Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Campus Apucarana
Curso de Bacharelado em Engenharia de Computação

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ANNA BEATRIZ CORREIA DOS SANTOS

BIANCA GABRIELLY SIQUEIRA

GABRIELLE KYOKO ALVES ABE

TRABALHO 2: ENGENHARIA DE SOFTWARE

APUCARANA

2025

SUMÁRIO

1 Papéis e Responsabilidades.....3
2 Padrões de Documentação.....3
3 Padrões não funcionais..... 4
4 Atividade e métricas de garantia.....4
5 Análise dos requisitos do projeto:.....4
6 Padrão de Arquitetura Base:..... 4
7 Conexão - proposta com o projeto..... 5
8 Especificação e documentação.....6
8.1 Banco de dados..... 6
8.2 Interface..... 6
9 Padrões de Projeto..... 7

1 Papéis e Responsabilidades

Para o desenvolvimento do código do projeto, foi conversado em equipe sobre os papéis e as responsabilidades que cada integrante seria responsável, nisso, foi decidido que as responsabilidades dos integrantes seriam:

Membro	Responsabilidade
Anna Beatriz Correia dos Santos	• Responsável pela implementação e manutenção do Banco de Dados (SQLite com Qt SQL). Inclui a implementação de create, update e delete para usuários e a garantia de segurança por meio do hashing nos dados de autenticação.
Bianca Gabrielly Siqueira	• Responsável pelo design e implementação das telas do aplicativo pós-login, como a a…;
Gabrielle Kyoko Alves Abe	• Responsável pela criação e manutenção das páginas de Login e Principal, incluindo o desenvolvimento do código C++.

2 Padrões de Documentação

A documentação do projeto foi dividida entre Documentação de Projeto e Documentação de Código. O repositório GitHub foi a fonte principal das alterações realizadas no projeto. O código foi documentado utilizando comentários que descrevem as classes e funcionalidades, permitindo com que a manutenção e entendimento sejam fluidas e entendíveis por todos os membros. O ambiente de desenvolvimento padrão é o Qt Creator, onde o sistema de build utilizado é o CMake.

3 Padrões não funcionais

O padrão não funcional adotado (de segurança) para fazer a autenticação do usuário através do login e assim armazenar os dados dos usuários, é baseado em Hashing Criptográfico. A senha é armazenada por meio de hashing, não em texto puro, diminuindo assim o risco de comprometimento em casos de acesso não autorizado ao banco de dados.

4 Atividade e métricas de garantia

O processo de Garantia de Qualidade (QA) do projeto se concentra em testes funcionais e de usabilidade. Testes de Integração de Ponta a Ponta são realizados nas funcionalidades críticas de Login e Criação de Conta para validar o fluxo de autenticação e a integridade dos dados no banco.

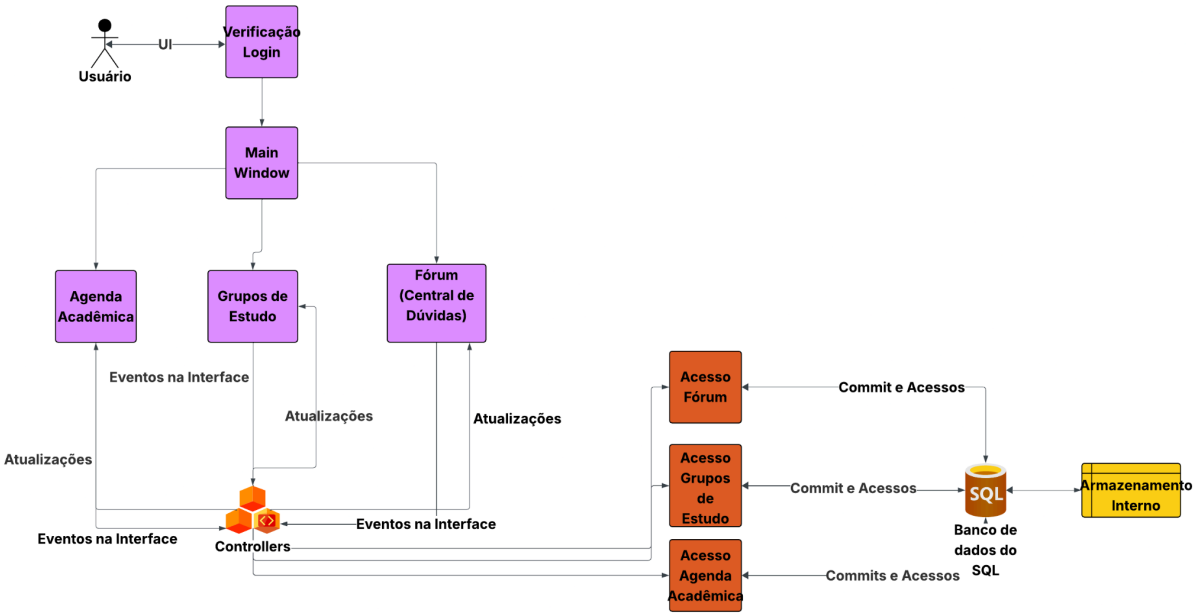
5 Análise dos requisitos do projeto:

Depois do levantamento de requisitos (T1 - primeira entrega), foram identificados como requisitos funcionais principais: cadastro de usuário(login) e publicação de comentários. Os requisitos não funcionais adotados para o projeto são: interface responsiva, autenticação do usuário para entrar no sistema e persistência no banco de dados.

6 Padrão de Arquitetura Base:

De acordo com a análise dos requisitos do projeto (considerando tanto a ideia inicial quanto os ajustes feitos com base nas HU ‘s), escolhemos o padrão de arquitetura Model-View-Controller (MVC). Esse padrão foi escolhido porque era o que melhor comportava o desenvolvimento do nosso projeto e também por conveniência e decisão entre as integrantes da equipe, que optaram por aproveitar uma base de código já pronto de outro projeto passado, adequando essa base às necessidades do projeto atual.

O MVC é dividido em três partes: Model, View e Controller. Essa separação tem como objetivo organizar o código, facilitar a manutenção e permitir o desenvolvimento paralelo entre a parte visual e a parte lógica da aplicação. Um exemplo dessa arquitetura é o diagrama de padrão que fizemos como parte do desenvolvimento do projeto:



As vantagens desse padrão são: separação clara da interface com o controle e lógica do sistema, facilidade de manutenção e evolução do sistema, já que ele permite o desenvolvimento paralelo (frontend e backend). As desvantagens são que devido essa modularização, a complexidade do código pode aumentar exigindo maior organização e disciplina das integrantes da equipe, ademais a comunicação entre as camadas pode mandar mais tempo de implementação, pois com o desenvolvimento paralelo a integração e junção de todas as partes no final pode ser complicada e difícil, podendo uma atrapalhar a outra.

7 Conexão - proposta com o projeto

A proposta está diretamente conectada ao objetivo central do projeto, que é facilitar a interação entre estudantes e professores em um ambiente digital de aprendizado, uma vez que a adoção do padrão MVC possibilita o desenvolvimento modular, permitindo a fácil adição de novas funcionalidades, como fóruns e compartilhamento de materiais.

Essa estrutura técnica apoia o cumprimento dos requisitos funcionais levantados, garantidos pela seguinte estrutura da arquitetura escolhida:

- Model: é representado pelo banco de dados SQLite e pelas classes que realizam as operações de dados (INSERT,SELECT,UPDATE,DELETE), gerenciando a lógica de entidades nas páginas “Criar Conta”, “Login” e “Central de Dúvidas” (Ou Fórum).
- View: é composta pelos arquivos “.ui” criados no Qt Designer, que definem a interface e a estrutura visual apresentada ao usuário.
- Controller: Controller é implementado pelas classes c++, responsáveis por fazer a ponte entre a interface (View) e o banco de dados (Model), processando as interações do usuário e gerenciado a lógica da aplicação.

Dessa forma, a arquitetura escolhida apoia diretamente o cumprimento dos requisitos levantados, promovendo uma implementação organizada, flexível e de fácil manutenção, visto que uma alteração na interface (arquivos .ui) não exige, em geral, modificações na lógica de dados. Além disso, a camada de controle (Arquivos C++) gerencia o fluxo da aplicação, interligando as funcionalidades necessárias.

8 Especificação e documentação

A proposta foi especificada e documentada de forma a garantir clareza no desenvolvimento e rastreabilidade entre os requisitos e as funcionalidades implementadas.

Foram utilizados diagramas para representar a estrutura e o comportamento do sistema, facilitando o entendimento por toda a equipe.

8.1 Banco de dados

Inicialmente, foi decidido que o banco de dados poderia ser armazenado em um servidor. No entanto, durante o desenvolvimento do projeto, a equipe optou pelo armazenamento local até o presente momento.

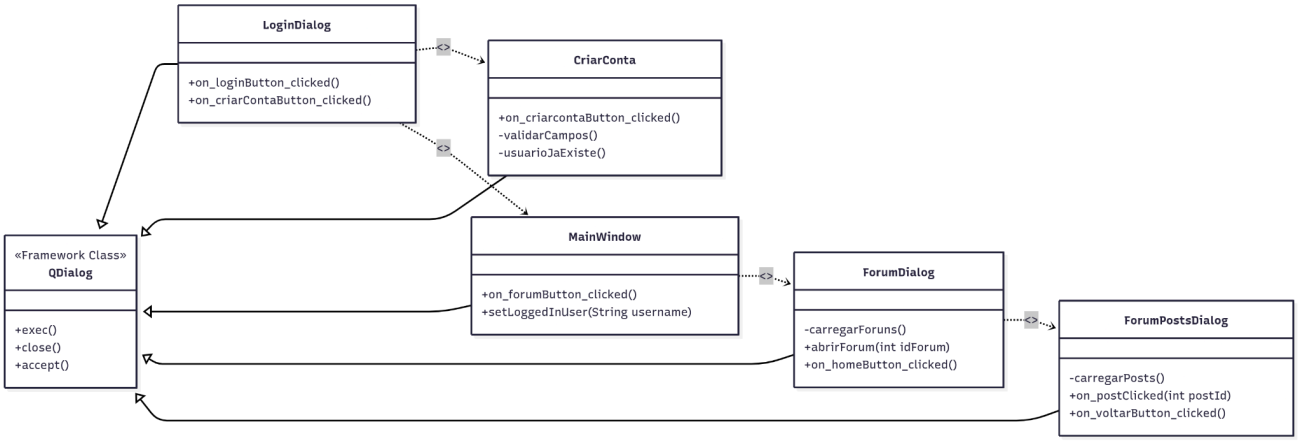
A portabilidade e a facilidade de distribuição foram as características principais para a escolha desse padrão, visto que a abordagem local não exige a configuração de um servidor.

Futuramente, poderemos utilizar um servidor para o armazenamento desses dados, com o intuito de aumentar a confiabilidade e a consistência.

8.2 Interface

Nas páginas de “Fóruns” o conteúdo precisa ser atualizado constantemente para a exibição de novas postagens. Por essa razão, a utilização
é necessário uma constante atualização sobre o que estará sendo passado, com isso, adicionar botões estáticos viraram uma opção inviável, visto que conforme o número de perguntas e de fóruns criados aumenta, os botões fixados não atenderam e não executariam todos os dados existentes na UI, sendo assim, foi decidido que a implementação de uma geração de widgets seria necessário.

9 Padrões de Projeto



Singleton e StrategyOs Design Patterns (Padrões de Projeto) são soluções elegantes e reutilizáveis para problemas comuns no desenvolvimento de software. Os padrões Singleton e Strategy são dois exemplos clássicos, cada um resolvendo um tipo diferente de problema de design.

O Singleton é o padrão que assegura que uma classe tenha apenas uma única instância durante todo o ciclo de vida da aplicação, ele serve para: controlar o acesso a recursos caros ou compartilhados, evitar que múltiplas instâncias causem inconsistências no sistema. Ou seja, é um padrão para centralizar recursos e garantir que você nunca terá mais de um daquele objeto.

O Strategy define uma família de algoritmos e os torna intercambiáveis em tempo de execução. Ele serve para: eliminar grandes blocos de if/else ou switch/case, permite que o comportamento de uma classe seja alterado dinamicamente (ex: mudar o método de cálculo de frete ou imposto), garante que novos algoritmos possam ser adicionados sem modificar o código existente (Princípio Aberto/Fechado). Ou seja, é um padrão para encapsular algoritmos e permitir que o cliente escolha o comportamento na hora, sem mudar a estrutura da classe.