HELLO NEIGHBOR: VALIDAÇÃO INTELIGENTE E GESTÃO DE OCORRÊNCIAS EM CONDOMÍNIOS

Bruno Souza de Oliveira [1], Graziela dos Santos Nunes [2], Orientador: Prof. Me. Gregorio Perez Peiro.
[1]7-CCOMP-00354825, [2]6-CCOMP-00345345.

RESUMO

O Presente artigo descreve a elaboração de um sistema inteligente para gestão de ocorrências em condomínios o Hello Neighbor, onde serão explorados as tecnologias e desafios que envolveram seu desenvolvimento, tendo como objetivo uma gestão mais eficaz e prática. Desenvolvido com HTML, CSS, JavaScript, Node.js e MongoDB, tecnologias que tiveram suma importância para proporcionar uma estrutura robusta e prática. A solução visa agilizar processos, reduzir inconsistências e aumentar a transparência na administração do condomínio, garantindo um fluxo seguro e previsível.[1]

Palavras-chave: Gestão de Condomínios; JavaScript; Software; Sistemas Inteligentes.

1. Introdução

A crescente verticalização das grandes cidades brasileiras tem transformado profundamente o modo de viver em comunidade. O aumento no número de edifícios residenciais, impulsionado pela densidade populacional e pela escassez de espaço urbano, trouxe consigo novos desafios de convivência, segurança e administração nos condomínios. Nesse cenário, a organização eficiente da rotina condominial tornou-se uma necessidade, especialmente no que diz respeito ao registro, acompanhamento e resolução de ocorrências internas, como reclamações, solicitações de manutenção ou incidentes entre moradores. [2]

Neste contexto, Hello Neighbor apresenta o desenvolvimento de um sistema voltado à gestão de ocorrências em condomínios, com foco na aplicação de fundamentos para a validação de dados e o controle de estados das ocorrências. A proposta visa garantir que todas as entradas de dados sigam padrões bem definidos e que o fluxo das ocorrências respeite regras consistentes e previsíveis, contribuindo para uma gestão mais ágil, segura e transparente dos ambientes residenciais compartilhados.

2. Metodologia

O projeto foi estruturado em camadas: interface do usuário (front-end), regras de negócio (validação e transição de estados) e persistência de dados (back-end). A seguir, detalham-se as principais etapas e decisões metodológicas adotadas.

1. Tecnologias Utilizadas

O sistema foi implementado utilizando uma combinação de tecnologias que garantem tanto a funcionalidade quanto a usabilidade da aplicação. A estrutura e o estilo da interface de entrada de dados e visualização de ocorrências foram desenvolvidos com HTML e CSS, proporcionando uma apresentação clara e organizada das informações. A lógica de validação, as transições de estado e a interação dinâmica com o usuário foram implementadas em JavaScript, assegurando uma experiência mais fluida e responsiva. Para garantir a integridade dos dados inseridos, foram utilizadas expressões regulares na validação dos campos de entrada. Já o armazenamento e a recuperação das ocorrências

em tempo real foram viabilizados através do uso conjunto de Node.js e MongoDB, permitindo uma gestão eficiente e segura das informações.

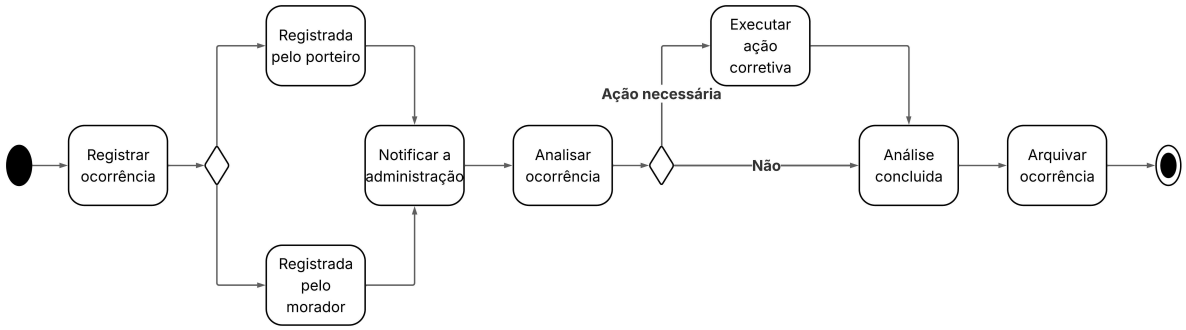
2. Validação de Dados

A validação de dados foi implementada como uma primeira camada de segurança e consistência no sistema. Para isso, utilizaram-se expressões regulares, que são representações formais de linguagens regulares, com o objetivo de identificar padrões específicos em campos sensíveis. Entre os dados validados, destacam-se as placas de veículos, verificadas conforme o padrão brasileiro (por exemplo, ABC-1234), os números de CPF, aceitando formatos com ou sem máscara, e os campos de data e hora, exigindo o formato YYYY-MM-DD HH\:MM. Além disso, campos obrigatórios como descrições ou categorias de ocorrência foram validados para garantir que não fossem deixados em branco. Essa verificação foi realizada diretamente no front-end por meio de JavaScript, evitando o envio de informações incorretas ao servidor e proporcionando uma experiência mais confiável ao usuário. No back-end, essas validações podem ser replicadas para reforçar a integridade geral do sistema.

3. Modelagem dos Estados das Ocorrências

A modelagem do fluxo de vida de uma ocorrência foi representada por meio de um diagrama de atividades da UML, conforme ilustrado na Figura 1 a seguir:

Figura 1. Diagrama de atividades mostrando o fluxo de uma ocorrência



O processo descrito no diagrama apresenta o fluxo de tratamento de uma ocorrência no contexto de um condomínio. Ele tem início com o registro da ocorrência, que pode ser realizado tanto por um porteiro quanto por um morador. Independentemente de quem realiza o registro, a informação é direcionada à administração responsável.

Uma vez notificada, a administração procede com a análise da ocorrência, avaliando sua natureza, gravidade e a necessidade de intervenção. Após essa análise, uma decisão é tomada: se for identificada a necessidade de ação corretiva, esta é executada de forma apropriada. Caso contrário, o processo segue diretamente para a conclusão da análise.

Com ou sem a execução de ações corretivas, todas as ocorrências são devidamente finalizadas e arquivadas, encerrando-se formalmente o fluxo. Esse modelo garante que todas as ocorrências sejam tratadas de forma padronizada, promovendo rastreabilidade, responsabilidade administrativa e uma gestão eficiente dos eventos registrados no ambiente. [3]

A lógica dessas transições foi implementada em JavaScript por meio de uma função, garantindo que mudanças de estado só ocorram mediante ações válidas e por usuários com permissões apropriadas.

4. Interface de Usuário

A interface foi projetada para ser intuitiva, com formulários claros e feedbacks imediatos de validação. Cada ocorrência pode ser visualizada com seu estado atual, histórico de transições e botões de ação (como "Analisar", "Resolver" ou "Arquivar") disponíveis conforme situação atual e o perfil do usuário.

5. Persistência de Dados

Para ampliar a escalabilidade do sistema, a API pode ser integrada a um back-end desenvolvido com Node.js e um banco de dados, como MongoDB ou SQLite. Nessa arquitetura, as ocorrências são armazenadas contendo um ID único, dados do morador ou autor, descrição, data e hora do registro, além do estado atual e do histórico de transições. As regras de transição e validação também podem ser replicadas ou reforçadas no back-end, com o objetivo de aumentar a segurança do sistema.

6. Testes e Validação do Modelo

Foram realizados testes de uso simulando diferentes cenários de entrada inválida, transições de estado corretas e incorretas, além de verificação de resposta da interface a essas situações. O objetivo foi garantir que o autômato implementado se comportasse de acordo com o modelo teórico definido e que os dados fossem sempre consistentes.

Figura 2. Exemplo de teste para garantir que o RA (ID) dos funcionários seja apenas números.

```
// Verificação para garantir que RA contenha apenas números
if (!/^\d+$/.test(funcionario.ra)) erros.push("O RA deve conter apenas números.");
if (!/^\d+$/.test(funcionario.tel)) erros.push("O tel deve conter apenas números.");
if (!/^\d+$/.test(funcionario.salario)) erros.push("O salario deve conter apenas números.");
if (!/^[A-Za-zÀ-ÖØ-öø-ÿ\s]+$/.test(funcionario.nome)) erros.push("O nome deve conter apenas letras.");
if (!/^[A-Za-zÀ-ÖØ-öø-ÿ\s]+$/.test(funcionario.area)) erros.push("A area deve conter apenas letras.");

if (!funcionario.salario) erros.push("O salário não pode estar vazio.");
if (!funcionario.tel) erros.push("O telefone não pode estar vazio.");
if (!funcionario.area) erros.push("A área não pode estar vazia.");

// if (!validarRa(funcionario.ra)) erros.push("RA é inválido.");
// if (!validaSalario(funcionario.salario)) erros.push("Salário é inválido.");
```

3. Desenvolvimento

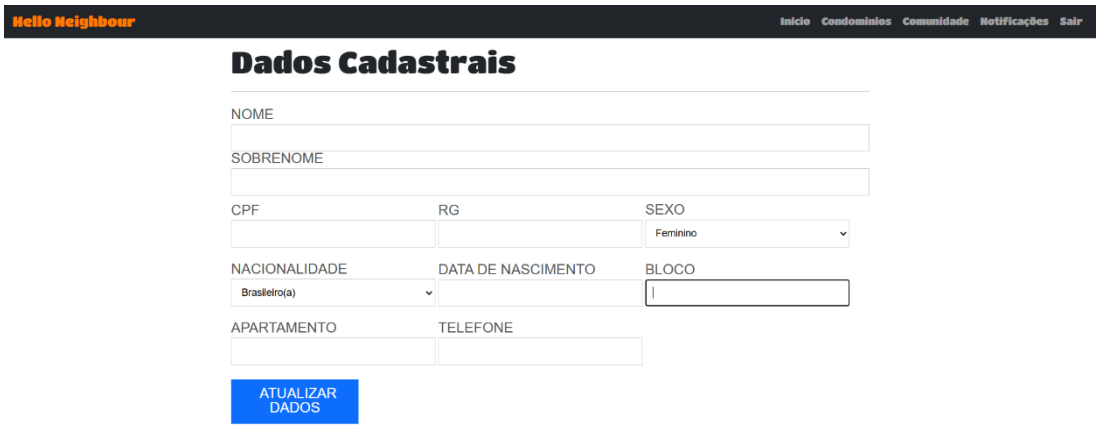
O desenvolvimento do sistema seguiu uma abordagem estruturada em camadas, garantindo a separação de responsabilidades entre a interface do usuário (front-end), a lógica de negócio (validação e transição de estados) e a persistência de dados (back-end). Essa arquitetura modular facilitou a manutenção, a escalabilidade e a consistência da aplicação. A escolha das tecnologias foi fundamental para o sucesso do projeto. O uso de HTML e CSS permitiu a criação de uma interface intuitiva e responsiva, enquanto o JavaScript viabilizou a implementação de validações dinâmicas e interações fluidas. A combinação de Node.js e MongoDB garantiu um armazenamento eficiente e seguro dos dados, atendendo aos requisitos de desempenho e confiabilidade.[4][5]

Além disso, a aplicação de linguagens formais e expressões regulares proporcionou um mecanismo robusto para validação de dados, garantindo a integridade das informações inseridas no sistema. A otimização desses processos, inspirada em técnicas de minimização de autômatos, contribuiu para a eficiência computacional do sistema, mesmo quando lidando com um volume significativo de ocorrências simultâneas. Essa combinação de fundamentos teóricos e soluções práticas resultou em um sistema não apenas funcional, mas também escalável e adaptável a diferentes contextos de gestão condominial.

A tela de cadastro do sistema Hello Neighbor foi projetada para garantir a coleta eficiente e segura dos dados dos usuários (moradores), seguindo padrões de usabilidade e validação robusta. Como ilustrado na Figura 3, a interface apresenta um formulário intuitivo, organizado em seções lógicas: informações pessoais (nome completo, CPF, data de nascimento), dados de contato (telefone) e vínculo

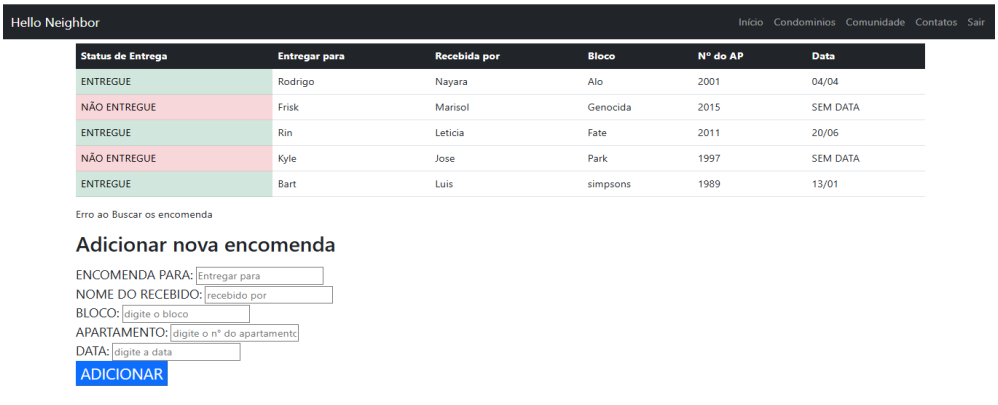
condominial (unidade). Cada campo implementa validação em tempo real — utilizando expressões regulares para formatos específicos (como CPF e telefone) e verificações de obrigatoriedade — com feedback visual imediato (ícones de validação e mensagens contextualizadas). Para garantir a privacidade, dados sensíveis serão mascarados durante a digitação (como CPF) e a tela inclui um termo de consentimento explícito para tratamento de dados, em conformidade com a LGPD (uma futura implementação). A disposição dos elementos segue princípios de design responsivo, adaptando-se a diferentes dispositivos sem perder funcionalidade.

Figura 3. Tela de dados cadastrais dos moradores do condominio



No front-end, aplicamos HTML para a estrutura básica do formulário, CSS para a estilização visual e JavaScript para implementar as validações em tempo real e a interatividade da interface. O back-end foi construído com Node.js e o framework Express para criar uma API eficiente, enquanto o MongoDB foi escolhido como banco de dados devido à sua flexibilidade no armazenamento das informações sobre as encomendas. O processo funciona de maneira integrada: quando um porteiro registra uma nova encomenda através do formulário, o JavaScript válida os dados e gera um código único antes de enviar as informações para o servidor Node.js, que por sua vez armazena os dados no MongoDB. A listagem das encomendas é atualizada dinamicamente, permitindo buscas e filtros por status ou morador como descrito na Figura 4.

Figura 4. Tela de encomendas mostrando todas as encomendas registradas e seus status.



4. Considerações Finais

O projeto desenvolveu um sistema eficiente para gestão de ocorrências, utilizando uma arquitetura em camadas (front-end, regras de negócio e back-end) que garantiu organização e facilidade de manutenção. A combinação de HTML, CSS e JavaScript permitiu uma interface dinâmica e validada, enquanto Node.js e MongoDB asseguraram armazenamento confiável e escalável. Como trabalhos

futuros, sugere-se a integração com APIs de notificação em tempo real, análise de dados para identificação de padrões recorrentes e a implementação de mecanismos de autenticação mais robustos. Dessa forma, o Hello Neighbor não apenas resolve problemas imediatos de administração condominial, mas também abre caminho para inovações em sistemas inteligentes de convivência urbana. Como trabalhos futuros, sugere-se a implementação de autenticação avançada (como OAuth 2.0) e a integração com sistemas externos, como notificações por e-mail ou mensagens automáticas. Além disso, a adoção de testes automatizados (como Jest ou Mocha) poderia aumentar ainda mais a confiabilidade do sistema.

5. Referências

[1] NUNES, G.; CARVALHO, M.; et al. “Hello Neighbor e uso de IOT como sistema de monitoramento”, Revista Interação; VI CONGRESSO ACADÊMICO DE TECNOLOGIA DA INFORMAÇÃO DA FAM – CATI. São Paulo, | v.-17-n.-1-2024 [Acesso 8 de maio 2025]. Disponível em: https:// https://www.vemprafam.com.br/wp-content/uploads/2024/04/v.-17-n.-1-2024.pdf?p=revista-interacao&x=0&y=0

[2] QUEIROZ, C. Cidades brasileiras estão mais verticais, Revista Pesquisa Fapesp; urbanismo. São Paulo, | edição 338 abr. 2024 [Acesso em 5 maio 2025]. Disponível em: https://revistapesquisa.fapesp.br/cidades-brasileiras-estao-mais-verticais/.

[3] FERREIRA, Jeferson; MARTINS, Eliane. Fluxo de Exceções Intraprocedimentais a partir do Diagrama de Atividades da UML 2.0. Instituto de Computação Universidade Estadual de Campinas, Campinas, São Paulo, 2010.

[4] FOWLER, M. Refatoração: Aperfeiçoando o Design de Códigos Existentes. 2. ed. Bookman, 2019.

[5] Lucas, W., Nunes, R., Bonifácio, R; et al. Understanding the adoption of modern Javascript features: An empirical study on open-source systems. Empir Software Eng 30, 107 –2025[Acesso em 15 de maio 2025]. Disponível em: https://link.springer.com/article/10.1007/s10664-025-10663-9