // Marcar o item
            item.check();

            // Verificar se o item está marcado
            assertTrue(item.isChecked());
        }
```

```
    @Test
    public void testUncheck() {
        // Criar um item de checklist
        ChecklistItem item = new ChecklistItem("Item de teste");

        // Marcar o item e depois desmarcá-lo
        item.check();
        item.uncheck();

        // Verificar se o item não está marcado
        assertFalse(item.isChecked());
    }
}
```

2° Teste:

```
import org.junit.Test;
import static org.junit.Assert.*;

public class ChecklistItemTest {

    @Test
    public void testChecklistItem() {
        // Criar um item de checklist
        ChecklistItem item = new ChecklistItem("Item de teste");

        // Verificar se a descrição está correta
        assertEquals("Item de teste", item.getDescription());

        // Verificar se o item não está marcado inicialmente
        assertFalse(item.isChecked());
    }
```

```
@Test
public void testCheck() {
    // Criar um item de checklist
    ChecklistItem item = new ChecklistItem("Item de teste");

    // Marcar o item
    item.check();

    // Verificar se o item está marcado
    assertTrue(item.isChecked());
}

@Test
public void testUncheck() {
    // Criar um item de checklist
    ChecklistItem item = new ChecklistItem("Item de teste");

    // Marcar o item e depois desmarcá-lo
    item.check();
    item.uncheck();

    // Verificar se o item não está marcado
    assertFalse(item.isChecked());
}

@Test
public void testChecklistItemDescriptionNotNull() {
    // Criar um item de checklist com descrição nula
    ChecklistItem item = new ChecklistItem(null);

    // Verificar se a descrição é nula
```

```
        assertNull(item.getDescription());
    }

    @Test
    public void testChecklistItemWithEmptyDescription() {
        // Criar um item de checklist com descrição vazia
        ChecklistItem item = new ChecklistItem("");

        // Verificar se a descrição está vazia
        assertEquals("", item.getDescription());
    }
}
```

2.2 Apresentação 2 Testes Componentes

```
//Buscar todos os usuários
app.get("/buscarUsuarios", function (req, res) {
  db.all(`SELECT * FROM usuarios`, [], (err, rows) => {
    if (err) {
      res.send(err);
    }
    res.send(rows);
  });
});

//Buscar todas as tarefas
app.get("/buscarTarefas", function (req, res) {
  db.all(`SELECT * FROM tarefas`, [], (err, rows) => {
    if (err) {
      res.send(err);
    }
    res.send(rows);
  });
});

// Criação da tabela 'usuarios', se ainda não existir
db.serialize(function () {
  db.run(
    `CREATE TABLE IF NOT EXISTS usuarios (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      nome TEXT,
      email TEXT UNIQUE,
      senha TEXT,
      humor INTEGER
    )`,
    function (err) {
      if (err) {
        console.error("Erro ao criar tabela:", err.message);
      }
    },
  );
});
```

```
// Criação da tabela 'tarefas', se ainda não existir
db.serialize(function () {
  db.run(
    `CREATE TABLE IF NOT EXISTS tarefas (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        id_usuario INTEGER,
        nome_tarefa TEXT NOT NULL,
        data_hora DATETIME,
        status INTEGER,
        FOREIGN KEY (id_usuario) REFERENCES usuarios(id)
      )`,
    function (err) {
      if (err) {
        console.error("Erro ao criar tabela:", err.message);
      }
    },
  );
});

app.get("/", function (req, res) {
  res.send("Servidor funcionando!");
});
```

```
/ Endpoint de login
app.post("/login", (req, res) => {
  const { email, senha } = req.body;
  db.get(
    "SELECT * FROM usuarios WHERE email = ? AND senha = ?",
    [email, senha],
    (err, row) => {
      if (err) {
        res.status(500).json({ error: "Erro ao autenticar usuário" });
      } else {
        if (row) {
          // Usuário encontrado, enviar resposta de sucesso com dados do usuário
          res.status(200).json({
            message: "Login bem-sucedido",
            user: {
              id: row.id,
```

```
            nome: row.nome,
            email: row.email,
            humor: row.humor,
          },
        });
      } else {
        // Usuário não encontrado ou senha incorreta
        res.status(401).json({ message: "Usuário ou senha incorretos" });
      }
    }
  },
);
});
```

```
// Endpoint de login
app.post("/login", (req, res) => {
  const { email, senha } = req.body;
  db.get(
    "SELECT * FROM usuarios WHERE email = ? AND senha = ?",
    [email, senha],
    (err, row) => {
      if (err) {
        res.status(500).json({ error: "Erro ao autenticar usuário" });
      } else {
        if (row) {
          // Usuário encontrado, enviar resposta de sucesso com dados do usuário
          res.status(200).json({
            message: "Login bem-sucedido",
            user: {
              id: row.id,
              nome: row.nome,
              email: row.email,
              humor: row.humor,
            },
          });
        } else {
          // Usuário não encontrado ou senha incorreta
          res.status(401).json({ message: "Usuário ou senha incorretos" });
        }  }
```

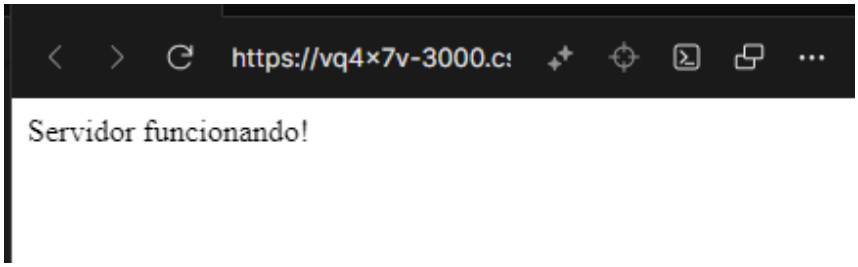
```
    },
  );
});

//Deletar usuário
app.delete("/deletarUsuario", function (req, res) {
  var email = req.body.email;
  var senha = req.body.senha;

  // Verificar se o email e a senha foram fornecidos
  if (!email || !senha) {
    return res
      .status(400)
      .send("Email e senha são obrigatórios para deletar a conta.");
  }
  db.get(
    "SELECT * FROM usuarios WHERE email = ? AND senha = ?",
    [email, senha],
    function (err, row) {
      if (err) {
        return res.status(500).send("Erro ao consultar o banco de dados.");
      } else {
        if (!row) {
          // Usuário não encontrado ou credenciais incorretas
          return res.status(401).send("Email ou senha incorretos.");
        } else {
          // Deletar o usuário do banco de dados
          var sql = "DELETE FROM usuarios WHERE email = ?";
          db.run(sql, [email], function (err) {
            if (err) {
              return res.status(500).send("Erro ao deletar a conta.");
            } else {
              return res.status(200).send("Conta deletada com sucesso.");
            }
          });
        }
      }
    },
  );
});
```

2.3 Apresentação Teste De Sistema

Print da tela do Servidor Funcionando



Tela de cadastro, com solicitação do nome, e-mail e senha

Tela Login com a solicitação do e-mail e senha

Print da tela Principal, onde o usuário consegue colocar como ele está naquele dia com os emoji e adicionar a tarefa no aplicativo Bem Viver Connect e verificar as tarefas cadastradas no aplicativo.

3. QUALIDADE DE SOFWARE

3.1 Indicar 4 Atributos De Qualidade De Software E Informar Como Foi Aplicado No Projeto Integrador (Pi)

1. Funcionalidade

Aplicação no Projeto:

- Requisitos Claros: Documentamos todas as funcionalidades que o aplicativo deveria ter, como adicionar, editar, e marcar tarefas como concluídas.
- Validação de Dados: Fizemos verificações para garantir que os dados inseridos pelos usuários seriam válidos.
- Testes Funcionais: Realizamos testes unitários e de integração para verificar se todas as funcionalidades estão funcionando conforme o esperado. Por exemplo, as rotas do servidor com a utilização do banco de dados.

2. Usabilidade

Aplicação no Projeto:

- Design Intuitivo: A interface é clara e intuitiva, com botões bem posicionados e descrições.
- Feedback do Usuário: Mensagens de feedback nas ações do usuário, confirmações ao adicionar uma tarefa e alertas de uma tarefa.

3. Confiabilidade

Aplicação no Projeto:

- Rastreio de erros: Tentamos mapear e evitar os possíveis erros que o usuário poderia enfrentar, como login ou senha errada.

4. Manutenibilidade

Aplicação no Projeto:

- Código Limpo e Comentado: Mantemos o código limpo, bem estruturado e comentado para facilitar as modificações e raciocínio nas outras versões.
- Controle de Versão: Usamos o GitHub, para gerenciar mudanças no código e colaborar eficientemente com todos da equipe.

3.2 Apresentar Um Modelo De Qualidade De Software

Modelo de Qualidade do Aplicativo Bem Viver Connect			
Características	**Sub Características**	**Aplicável**	**Justificativa**
Adequação Funcional	**Correção funcional**	**X**	O aplicativo tem o objetivo de ser uma ferramenta eficaz e personalizável com lembretes, checklist em um único ambiente digital, para melhoria da saúde e bem-estar do usuário.
Eficiência de Desempenho	**Utilização de recursos**	**X**	No código do aplicativo, está sendo desenvolvido a programação com telas de usuário e tela de listas para inserção e edição das tarefas.
Capacidade de Interação	**Operabilidade**	**X**	O aplicativo será intuitivo para o usuário, para ele ter livre acesso para inclusão de itens sobre suas atividades diárias.
	Proteção contra erros do usuário	**X**	O Aplicativo no seu código, irá ter um comando que quando tiver algum bug no sistema referente ao usuário, irá acionar o erro para o desenvolvedor do aplicativo. Caso não seja possível resolver o problema, será necessário entrar em contato pessoalmente com o usuário
Confiabilidade	**Disponibilidade**	**X**	O aplicativo tem como objetivo estar disponível em todos os horários para os usuários, justificando que o usuário final sejam jovens que tem como objetivo a gerenciar suas tarefas diárias de forma organizada.
Segurança	**Confidencialidade**	**X**	O aplicativo irá seguir a LGPD com proteção de dados do usuário.
Capacidade de Manutenção	**Testabilidade**	**X**	O aplicativo no seu desenvolvimento irá passar por processos de testes de códigos e usabilidade e melhorando de acordo com feedbacks dos usuários.

3.3 Apresentar um Processo (plano) de gerenciamento de qualidade de software

1. Objetivos de Qualidade

- Funcionalidade Básica: Garantir que o App tenha as funcionalidades essenciais: adicionar, editar e marcar tarefas como concluídas.
- Usabilidade: Possuir uma interface simples e intuitiva.
- Confiabilidade: Assegurar que o aplicativo funcione com o mínimo de erros possíveis.
- Manutenibilidade Básica: Facilitar ajustes e correções rápidas durante o desenvolvimento.

2. Processo de Qualidade

Planejamento

- Definição de Requisitos: Documentar as funcionalidades essenciais do protótipo.
- Plano de Testes Básico: Definir testes básicos para garantir que as funcionalidades principais estejam operando corretamente.

Desenvolvimento

- Codificação: Desenvolver o aplicativo com simplicidade e clareza do código.
- Revisão de Código: Revisar o código em grupo para identificar possíveis problemas e melhorias.

Testes

- Testes Funcionais Básicos: Verificar manualmente se as funcionalidades de adicionar, editar, e marcar tarefas estão funcionando.

- Testes de Usabilidade Informais: Pedir a colegas para usar o aplicativo e fornecer feedback sobre a interface e a experiência do usuário.

Implantação

- Demonstração: Testamos o aplicativo no celular durante a aula.
- Feedback do Professor: Tivemos um feedback durante o teste em aula para melhorias.

3. Ferramentas de Qualidade

- Controle de Versão: Utilizamos GitHub para versionamento do código, mesmo que de forma básica.
- Ambiente de Desenvolvimento: Utilizamos Android Studio para desenvolver e testar o aplicativo.

4. Equipe e Responsabilidades

- Desenvolvedores: Todos os membros do grupo são responsáveis pela codificação, revisão de código e testes.

5. Cronograma

- Planner: Utilizamos o uso do Trello para fazer um cronograma e organizar as tarefas para entrega.

REFERÊNCIAS BIBLIOGRÁFICAS

SOMMERVILLE, I. **Engenharia de Software.** 11ª Edição. São Paulo: Pearson Addison-Wesley, 2017.