1 - Informe o nome da equipe e o nome dos 4 alunos que farão todos os TDE's em grupo:

Nome da Equipe: Code Solution.
Alunos: Camila Specalski, Hemerson Lacovic.

2 - Você deverá pensar em uma ideia de software simples para ser desenvolvido. Descreva aqui a ideia do software e pelo menos 5 requisitos.

O software será uma calculadora básica que permitirá aos usuários realizar operações matemáticas simples como: soma, subtração, multiplicação, divisão e cálculo de porcentagem.

Requisitos do Software:

Interface de Usuário Simples:
- Deve ter botões para cada operação (soma, subtração, multiplicação, divisão, porcentagem).
- Deve ter um campo de entrada para o usuário inserir os números.
- Deve ter um display para mostrar os resultados das operações.

Operações Básicas:
- Deve permitir a realização de operações de soma, subtração, multiplicação, divisão e porcentagem.

Validação de Entrada:
- Deve validar as entradas do usuário para garantir que são números válidos.
- Deve tratar erros como divisão por zero e entradas inválidas de forma amigável.

Memória de Cálculo:
- Deve ter uma funcionalidade para armazenar o último resultado calculado, permitindo seu uso em operações futuras.

Compatibilidade e Responsividade:
- Deve ser compatível com diferentes dispositivos e tamanhos de tela, garantindo uma boa experiência de uso em desktops, tablets e smartphones.
- A interface deve ser responsiva, ajustando-se ao tamanho da tela do dispositivo.

3 - Como você planejaria a construção de um software?

1. Planejamento

a. Definição do Escopo
- Objetivo do Software: Definir claramente o propósito e as funcionalidades principais.
- Requisitos Funcionais e Não Funcionais: Especificar o que o software deve fazer (funcionais) e as restrições sob as quais ele deve operar (não funcionais).

b. Cronograma e Recursos

- Timeline: Estabelecer prazos para cada fase do projeto.
- Recursos Humanos e Materiais: Identificar a equipe necessária e as ferramentas/software que serão usados.

2. Análise de Requisitos

a. Coleta de Requisitos
- Entrevistas e Questionários: Coletar informações dos stakeholders.
- Documentação de Requisitos: Criar um documento detalhado que liste todos os requisitos do software.

b. Análise e Validação
- Revisão de Requisitos: Verificar se os requisitos são claros, completos e viáveis.
- Modelagem de Casos de Uso: Criar diagramas de casos de uso para ilustrar as interações do usuário com o sistema.

3. Projeto de Software

a. Arquitetura do Sistema
- Diagrama de Arquitetura: Definir a estrutura geral do software e as interações entre os componentes.

b. Design Detalhado
- Diagrama de Classes e Sequência: Desenhar diagramas detalhados para a implementação.
- Protótipos de Interface: Criar protótipos das telas e interfaces do usuário.

4. Desenvolvimento

a. Configuração do Ambiente
- Ferramentas de Desenvolvimento: Instalar e configurar IDEs, bibliotecas e frameworks necessários.
- Controle de Versão: Configurar um sistema de controle de versão como Git.

b. Codificação
- Implementação de Funcionalidades: Desenvolver o código seguindo as especificações de design.
- Revisões de Código: Realizar revisões de código regularmente para manter a qualidade.

5. Testes

a. Plano de Testes
- Estratégia de Testes: Definir a abordagem e tipos de testes (unitários, integração, sistema, aceitação).
- Casos de Teste: Especificar os cenários de teste e os dados de entrada/saída esperados.

b. Execução de Testes
- Automatização de Testes: Implementar testes automatizados quando possível.
- Registro de Defeitos: Documentar e priorizar os defeitos encontrados.

6. Implantação

a. Preparação para Implantação
- Documentação de Implantação: Criar documentação detalhada sobre o processo de implantação.
- Ambiente de Produção: Configurar e testar o ambiente onde o software será executado.

b. Implantação

- Lançamento do Software: Implantar o software no ambiente de produção.
- Monitoramento Pós-Implantação: Monitorar o sistema para identificar e corrigir qualquer problema inicial.

7. Manutenção

a. Suporte e Atualizações

- Correção de Bugs: Resolver problemas identificados pelos usuários.
- Melhorias e Atualizações: Adicionar novas funcionalidades ou melhorar as existentes com base no feedback.

8. Documentação

a. Documentação Técnica

- Manual do Desenvolvedor: Incluir detalhes sobre a arquitetura, design e código.
- Manual do Usuário: Fornecer instruções detalhadas para os usuários finais.

b. Relatórios de Teste

- Resultados dos Testes: Documentar os resultados dos testes e as correções realizadas.

9. Revisão e Avaliação

a. Avaliação do Projeto

- Análise de Desempenho: Avaliar o desempenho do software em termos de velocidade, estabilidade e escalabilidade.
- Feedback dos Stakeholders: Coletar feedback dos usuários e stakeholders para identificar áreas de melhoria.

4 - Que desafios você acha que seriam os mais difíceis?

Os desafios mais difíceis na construção de um software incluem garantir que os requisitos sejam claros e completos, controlar o escopo do projeto e cumprir prazos diante de mudanças frequentes e problemas inesperados. Além disso, integrar o software com sistemas legados e garantir sua performance e escalabilidade são tecnicamente desafiadores.

Manter a qualidade do software através de testes abrangentes e gerenciar defeitos de maneira eficiente são tarefas complexas. Proteger o software contra vulnerabilidades também é um desafio contínuo. A gestão eficaz da equipe, especialmente em times distribuídos, exige ferramentas de colaboração e comunicação eficientes.

Após a implantação, surgem problemas imprevistos que exigem monitoramento constante e correções rápidas. Manter o software atualizado com novas funcionalidades e correções baseadas no feedback dos usuários é um desafio constante. Superar esses obstáculos requer planejamento cuidadoso, metodologias adequadas, ferramentas apropriadas e uma equipe bem treinada e comunicativa.

5 - Que fase você considera como a mais complicada?

Consideramos a fase de coleta e definição de requisitos como a mais complicada na construção de um software. Essa etapa é fundamental para o sucesso do projeto, pois qualquer erro ou ambiguidade pode levar a problemas significativos nas fases subsequentes. Capturar todas as necessidades e expectativas dos stakeholders de forma clara e detalhada é um grande desafio. Requisitos ambíguos podem resultar em mal-entendidos, levando a um produto final que não atende às expectativas.

Além disso, durante o desenvolvimento, as necessidades dos stakeholders podem mudar, resultando em alterações de requisitos. Gerenciar essas mudanças sem comprometer o cronograma e o orçamento do projeto é extremamente difícil. Alinhar as expectativas e prioridades de diferentes stakeholders, que muitas vezes podem ser conflitantes, é uma tarefa complicada e demorada. A comunicação eficaz é essencial para garantir que todos compreendam os requisitos da mesma forma, o que inclui a comunicação entre analistas de negócios, desenvolvedores e stakeholders não técnicos.

6 - Que estratégias utilizaria para conseguir alcançar os Requisitos (especificações) e Expectativas do cliente?

Para alcançar os requisitos e expectativas do cliente na construção de um software, manteria uma comunicação contínua e aberta desde o início, utilizando entrevistas, workshops e sessões de brainstorming para entender suas necessidades.

Criaria protótipos e wireframes para obter feedback antecipado e utilizaria metodologias ágeis para permitir entregas incrementais e ajustes rápidos.

Realizaria revisões contínuas dos requisitos, usando modelagem de casos de uso e cenários de usuário para assegurar a precisão. Implementaria testes contínuos e usabilidade para identificar e resolver problemas rapidamente. Após a implementação, manteria um canal de suporte aberto para responder a problemas e ajustes, garantindo a satisfação do cliente a longo prazo.

7 - Por que os custos de desenvolvimento são difíceis de aceitar e prever?

Os custos de desenvolvimento de software são difíceis de aceitar e prever por vários fatores. Primeiro, cada projeto pode ser muito diferente em termos de complexidade e escopo, e mesmo pequenas mudanças no escopo podem ter um impacto significativo no custo total. Além disso, os requisitos muitas vezes não são totalmente definidos ou claros no início do projeto, e conforme o desenvolvimento avança, novas necessidades e ajustes podem surgir, o que dificulta a previsão precisa dos custos. Outro aspecto crítico é a escolha das tecnologias e ferramentas. Tecnologias novas ou menos familiares podem aumentar os custos devido à curva de aprendizado e à necessidade de treinamento adicional para a equipe de desenvolvimento. A experiência da equipe é outro fator importante. Times mais experientes tendem a ser mais eficientes, mas geralmente são mais caros. Além disso, imprevistos técnicos inesperados, como bugs difíceis de resolver ou integrações complicadas, podem surgir a qualquer momento, exigindo mais tempo e dinheiro para serem resolvidos. E muitas vezes, as estimativas iniciais de custos acabam subestimando a real complexidade do projeto porque faltam experiência ou

dados previstos. Todos esses fatores juntos fazem com que a previsão de tempo e custos de desenvolvimento de software seja sempre um desafio para empresas e desenvolvedores.

8 - Por que não podemos achar todos os erros antes da entrega?

Encontrar todos os erros antes de entregar um software é praticamente impossível por diversos motivos. A complexidade de um software é um grande desafio. Existem muitos caminhos de execução possíveis, e testar cada um deles de forma completa é inviável, especialmente considerando as limitações de tempo e recursos que se tem. Testar todas as combinações possíveis não é viável na prática. Além disso, o ambiente de uso dos usuários é bastante variável. Não é possível simular todas as condições e configurações que os usuários podem ter. O software pode se comportar de maneiras diferentes em diversos sistemas operacionais, versões de hardware e configurações específicas, as quais não são possíveis de replicar totalmente nos testes. As interações imprevisíveis também são um problema. Certos erros só aparecem em situações muito específicas de uso real. Mesmo testando bastante, sempre haverá condições únicas que não foram previstas. O fator humano também é importante. Tanto desenvolvedores quanto testadores podem deixar passar erros ou deixar de notar certos problemas. Por fim, as correções de bugs e as atualizações constantes podem introduzir novos erros. Ao corrigir um problema, pode-se acidentalmente criar outro. As mudanças contínuas no código para adicionar funcionalidades ou corrigir bugs acabam gerando novas falhas, mantendo o ciclo de identificação e correção de erros. Tudo isso junto torna impossível encontrar e eliminar todos os erros antes da entrega do software.

9 - Por que é tão alto o esforço de manutenção dos softwares já existentes?

O alto esforço de manutenção dos softwares já existentes se deve a diversos fatores. À medida que o software evolui, ele se torna mais complexo, o que dificulta a manutenção. Decisões de curto prazo durante o desenvolvimento podem gerar uma dívida técnica que complica futuras alterações. Manter sistemas antigos pode ser desafiador devido ao uso de tecnologias obsoletas. A falta de documentação adequada pode levar à perda de conhecimento crucial ao longo do tempo, dificultando a compreensão e modificação do software. As mudanças de requisitos exigem que o software seja adaptado a novas necessidades, o que pode ser complicado e demorado. Integrar o software com novas tecnologias e sistemas também pode ser problemático, especialmente quando se trata de compatibilidade. A rotatividade de equipe significa que novos desenvolvedores precisam entender sistemas legados, o que pode ser um processo lento e difícil. Por fim, corrigir bugs em sistemas complexos pode ser demorado e arriscado, aumentando o esforço de manutenção.

10 – Escolha um modelo de desenvolvimento de software que poderia auxiliar ou ser mais interessante para a parte de TESTES e justifique sua escolha.

Um modelo de desenvolvimento de software que pode ser muito interessante para a parte de testes é o Desenvolvimento Dirigido por Testes (TDD - Test-Driven Development). Esse modelo é

especialmente útil para destacar a importância dos testes durante todo o processo de desenvolvimento.

No TDD, o ciclo começa com a escrita de um teste antes mesmo de se começar a codificar a funcionalidade. Inicialmente, esse teste falha porque o código ainda não foi implementado. O próximo passo é escrever o código mínimo necessário para que o teste passe. Depois disso, o código é refatorado para melhorar sua qualidade, mas o teste continua aprovado.

A escolha do TDD para a parte de testes tem várias vantagens:

- Foco na Qualidade: Com o TDD, os testes são prioritários desde o início, o que garante que cada funcionalidade seja testável e realmente testada.
- Detecção Precoce de Defeitos: Escrever os testes antes do código ajuda a identificar problemas mais cedo no ciclo de desenvolvimento, o que facilita a correção rápida.
- Design Orientado a Testes: O TDD promove um design mais modular e com baixo acoplamento, tornando o código mais fácil de testar.
- Documentação: Os testes servem como uma forma de documentação executável, mostrando claramente como o sistema deve se comportar.
- Confiança nas Mudanças: Com uma suíte de testes bem construída, os desenvolvedores podem modificar o código com mais segurança, sabendo que os testes ajudarão a detectar problemas.
- Cobertura de Testes Aprimorada: O TDD naturalmente resulta em alta cobertura de testes, já que nenhum código é escrito sem um teste correspondente.
- Feedback Rápido: Os desenvolvedores recebem feedback imediato sobre o impacto das suas mudanças, o que permite ajustes rápidos e eficazes.

Embora o TDD possa parecer mais lento no início e exigir uma mudança de mentalidade na equipe, os benefícios a longo prazo em termos de qualidade do software e eficiência no processo de teste compensam muito. Esse modelo não só melhora a abordagem de testes, mas também tem um impacto positivo em todo o ciclo de desenvolvimento, resultando em um produto final mais robusto e confiável.

11 – Construa um exemplo de Solicitação de Sistema

Sistema de Reserva de Salas de Reunião

Responsável pelo projeto: Camila Specalski e Hemerson Lacovic.

Necessidade da Empresa: Este projeto foi criado com o objetivo de otimizar a gestão das salas de reunião.

Este documento descreve os requisitos para o desenvolvimento de um Sistema de Reserva de Salas de Reunião para a empresa Integrado. O sistema deve permitir que os funcionários reservem, visualizem e gerenciem salas de reunião de forma eficiente, melhorando a organização e o uso das instalações.

Requisitos do Negócio

Funcionais

Cadastro e Autenticação

- O sistema deve permitir que os usuários se cadastrem utilizando o e-mail corporativo e uma senha.
- O sistema deve ter funcionalidades de login e logout.
- O sistema deve permitir a recuperação de senha via e-mail.

Reserva de Salas

- Os usuários devem poder visualizar a lista de salas disponíveis e suas capacidades.
- Os usuários devem poder reservar uma sala escolhendo a data, horário de início e fim, e o número de participantes.
- O sistema deve permitir que os usuários adicionem uma descrição ou propósito para a reunião.

Gerenciamento de Reservas

- Os usuários devem poder visualizar suas reservas futuras e passadas.
- Os usuários devem poder cancelar ou alterar uma reserva existente.
- O sistema deve atualizar a disponibilidade das salas em tempo real para refletir as mudanças nas reservas.

Notificações

- O sistema deve enviar um e-mail de confirmação quando uma reserva for feita, alterada ou cancelada.
- O sistema deve enviar um lembrete por e-mail 24 horas antes do início da reunião.

Administração

- Administradores devem poder adicionar, editar ou remover salas de reunião e definir suas capacidades.
- Administradores devem poder visualizar todas as reservas feitas e realizar ajustes conforme necessário.

Não Funcionais

Usabilidade

- O sistema deve ter uma interface intuitiva e fácil de usar, acessível através de navegadores web.
- O sistema deve fornecer feedback claro sobre ações realizadas, como reservas bem-sucedidas ou erros.

Restrições

- O sistema deve ser compatível com os navegadores mais comuns (Chrome, Firefox, Safari, Edge).
- O sistema deve ser desenvolvido utilizando tecnologias web padrão (HTML5, CSS3, JavaScript).

Dependências

- O sistema deve se integrar com o servidor de e-mail da empresa para o envio de notificações.

- O sistema deve utilizar uma base de dados relacional para armazenar informações sobre salas e reservas.

Critérios de Aceitação

- O sistema deve passar em todos os testes funcionais e de usabilidade descritos neste documento.
- O sistema deve ser validado por um grupo de usuários finais durante um período de teste.
- O sistema deve atender aos requisitos de segurança estabelecidos.

Valor agregado:
O Sistema de Reserva de Salas de Reunião deve simplificar e otimizar a gestão das salas de reunião e o uso dos espaços, facilitando a organização das reuniões, evitando conflitos e reservas duplicadas, contribuindo para a eficácia geral das operações da empresa.

Questões especiais ou Restrições
Orçamento de desenvolvimento: R$ 7.000,00
Orçamento para manutenção: R$ 1.000,00
Segurança:

- O sistema deve garantir que os dados dos usuários e das reservas sejam armazenados e transmitidos de forma segura.
- O sistema deve proteger contra acessos não autorizados e vazamentos de dados.

Performance:

- Suporte para até 30 usuários simultâneos.

Prazo:

- O sistema deverá estar funcionando adequadamente dentro de um período de 3 meses.

Tamanho: Pequeno a médio porte.

Time: 1 Gerente de Projeto, 1 Designer, 1 Desenvolvedor front-end, 1 Desenvolvedor back-end, 1 Testador.

Custo Estimado:
Desenvolvimento: R$7.000,00
Testes: R$2.000,00
Implementação e Treinamento: 1.000,00
Manutenção (mensal): R$1.000,00

Proposta: Melhorar a gestão e a utilização das salas de reunião dentro da empresa, substituindo processos manuais por soluções digitais. O sistema implementado deve facilitar o gerenciamento de recursos e aumentar a eficiência operacional, alinhando-se às estratégias de eficiência e produtividade da empresa e promovendo um ambiente de trabalho mais organizado e eficiente.

Extensão: O tempo estimado para a conclusão é de 3 meses, abrangendo desenvolvimento, testes e implementação. O valor será agregado à empresa assim que o sistema entrar em operação, após o desenvolvimento e treinamento dos usuários, começando a ser percebido imediatamente com a melhoria na gestão das salas e otimização dos processos.

Risco: Alta probabilidade de sucesso.
Áreas de Atenção e Riscos de Falha:
Notificações:

- Risco: A implementação de notificações pode apresentar desafios, especialmente se envolver integração com servidores de e-mail e a necessidade de garantir que todos os e-mails sejam entregues corretamente.
- Mitigação: Realizar testes extensivos para verificar a entrega e a precisão das notificações. Certificar-se de que há um sistema de log para rastrear falhas na entrega de e-mails e ter um plano de contingência.

Integração com E-mail Corporativo:

- Risco: A integração com o e-mail corporativo pode ser complexa e suscetível a problemas de configuração ou compatibilidade. Também há preocupações com a segurança e privacidade dos dados.
- Mitigação: Realizar uma análise detalhada da configuração do servidor de e-mail e garantir que a integração seja feita de acordo com as melhores práticas de segurança. Incluir testes de integração específicos para garantir que as mensagens sejam enviadas e recebidas conforme o esperado.

Escopo: O Sistema de Reserva de Salas de Reunião deve abranger diversos departamentos e diferentes unidades da empresa, afetando toda a organização ao facilitar a gestão e a reserva de salas de reunião em diferentes áreas.

Retorno do Investimento: Maior eficiência operacional, redução de custos e melhoria na satisfação dos funcionários, resultando em uma operação mais eficaz e bem gerida. A economia com gestão de conflitos e a otimização do uso dos espaços deve gerar benefícios que superam os custos de desenvolvimento e manutenção do sistema.

12 – Construa um exemplo de Proposta de Sistema.

Proposta de Sistema: Sistema de Reserva de Salas de Reunião

Introdução
Este documento propõe o desenvolvimento de um sistema que permitirá aos funcionários reservar, visualizar e gerenciar as salas de forma otimizada. Este sistema ajudará a melhorar a organização, a utilização das salas e a eficiência geral das operações.

Objetivos do Sistema

- Melhorar a organização: Fornecer uma plataforma centralizada para gerenciar todas as reservas de salas de reunião.
- Aumentar a eficiência: Facilitar a visualização de disponibilidade e a reserva de salas, reduzindo o tempo gasto na coordenação de reuniões.
- Reduzir conflitos de reserva: Atualizar a disponibilidade em tempo real para evitar reservas duplicadas ou conflitos.

Escopo do Sistema
O sistema permitirá aos usuários:

- Verificar a disponibilidade das salas de reunião.
- Fazer reservas para uma sala específica.

- Cancelar ou modificar reservas existentes.
- Receber confirmações e lembretes sobre reservas.

Requisitos Funcionais

Cadastro e Autenticação
- Permitir que os usuários se cadastrem utilizando o e-mail corporativo e uma senha.
- Permitir a recuperação de senha via e-mail.

Reserva de Salas
- Visualizar a lista de salas disponíveis e suas capacidades.
- Reservar uma sala escolhendo a data, horário de início e fim, e o número de participantes.
- Adicionar uma descrição ou propósito para a reunião.

Gerenciamento de Reservas
- Visualizar reservas futuras e passadas.
- Cancelar ou alterar reservas existentes.
- Atualizar a disponibilidade das salas em tempo real.

Notificações
- Enviar um e-mail de confirmação quando uma reserva for feita, alterada ou cancelada.
- Enviar um lembrete por e-mail 24 horas antes do início da reunião.

Administração
- Permitir que administradores adicionem, editem ou removam salas de reunião e definam suas capacidades.
- Visualizar todas as reservas feitas e realizar ajustes conforme necessário.

Requisitos Não Funcionais

Usabilidade
- Interface intuitiva e fácil de usar, acessível através de navegadores web.
- Fornecer feedback claro sobre ações realizadas, como reservas bem-sucedidas ou erros.

Segurança
- Garantir que os dados dos usuários e das reservas sejam armazenados e transmitidos de forma segura.
- Proteger contra acessos não autorizados e vazamentos de dados.

Restrições
- Compatibilidade com navegadores comuns (Chrome, Firefox, Safari, Edge).
- Utilização de tecnologias web padrão (HTML5, CSS3, JavaScript).

Dependências
- Integração com o servidor de e-mail da empresa para notificações.
- Uso de uma base de dados relacional para armazenar informações.

Tecnologias Propostas
- Front-end: HTML5, CSS3, JavaScript, React.

- Back-end: Node.js, Express.js.
- Banco de Dados: MySQL ou PostgreSQL.
- Notificações: Integração com o servidor de e-mail da empresa.

Cronograma de Desenvolvimento
Fase 1: Análise de Requisitos (2 semanas)
Fase 2: Design de Arquitetura e Interface (2 semanas)
Fase 3: Desenvolvimento e Testes Unitários (5 semanas)
Fase 4: Testes de Integração e Ajustes Finais (2 semanas)
Fase 5: Implementação e Treinamento de Usuários (1 semana)

Questões Especiais e Restrições:

Orçamento:
- Desenvolvimento: R$ 7.000,00
- Testes: R$ 2.000,00
- Implementação e Treinamento: R$ 1.000,00
- Manutenção (mensal): R$ 1.000,00

Segurança:
- Garantir armazenamento e transmissão segura de dados.
- Proteção contra acessos não autorizados e vazamentos.
-

Performance:
- Suporte para até 30 usuários simultâneos.

Prazo:
- Implementação dentro de 3 meses.

13 – Construa um exemplo de uma Especificação de Sistema.

Este documento define as especificações detalhadas para o Sistema de Reserva de Salas de Reunião da empresa Integrado. O objetivo é fornecer uma visão clara dos requisitos funcionais e não funcionais do sistema, garantindo que atenda às necessidades dos usuários e da organização.

Escopo do Sistema
O sistema permitirá aos funcionários da empresa reservar, visualizar e gerenciar salas de reunião. Será acessível via navegadores web e integrará com o servidor de e-mail da empresa para notificações.

Requisitos Funcionais

1. Cadastro e Autenticação
- RF-01: O sistema deve permitir que usuários se cadastrem usando e-mail corporativo e senha.
- RF-02: O sistema deve fornecer funcionalidades de login e logout.
- RF-03: O sistema deve permitir recuperação de senha via e-mail.

2. Reserva de Salas
   - RF-04: Os usuários devem poder visualizar a lista de salas disponíveis e suas capacidades.
   - RF-05: Os usuários devem poder reservar uma sala especificando data, horário de início e fim, e número de participantes.
   - RF-06: O sistema deve permitir adicionar uma descrição ou propósito para a reunião.

3. Gerenciamento de Reservas
   - RF-07: Os usuários devem visualizar suas reservas futuras e passadas.
   - RF-08: Os usuários devem poder cancelar ou alterar uma reserva existente.
   - RF-09: O sistema deve atualizar a disponibilidade das salas em tempo real.

4. Notificações
   - RF-10: O sistema deve enviar e-mail de confirmação após uma reserva, alteração ou cancelamento.
   - RF-11: O sistema deve enviar um lembrete por e-mail 24 horas antes do início da reunião.

5. Administração
   - RF-12: Administradores devem adicionar, editar ou remover salas e definir capacidades.
   - RF-13: Administradores devem visualizar todas as reservas e realizar ajustes.

Requisitos Não Funcionais

1. Usabilidade
   - RNF-01: O sistema deve ter uma interface intuitiva, acessível por navegadores web.
   - RNF-02: O sistema deve fornecer feedback claro sobre ações realizadas.

2. Segurança

   - RNF-03: O sistema deve garantir segurança no armazenamento e transmissão dos dados.
   - RNF-04: O sistema deve proteger contra acessos não autorizados e vazamentos de dados.

3. Performance
   - RNF-05: O sistema deve suportar até 30 usuários simultâneos.

4. Compatibilidade
   - RNF-06: O sistema deve ser compatível com os navegadores mais comuns (Chrome, Firefox, Safari, Edge).
   - RNF-07: Deve utilizar tecnologias web padrão (HTML5, CSS3, JavaScript).

Restrições

1. Orçamento
   - R-01: O orçamento para desenvolvimento é R$ 7.000,00.
   - R-02: O orçamento para manutenção é R$ 1.000,00 mensais.

2. Prazo

- R-03: O sistema deve ser implementado e funcional em 3 meses.

Dependências

1. Integração com Servidor de E-mail
   - D-01: O sistema deve integrar com o servidor de e-mail da empresa para envio de notificações.

2. Base de Dados
   - D-02: O sistema deve utilizar uma base de dados relacional para armazenar informações sobre salas e reservas.

Critérios de Aceitação
- CA-01: O sistema deve passar em todos os testes funcionais e de usabilidade descritos.
- CA-02: Deve ser validado por um grupo de usuários finais durante um período de teste.
- CA-03: Deve atender aos requisitos de segurança estabelecidos.

Tecnologias Propostas
Front-end: React.js,
Back-end: Node.js, Express.js
Banco de Dados: PostgreSQL
Notificações: Integração com SendGrid para e-mails e Firebase Cloud Messaging para notificações push
Real-time Updates: WebSockets (Socket.io)

14 – Dê exemplos de documentações que poderiam auxiliar a execução dos testes.

Na execução de testes de software, diversas documentações podem ser úteis para guiar e estruturar o processo. Alguns exemplos:

Especificação de Requisitos
Este documento detalha todas as funcionalidades esperadas do software. Serve como base para criar casos de teste e verificar se o sistema atende às expectativas do cliente.

Plano de Teste
O plano de teste descreve a estratégia geral, objetivos, escopo, cronograma e recursos necessários para os testes. É um guia abrangente para toda a atividade de teste.

Casos de Teste
Os casos de teste documentam cenários específicos a serem testados, incluindo passos detalhados, dados de entrada e resultados esperados. São essenciais para testes sistemáticos e reproduzíveis.

Matriz de Rastreabilidade

A matriz de rastreabilidade relaciona requisitos a casos de teste. Garante que todas as funcionalidades sejam testadas adequadamente.

Manual do Usuário
O manual do usuário fornece instruções sobre como usar o software do ponto de vista do usuário final. Útil para testes de usabilidade e para verificar se o software funciona conforme descrito para o usuário final.

Documentação da API
A documentação a API detalha como interagir corretamente com a API. Crucial para testes de integração.

Relatórios de Testes Anteriores
Os relatórios de testes anteriores fornecem contexto histórico dos testes realizados anteriormente. Podem ajudar a identificar áreas problemáticas que merecem atenção especial.

Diagramas de Fluxo
Os diagramas que ilustram os processos do sistema. Auxiliam na compreensão dos processos do sistema, úteis para planejar testes de fluxo de trabalho.

Estas documentações, quando bem elaboradas e mantidas atualizadas, podem melhorar significativamente a eficácia e eficiência do processo de teste, garantindo uma cobertura abrangente e resultados confiáveis.

15 – Pesquise e apresente 3 tipos de testes diferentes.

Testes Unitários

Os testes unitários focam em testar unidades individuais de código, geralmente funções ou métodos, para garantir que cada parte do código funcione corretamente de forma isolada. Esses testes são realizados pelos desenvolvedores durante o processo de codificação e são geralmente automatizados e executados frequentemente.

Benefícios

- Identificação precoce de bugs: Facilita a detecção de erros logo no início do ciclo de desenvolvimento.
- Facilidade de manutenção: Ajuda a garantir que mudanças no código não quebrem funcionalidades existentes.
- Documentação do código: Servem como documentação sobre como as unidades de código devem funcionar.
- Rapidez: Os testes unitários são rápidos de executar, permitindo uma verificação constante durante o desenvolvimento.

Ferramentas Comuns

- JUnit: Usado para testes unitários em Java.
- NUnit: Usado para testes unitários em .NET.
- Jest: Usado para testes unitários em JavaScript/TypeScript.
- pytest: Usado para testes unitários em Python.

Testes de Usabilidade
Os testes de usabilidade avaliam a facilidade de uso e a experiência do usuário com o software. Envolvem usuários reais interagindo com o sistema, buscando identificar problemas de interface, navegação e design. Esses testes podem ser realizados em protótipos ou no produto final.

Benefícios

- Melhoria da experiência do usuário: Identifica problemas de usabilidade que podem ser corrigidos para melhorar a interação do usuário com o sistema.
- Aumento da satisfação do usuário: Um sistema mais fácil de usar resulta em usuários mais satisfeitos.
- Identificação de problemas de design: Ajuda a detectar falhas na interface do usuário que podem não ser evidentes apenas com testes funcionais.

Métodos Comuns

- Testes com usuários reais: Observação direta de usuários interagindo com o sistema para identificar dificuldades e áreas de melhoria.
- Entrevistas e questionários: Coleta de feedback dos usuários sobre suas experiências.
- Análise heurística: Especialistas em usabilidade avaliam a interface com base em princípios de design estabelecidos.

Testes de Caixa Preta
Os testes de caixa preta, também conhecidos como testes funcionais, focam nas entradas e saídas do sistema, sem considerar o código interno. Verificam se o software atende aos requisitos especificados e podem ser realizados por testadores que não conhecem a implementação interna.

Benefícios

- Perspectiva do usuário: Avalia o sistema do ponto de vista do usuário final, garantindo que os requisitos funcionais sejam atendidos.
- Detecção de erros em interfaces: Identifica problemas na interação entre diferentes componentes do sistema.
- Independência da implementação: Pode ser realizado sem conhecimento detalhado do código, permitindo a participação de testadores não técnicos.

Técnicas Comuns

- Particionamento de equivalência: Divide as entradas possíveis em classes de equivalência para reduzir o número de casos de teste.
- Análise do valor limite: Testa os limites das classes de equivalência para garantir que o sistema lide corretamente com entradas nos extremos.
- Teste de tabela de decisão: Usa tabelas para representar combinações de entradas e saídas esperadas, garantindo cobertura completa de todas as possíveis interações.