FECAP

PROJETO Treinar.AI

Requisitos da disciplina Modelagem de Software e Arquitetura de Sistemas

São Paulo

2024

INTEGRANTES DO PROJETO e RA’S

Nicolas Bueno Zagatto	-	20010562
Carlos Augusto Santos de Almeida	-	20010535
Lucas Silva Cardiais	-	20010551
Victor Barcelos Araújo	-	23024553

Contents

1. INTRODUÇÃO

O aplicativo Treinar AI é um projeto educacional, que simplifica a criação e organização de treinos para os alunos ingressantes das academias, dessa forma poupando tempo tanto dos alunos, quanto dos profissionais. Com uma interface intuitiva, os usuários podem criar treinos personalizados, adaptados às suas necessidades e objetivos. Uma biblioteca abrangente de exercícios está disponível para orientar a execução correta. Este aplicativo é destinado a qualquer pessoa interessada em frequentar uma academia, é mais indicado para pessoas iniciantes que estão buscando orientação e um treino base, mas também pode ser usado por pessoas experientes ou entusiastas de fitness que procuram uma ferramenta prática e eficiente. Nosso objetivo é facilitar a criação de treinos e educar sobre a importância do planejamento adequado e da execução correta dos exercícios, promovendo um estilo de vida saudável e ativo.

2. Teste de Software

2.1. Apresentar 2 testes unitários.

Teste 1: Validação de E-mail Válido

Objetivo: Verificar se o método `isValidEmail` retorna `true` para um e-mail válido.

Código do Teste:

```
import static org.junit.Assert.assertTrue;
import org.junit.Test;
public class MainActivityTest {
    @Test
    public void testValidEmail() {
        MainActivity mainActivity = new MainActivity();
        String emailValido = "usuario@exemplo.com";
        boolean resultado = mainActivity.isValidEmail(emailValido);
        assertTrue("O email deveria ser considerado válido", resultado);
    }
```

```
}
```

Este teste cria uma instância da MainActivity e utiliza o método isValidEmail para verificar se um e-mail válido (usuario@exemplo.com) é corretamente identificado como válido. O teste espera que o método retorne true.

Teste 2: Validação de Senha Válida

Objetivo: Verificar se o método isValidPassword retorna true para uma senha que atenda aos critérios de segurança.

Código do Teste:

```
import static org.junit.Assert.assertTrue;

import org.junit.Test;

public class MainActivityTest {

    @Test

    public void testValidPassword() {

        MainActivity mainActivity = new MainActivity();

        String senhaValida = "Senha123";

        boolean resultado = mainActivity.isValidPassword(senhaValida);

        assertTrue("A senha deveria ser considerada válida", resultado);

    }

}
```

Este teste cria uma instância da MainActivity e utiliza o método isValidPassword para verificar se uma senha válida (Senha123) é corretamente identificada como válida. O teste espera que o método retorne true.

Para executar os testes, seguimos o seguinte passo a passo:

- Adicionamos a dependência do JUnit ao arquivo build.gradle:

  testImplementation 'junit:junit:4.13.2'

- Criamos um novo arquivo de teste MainActivityTest.java dentro do diretório src/test/java/com/example/treinarai/.

- Colamos os códigos dos testes no arquivo MainActivityTest.java.

• Executamos os testes utilizando a configuração de teste do Android Studio ou através do terminal com o comando: ./gradlew test.

Os testes unitários descritos garantem que os métodos de validação de e-mail e senha da MainActivity estão funcionando corretamente. Esses testes ajudam a identificar problemas de validação que podem impedir que usuários legítimos acessem o sistema ou permitam o acesso de usuários não autorizados.

2.2. Apresentar 2 testes de componentes

Teste 1: Interação com Campos de Texto e Botão de Login

Objetivo: Verificar se os campos de texto e o botão de login interagem corretamente e iniciam o processo de validação.

Código do Teste:

```
import androidx.test.ext.junit.runners.AndroidJUnit4;

import androidx.test.rule.ActivityTestRule;

import org.junit.Rule;

import org.junit.Test;

import org.junit.runner.RunWith;

import static androidx.test.espresso.Espresso.onView;

import static androidx.test.espresso.action.ViewActions.click;

import static androidx.test.espresso.action.ViewActions.replaceText;

import static androidx.test.espresso.assertion.ViewAssertions.matches;

import static androidx.test.espresso.matcher.ViewMatchers.withId;

import static androidx.test.espresso.matcher.ViewMatchers.withText;

@RunWith(AndroidJUnit4.class)

public class MainActivityComponentTest {

    @Rule

    public ActivityTestRule<MainActivity> activityRule = new
ActivityTestRule<>(MainActivity.class);
```

```
    @Test

    public void testLoginButton_withValidInput() {
onView(withId(R.id.campoEmail)).perform(replaceText("usuario@exemplo.com"
));

        onView(withId(R.id.campoSenha)).perform(replaceText("Senha123"));

        onView(withId(R.id.botaoEntrar)).perform(click());
onView(withText("Carregando...")).check(matches(withText("Carregando...")));

    }

}
```

Este teste utiliza a biblioteca Espresso para automatizar a interação com a interface do usuário. O teste insere um e-mail e uma senha válidos nos campos de texto e clica no botão de login. Em seguida, verifica se o diálogo de carregamento é exibido, confirmando que o processo de validação foi iniciado.

Teste 2: Mensagem de Erro para E-mail Inválido

Objetivo: Verificar se uma mensagem de erro apropriada é exibida quando o usuário insere um e-mail inválido.

Código do Teste:

```
import androidx.test.ext.junit.runners.AndroidJUnit4;

import androidx.test.rule.ActivityTestRule;

import org.junit.Rule;

import org.junit.Test;

import org.junit.runner.RunWith;

import static androidx.test.espresso.Espresso.onView;

import static androidx.test.espresso.action.ViewActions.click;

import static androidx.test.espresso.action.ViewActions.replaceText;

import static androidx.test.espresso.assertion.ViewAssertions.matches;

import static androidx.test.espresso.matcher.ViewMatchers.withId;

import static androidx.test.espresso.matcher.ViewMatchers.withText;
```

```
@RunWith(AndroidJUnit4.class)

public class MainActivityComponentTest {

    @Rule

    public ActivityTestRule<MainActivity> activityRule = new
ActivityTestRule<>(MainActivity.class);

    @Test

    public void testLoginButton_withInvalidEmail() {

        onView(withId(R.id.campoEmail)).perform(replaceText("usuarioinvalido"));

        onView(withId(R.id.campoSenha)).perform(replaceText("Senha123"));

        onView(withId(R.id.botaoEntrar)).perform(click());

        onView(withText("Por favor, insira um email
válido")).check(matches(withText("Por favor, insira um email válido")));

    }

}
```

Este teste também utiliza a biblioteca Espresso. O teste insere um e-mail inválido e uma senha válida nos campos de texto e clica no botão de login. Em seguida, verifica se a mensagem de erro "Por favor, insira um e-mail válido" é exibida, confirmando que a validação de e-mail está funcionando corretamente.

Para executar os testes de componentes, seguimos o seguinte passo a passo:

- Adicionamos a dependência do Espresso ao arquivo build.gradle:

  androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

  androidTestImplementation 'androidx.test.ext:junit:1.1.3'

- Criamos um novo arquivo de teste MainActivityComponentTest.java dentro do diretório src/androidTest/java/com/example/treinarai/.

- Colamos os códigos dos testes no arquivo MainActivityComponentTest.java.

- Executamos os testes utilizando a configuração de teste de instrumentação do Android Studio.

Os testes de componentes descritos garantem que a interface de usuário da classe MainActivity interage corretamente com os usuários. Esses testes ajudam a identificar problemas de usabilidade e garantem que o fluxo de autenticação está funcionando conforme o esperado.

2.3. Apresentar um teste de sistema.

O teste de sistema foi feito em formato de vídeo e acompanha a documentação no GitHub com o nome:

testeQualdevops_testeSistema

3. Qualidade de Software

3.1. Indicar 4 atributos de qualidade de software e informar como foi aplicado no projeto integrador (PI)

1. Funcionalidade

- Precisão das Recomendações**:** Fornecimento de templates de treino personalizados conforme o biotipo do aluno, garantido por algoritmos precisos.
- Gestão de Perfil**:** Funcionalidades robustas para edição de perfil e ajuste de preferências, validadas por testes de unidade e integração.

2. Usabilidade

- Interface Intuitiva**:** Navegação clara e fácil entre as telas (login, cadastro, perfil, Typebot), melhorada através de testes de usabilidade.
- Tutoriais e Dicas**:** Inclusão de tutoriais e dicas para facilitar a compreensão e uso do aplicativo pelos novos usuários.

3. Confiabilidade

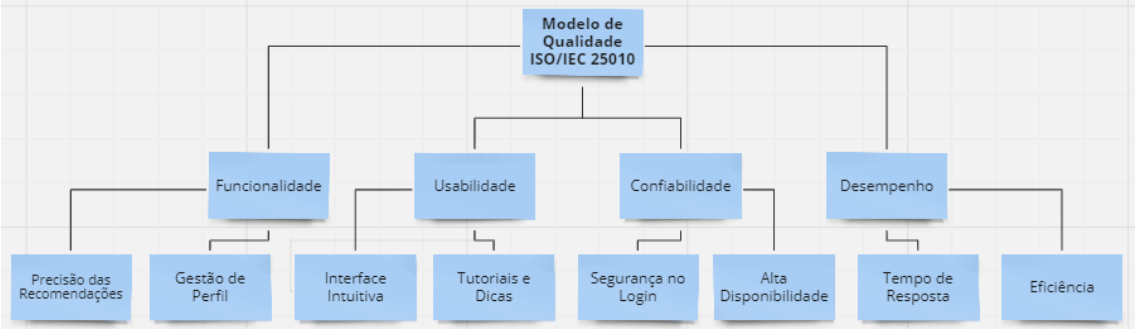
- Segurança no Login: Implementação de protocolos seguros para autenticação e proteção dos dados do usuário.
- Alta Disponibilidade**:** Monitoramento contínuo e backups regulares para assegurar que o aplicativo esteja sempre disponível.

4. Desempenho

- Tempo de Resposta**:** Otimização do tempo de resposta do aplicativo para garantir que as telas carreguem rapidamente e as recomendações sejam geradas sem demora.

- Eficiência: Uso eficiente de recursos do dispositivo (memória, CPU) para garantir que o aplicativo funcione bem em diferentes tipos de hardware, minimizando travamentos e lentidão.

3.2. Apresentar um Modelo de qualidade de software



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

1. Definição de Objetivos de Qualidade
   - Identificar as principais metas de qualidade para o aplicativo, como usabilidade, confiabilidade e desempenho.
   - Estabelecer critérios de aceitação para cada objetivo de qualidade.
2. Planejamento de Qualidade
   - Designar responsabilidades claras para a equipe de desenvolvimento em relação ao controle de qualidade.
   - Estabelecer um plano de testes que cubra todas as funcionalidades principais do aplicativo.
3. Desenvolvimento com Foco na Qualidade
   - Implementar boas práticas de desenvolvimento de software, como design modular e padrões de codificação consistentes.
   - Realizar revisões de código regulares para identificar e corrigir problemas de qualidade.
4. Testes de Qualidade
   - Desenvolver casos de teste abrangentes que cubram todos os requisitos funcionais e não funcionais do aplicativo.
   - Realizar testes de unidade, integração, sistema e aceitação para garantir a qualidade em todos os níveis.

5. Gestão de Defeitos

- Utilizar ferramentas de rastreamento de problemas para registrar e priorizar defeitos encontrados durante os testes.
- Garantir que os defeitos sejam corrigidos e retestados antes da liberação do aplicativo.

6. Melhoria Contínua

- Realizar retrospectivas regulares para avaliar o processo de gerenciamento de qualidade e identificar áreas de melhoria.
- Implementar ações corretivas e preventivas para melhorar continuamente a qualidade do software.

Exemplo de Cronograma de Qualidade

Atividade	Tempo Estimado
Planejamento de Qualidade	1 semana
Desenvolvimento e Revisão	4 semanas
Testes de Qualidade	2 semanas
Correção de Defeitos	1 semana
Melhoria Contínua	Ongoing

4. REFERÊNCIAS BIBLIOGRÁFICAS

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

Slides da disciplina de Testes e Qualidade de Software (DevOps).