Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Rede Descentralizada de Compartilhamento de Conteúdo e Serviços

Oromar Voit de Rezende

JUIZ DE FORA
MARÇO, 2024

Rede Descentralizada de Compartilhamento de Conteúdo e Serviços

Oromar Voit de Rezende

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Marcelo Ferreira Moreno

JUIZ DE FORA

MARÇO, 2024

Rede Descentralizada de Compartilhamento de Conteúdo e Serviços

Oromar Voit de Rezende

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Marcelo Ferreira Moreno
Doutor em Informática - PUC-Rio

Carlos Pernisa Júnior
Doutor em Comunicação e Cultura - UFRJ

Eduardo Barrere
Doutor em Engenharia de Sistemas e Computação - COPPE/UFRJ

Stanley Cunha Teixeira
Doutor em Tecnologias da Inteligência e Design Digital - PUC-SP

JUIZ DE FORA
14 DE MARÇO, 2024

Resumo

Este trabalho propõe um modelo de arquitetura para a *Timelink*, uma rede *peer-to-peer* descentralizada voltada para o compartilhamento eficiente de conteúdo. A arquitetura foi planejada para operar em um ambiente distribuído, sem hierarquia entre os nós, garantindo que todos os dispositivos tenham o mesmo nível de importância e possam funcionar de maneira independente. Além disso, a proposta visa proporcionar uma estrutura escalável, flexível e confiável, capaz de alcançar consistência eventual e replicar conteúdo de forma eficiente.

Também foi desenvolvido um protótipo funcional adaptado ao contexto educacional. Para isso, foram adotadas premissas simplificadas, como disponibilidade constante dos nós e replicação integral do conteúdo, o que permitiu demonstrar a viabilidade prática da arquitetura. Os resultados mostraram que a organização modular em camadas hierárquicas facilita a escalabilidade e a manutenção, permitindo a adaptação a diferentes cenários de uso.

Palavras-chave: Rede de Entrega de Conteúdos, Redes *peer-to-peer*, *Timelink*, Sistema de Arquivos Distribuído

Abstract

This work proposes an architectural model for *Timelink*, a decentralized peer-to-peer network focused on efficient content sharing. The architecture was designed to operate in a distributed environment, without hierarchy among the nodes, ensuring that all devices have the same level of importance and can function independently. Additionally, the proposal aims to provide a scalable, flexible, and reliable structure capable of achieving eventual consistency and efficiently replicating content.

A functional prototype adapted to the educational context was also developed. To achieve this, simplified assumptions were adopted, such as constant node availability and full content replication, which allowed demonstrating the practical feasibility of the architecture. The results showed that the modular organization in hierarchical layers facilitates scalability and maintenance, enabling adaptation to different usage scenarios.

Keywords: Content Delivery Network, peer-to-peer Network, *Timelink*, Distributed File System

Agradecimentos

A Deus, pela força, sabedoria e inspiração que me guiaram ao longo dessa jornada, ajudando-me a superar cada desafio.

Aos meus pais, pelo amor incondicional, pelo apoio constante e por acreditarem em mim em todos os momentos da minha vida.

Aos meus amigos, que torceram por mim, me ajudaram e insistiram para que esse trabalho fosse concluído. A amizade e o incentivo de vocês foram fundamentais para que eu chegasse até aqui.

A todos os colaboradores da Universidade Federal de Juiz de Fora que, de alguma forma, contribuíram para o meu crescimento pessoal e profissional ao longo desses anos. Em especial, ao professor e orientador Marcelo Ferreira Moreno, pela orientação valiosa, paciência e dedicação ao longo de todo o processo. Sem o seu suporte e ensinamentos, este trabalho não teria se concretizado.

Ao Thomás Marques Brandão Reis, pela parceria no desenvolvimento do aplicativo LD.Edu e por sua contribuição essencial para a implementação do projeto em Lima Duarte.

Este trabalho contou com o apoio da Rede Integrada de Pesquisa em Alta Velocidade (RePesq) da UFJF.[1]

[1] ⟨https://repesq.ufjf.br/⟩

"O que você quer que eu faça? Que eu me fantasie e dance a Hula?".

Timão

Conteúdo

Lista de Figuras

Lista de Abreviações

API	*Application Programming Interface*
APPA	*Atlas Peer-to-Peer Architecture*
CDN	*Content Delivery Network*
CID	*Content Identifier*
IPFS	*Interplanetary File System*
IPNS	*Interplanetary Name System*
IRM	*Integrated file Replication and consistency Maintenance mechanism*
P2P	*peer-to-peer*
SBTVD-T	Sistema Brasileiro de TV Digital Terrestre

1 Introdução

Nos últimos anos, é possível observar que existe uma crescente oferta de produtos e serviços nas mídias online. No entanto, o acesso à internet de qualidade ainda é um desafio persistente. Segundo o IBGE (2021), no ano de 2019 a Internet estava presente em 82,7% dos lares brasileiros. No entanto, de acordo com uma pesquisa realizada pela NIC.br em 2024, apenas 22% desses usuários possuem condições satisfatórias de conectividade (CRAIDE, 2024). Esses dados refletem um problema que não surgiu recentemente, mas que se mantém ao longo dos anos e afeta uma quantidade significativa de pessoas que estão à margem do processo de inclusão digital.

Por outro lado, o IBGE (2021) estima que a televisão está presente em 96,3% dos domicílios particulares permanentes. Ou seja, a televisão ainda é o principal meio de transmissão de mídia visual para a população. Apesar disso, diferentemente da Internet, os telespectadores são apenas consumidores de uma grade fixa de conteúdo.

Para resolver esse problema, Teixeira (2018) propõe o conceito de *Timelink* (rede no tempo). A ideia é substituir a *timeline* (linha do tempo), conceito atualmente empregado na programação televisiva. Neste novo paradigma, os receptores de TV digital podem acessar o conteúdo dos canais abertos fora do horário original de transmissão.

Uma solução possível seria utilizar receptores de TV dedicados para gravar o conteúdo de diferentes emissoras, armazenando-o em um grande volume de dados em um ponto central, onde esses receptores estariam instalados. Esse repositório centralizado poderia, então, distribuir o conteúdo para os usuários, permitindo o consumo sob demanda.

No entanto, essa abordagem implicaria em desafios significativos de infraestrutura. Primeiramente, seria necessário armazenamento massivo para acomodar o grande volume de dados gerado pelas gravações de múltiplas emissoras. Além disso, para compartilhar o conteúdo de forma eficiente, seria exigida uma excelente conectividade entre o repositório central e a rede de distribuição, além de conectividade razoável para cada cliente.

Para escalar essa solução, seria necessário o uso de *Content Delivery Networks* (CDNs), similar ao que é feito por serviços de *streaming* comerciais. No entanto, o alto custo de implementação e manutenção de CDNs seria inevitavelmente repassado aos consumidores, fazendo com que o acesso ao conteúdo televisivo gratuito se tornasse pago.

Para viabilizar a *Timelink*, é necessário desenvolver um suporte tecnológico que permita a disseminação de conteúdo. Esse deve ser capaz de formar uma rede colaborativa descentralizada, de forma que os colaboradores possam capturar o conteúdo da televisão e disponibilizá-lo. Dessa maneira, a mídia pode ser replicada conforme demanda e sua capacidade de absorção de conteúdo cresce com a entrada de novos colaboradores. Essa alternativa descentralizada distribui a carga de armazenamento e conectividade de maneira colaborativa e econômica, mantendo o acesso gratuito ao conteúdo.

É importante observar que essa tecnologia suporta também conteúdo gerado pelos usuários e comunidades e outros tipos de mídia além do audiovisual. Nessa mesma plataforma distribuída podem ser integradas informações de utilidade pública, conteúdo audiovisual educativo ou aplicações específicas de serviços públicos digitais.

Como resultado, temos que essa rede tem elevado potencial de inclusão digital, uma vez que viabiliza o acesso a serviços digitais antes presentes apenas na Internet, como telemedicina, aulas remotas ou vídeos de entrenimento.

Esse potencial se mantém e, de fato, se expande, mesmo com a chegada, em futuro próximo, da tecnologia de TV digital de próxima geração denominada DTV+. Também conhecida como TV 3.0, a plataforma de TV orientada a aplicativos DTV+ aprofunda a integração da radiodifusão com serviços online de streaming e dados (MORENO et al., 2023). Mas com a falta de conectividade significativa, as vantagens da DTV+ poderão ser exploradas por boa parte da população brasileira com a implantação de redes descentralizadas, como mitigação às dificuldades enfrentadas na implantação de politicas públicas para a universalização do acesso à Internet.

O objetivo principal deste trabalho é desenvolver uma arquitetura de software capaz de suportar a *Timelink*, uma rede descentralizada que oferece consistência eventual e replicação eficiente para o compartilhamento de conteúdos e serviços. A arquitetura proposta deve ser flexível e escalável, permitindo sincronização de dados de forma descen-

tralizada e confiável.

Para validar a arquitetura proposta, o trabalho também tem como objetivo implementar um protótipo funcional da *Timelink*, adaptado a um cenário educacional com requisitos simplificados. Esse protótipo visa demonstrar a viabilidade da solução em um ambiente de replicação total, explorando o uso do IPFS (*Interplanetary File System*) (BENET, 2014) para comunicação e sincronização. A implementação tem como foco testar a arquitetura em um contexto real, avaliando seu desempenho, consistência eventual e facilidade de uso em uma aplicação educacional prática.

O presente trabalho está organizado em seis capítulos, abordando desde os fundamentos teóricos até a implementação prática do protótipo da *Timelink*.

No Capítulo 2 são apresentados os conceitos essenciais para o desenvolvimento da *Timelink*, incluindo *Content Delivery Networks* (CDNs), Redes *Peer-to-Peer* (P2P), Sistemas de Arquivos Distribuídos, IPFS e o próprio conceito de *Timelink*. Esses temas fornecem a base teórica necessária para compreender as decisões arquiteturais e as tecnologias utilizadas na implementação do protótipo.

No Capítulo 3 é apresentada uma análise crítica da literatura relacionada à replicação e consistência em redes P2P, abordando as principais abordagens e desafios. São discutidos trabalhos que influenciaram as escolhas de design da *Timelink*.

No Capítulo 4 é apresentada a estrutura modular da *Timelink*, organizada em camadas hierárquicas para facilitar a manutenção e a escalabilidade. São descritas as funções e interações das quatro camadas principais: Comunicação, Dados, Serviços e Interface com o Usuário, detalhando como se comunicam entre si para fornecer replicação eficiente, sincronização eventual e consistência eventual.

No Capítulo 5 são explicadas as adaptações feitas na arquitetura para viabilizar o protótipo funcional, incluindo a simplificação das premissas e o uso do IPFS para comunicação e replicação. São descritas as tecnologias utilizadas e as limitações identificadas, como a ausência de resolução de conflitos e a dependência do nó de *bootstrap*.

No Capítulo 6 são apresentadas as conclusões do trabalho. Destacam-se as contribuições da arquitetura modular para a escalabilidade e flexibilidade da rede, além de validar a viabilidade prática por meio do protótipo implementado com requisitos simpli-

ficados. Também são discutidas as limitações identificadas e apresentadas propostas para trabalhos futuros, como a implementação de replicação seletiva, resolução automatizada de conflitos e exploração de alternativas ao IPFS. Por fim, o capítulo sugere possibilidades de expansão do uso da *Timelink* para outros contextos de compartilhamento de conteúdo.

2 Fundamentação Teórica

Neste capítulo, apresentam-se alguns conceitos fundamentais que embasam este trabalho e são essenciais para a sua compreensão.

Inicialmente, são discutidas as *Content Delivery Networks* (CDNs), que utilizam uma rede de servidores geograficamente distribuídos para otimizar a entrega de dados, proporcionando maior desempenho, escalabilidade e confiabilidade na comunicação entre usuários e serviços. Em seguida, explora-se o modelo descentralizado das redes *peer-to-peer* (P2P), onde cada nó atua simultaneamente como cliente e servidor, promovendo uma distribuição colaborativa dos recursos e fortalecendo a resiliência da rede.

O capítulo também aborda os Sistemas de Arquivos Distribuídos, que permitem o armazenamento e acesso a dados em ambientes com múltiplos nós interconectados, garantindo transparência, tolerância a falhas e segurança no gerenciamento das informações. Também é apresentado o IPFS, um protocolo que emprega o endereçamento baseado em conteúdo para assegurar a integridade e a eficiência na recuperação dos dados.

Por fim, a discussão se volta para o conceito de *Timelink*, que introduz a ideia da rede do tempo. A *Timelink* propõe a substituição da tradicional linha do tempo linear por uma rede de possibilidades interligadas, permitindo uma navegação não linear e uma experiência interativa na programação televisiva.

2.1 *Content Delivery Network*

Uma *Content Delivery Network* (CDN), ou Rede de Entrega de Conteúdo, é uma infraestrutura composta por um conjunto de servidores distribuídos geograficamente que colaboram para fornecer conteúdo a usuários finais de forma rápida, eficiente e confiável. Esses servidores são estrategicamente posicionados em diferentes localidades para se aproximarem geograficamente dos usuários, minimizando a latência e melhorando a experiência de navegação. O conteúdo, como arquivos de mídia, páginas *web* e aplicativos, é replicado nesses servidores, permitindo que os usuários acessem os dados a partir do servidor mais

próximo, o que reduz o tempo de carregamento e aumenta a eficiência na entrega de informações. (LI, 2008)

Entre as principais vantagens do uso de uma CDN estão:

- **Desempenho:** A proximidade entre servidores e clientes reduz o tempo de comunicação e a latência, proporcionando carregamento mais rápido das páginas e uma experiência de usuário aprimorada.
- **Disponibilidade e Confiabilidade:** Como o conteúdo é replicado em vários servidores distribuídos, a queda de um servidor não compromete a disponibilidade dos serviços, uma vez que outros servidores na rede podem fornecer os mesmos dados, aumentando a resiliência contra falhas.
- **Escalabilidade:** CDNs são projetadas para lidar com picos de tráfego, distribuindo as solicitações entre vários servidores e evitando sobrecarga no servidor de origem.
- **Segurança:** Algumas CDNs oferecem recursos de proteção contra ataques DDoS, mitigando tráfego malicioso antes de atingir o servidor de origem.

Essa arquitetura distribuída é especialmente útil para fornecer conteúdos estáticos (como imagens, vídeos e arquivos CSS/JavaScript) e conteúdos dinâmicos em tempo real, como transmissões ao vivo e aplicativos interativos. Grandes empresas de tecnologia, como Netflix, Google e Amazon, utilizam CDNs para fornecer seus conteúdos de forma eficiente para milhões de usuários em todo o mundo.

2.2 Redes *peer-to-peer*

Redes *peer-to-peer* (P2P), traduzidas como redes par-a-par, são uma arquitetura de rede descentralizada em que todos os nós têm funcionalidades equivalentes, atuando simultaneamente como clientes e servidores. Diferente do modelo tradicional cliente-servidor, não há um servidor centralizado que controle o fluxo de dados; ao invés disso, cada nó na rede compartilha diretamente seus recursos — como largura de banda, poder de processamento e armazenamento — com outros nós. (LI, 2008)

Essa comunicação bilateral permite que os nós se conectem dinamicamente, trocando informações e dados sem a necessidade de intermediários centralizados. Dessa forma, os recursos são distribuídos por toda a rede, possibilitando compartilhamento de arquivos, computação distribuída e transmissão de mídia em tempo real de forma eficiente e escalável.

Entre as principais características e vantagens das redes P2P estão:

- **Resiliência e Tolerância a Falhas:** A ausência de um ponto central de controle torna a rede altamente resiliente, pois a saída de um nó não compromete o funcionamento do restante da rede. Os dados e recursos são replicados em múltiplos nós, o que aumenta a tolerância a falhas.

- **Escalabilidade Dinâmica:** A entrada de novos nós aumenta automaticamente a capacidade computacional e a capacidade de armazenamento da rede, pois cada nó contribui com seus recursos. Assim, quanto maior o número de participantes, maior a eficiência na distribuição de dados.

- **Eficiência na Utilização de Recursos:** Os recursos são utilizados de forma colaborativa e distribuída, evitando a sobrecarga em um único servidor e maximizando o uso da largura de banda disponível em toda a rede.

- **Descentralização e Autonomia:** Como não há um servidor central, as redes P2P oferecem maior autonomia e controle aos usuários, aumentando a privacidade e dificultando a censura.

No entanto, as redes P2P também apresentam desafios, como segurança (devido à ausência de controle centralizado) e gerenciamento de tráfego (já que o fluxo de dados é distribuído de maneira dinâmica e imprevisível). Ainda assim, sua flexibilidade e robustez tornam essa arquitetura ideal para aplicativos como compartilhamento de arquivos, streaming de vídeo e computação distribuída.

2.3 Sistema de Arquivos Distribuídos

Sistemas de Arquivos Distribuídos são arquiteturas que permitem o armazenamento e o acesso a dados por meio de múltiplos nós interconectados, oferecendo aos usuários uma visão unificada dos arquivos, como se estivessem em um único sistema local, independentemente da localização física dos dados. Essa abordagem é fundamental para ambientes de larga escala e para aplicações que demandam alta disponibilidade e tolerância a falhas.

Essa arquitetura consiste em um conjunto de serviços e protocolos que possibilitam a distribuição, replicação e gerenciamento de arquivos entre servidores e clientes, de modo que o armazenamento dos dados não se restringe a um único dispositivo, mas se estende por toda a rede. Os usuários interagem com o sistema por meio de interfaces que abstraem as complexidades da localização, replicação e sincronização dos arquivos, proporcionando uma experiência semelhante à de um sistema de arquivos local. (TANENBAUM; STEEN, 2007)

Entre as principais características estão:

- **Transparência:** Abstraem a complexidade inerente à distribuição dos dados. Essa transparência pode se manifestar em diferentes aspectos, como localização (os usuários não precisam saber onde os dados estão armazenados), acesso (a forma de acessar os dados é uniforme, independentemente do nó onde eles residem) e replicação (as cópias dos dados são gerenciadas automaticamente para garantir disponibilidade).

- **Escalabilidade:** A arquitetura distribuída permite que o sistema cresça horizontalmente, adicionando novos nós conforme a demanda aumenta, sem que isso comprometa significativamente o desempenho.

- **Tolerância a Falhas e Alta Disponibilidade**: Por meio de mecanismos de replicação e redundância, os sistemas de arquivos distribuidos garantem que a falha de um ou mais nós não comprometa a integridade ou o acesso aos dados.

- **Consistência:** Garantir que todos os nós visualizem a mesma versão dos dados, mesmo diante de atualizações concorrentes, é um desafio central.

- **Segurança:** Incorporar mecanismos de autenticação, controle de acesso e criptogra-

fia, para proteger os dados contra acessos não autorizados e garantir a privacidade das informações.

Apesar de suas vantagens, a implementação de um sistema de arquivos distribuídos enfrenta diversos desafios:

- **Gerenciamento de Concorrência e Consistência**: A coordenação de múltiplos acessos simultâneos e a garantia de que todas as réplicas dos dados se mantenham consistentes é um dos maiores desafios, especialmente em ambientes com alta latência e grande número de nós
- **Latência de Rede:** A comunicação entre nós distribuídos geograficamente pode introduzir atrasos, afetando o desempenho do sistema. Estratégias como caching e algoritmos de replicação eficiente são necessárias para mitigar esses efeitos.
- **Segurança e Privacidade:** Em um ambiente distribuído, garantir a segurança dos dados e proteger contra ataques, como interceptação e adulteração, é uma preocupação constante, exigindo soluções robustas de criptografia e mecanismos de autenticação.

Em resumo, os Sistemas de Arquivos Distribuídos oferecem uma infraestrutura poderosa para o gerenciamento de dados em larga escala, combinando transparência, escalabilidade, tolerância a falhas e segurança. No entanto, a implementação desses sistemas requer a superação de desafios técnicos significativos, especialmente no que diz respeito à consistência, latência e integração de ambientes heterogêneos.

2.4 IPFS

O *InterPlanetary File System* (IPFS) [2] é um protocolo de armazenamento e um software de compartilhamento de arquivos que visa criar um sistema de arquivos distribuído, descentralizado e de alta eficiência. Desenvolvido para substituir o modelo tradicional de endereçamento baseado em localização, o IPFS utiliza um esquema de endereçamento baseado em conteúdo, no qual as informações são armazenadas e recuperadas através de

[2] ⟨https://ipfs.tech/⟩

uma *hash* criptográfica associada ao conteúdo do arquivo, e não à sua localização em um servidor específico. (BENET, 2014)

No IPFS, cada arquivo recebe um identificador exclusivo chamado de *Content Identifier* (CID), gerado por meio de um algoritmo de *hash* criptográfico (SHA-256 (IPFS Docs, 2025a)). Esse identificador é imutável e único para cada conteúdo, garantindo a integridade dos dados e evitando duplicações. Para fazer o download de um arquivo, o usuário consulta o CID na rede IPFS, que localiza e entrega o conteúdo a partir de nós que possuem uma cópia daquele arquivo. Dessa forma, não é necessário conhecer previamente a localização exata do arquivo na rede, pois a descoberta de conteúdo é realizada por meio do CID.

Apesar de utilizar um modelo de endereçamento baseado em conteúdo, o IPFS também oferece o serviço *InterPlanetary Name System* (IPNS), que possibilita a associação de CIDs a chaves criptográficas públicas. Esse serviço fornece endereços mutáveis, permitindo a atualização de conteúdo sem alterar o CID original. Com isso, o IPNS possibilita a criação de links permanentes que apontam para versões atualizadas dos arquivos, funcionando de forma semelhante a um sistema de DNS descentralizado. (IPFS Docs, 2025b)

Uma das principais características do IPFS é sua descentralização, eliminando a necessidade de servidores confiáveis ou pontos únicos de falha. A arquitetura do IPFS é inspirada em redes *peer-to-peer* (P2P), na qual cada nó armazena e compartilha partes dos arquivos com outros nós na rede, de forma colaborativa e distribuída. Isso resulta em um sistema altamente resiliente, escalável e tolerante a falhas, onde a entrada ou saída de nós não compromete o acesso ao conteúdo, uma vez que as informações são replicadas e distribuídas por toda a rede.

2.5 *Timelink*

O avanço da TV digital e a convergência com a internet estão revolucionando a forma como o conteúdo audiovisual é produzido, transmitido e consumido. Nesse cenário, a tradicional linha do tempo utilizada na programação televisiva – baseada em um fluxo linear e inflexível – está se mostrando inadequada para atender às novas demandas de

interatividade e personalização dos telespectadores. Estes, cada vez mais habituados às experiências digitais proporcionadas por *smartphones*, *tablets* e serviços de streaming, desejam não apenas escolher o que assistir, mas também quando e como consumir o conteúdo.

Foi nesse contexto que Teixeira (2018) desenvolveu o conceito de *Timelink*, uma inovação no design e na arquitetura da informação a partir de redes de difusão como a TV digital terrestre. A *Timelink* propõe a substituição da *linha do tempo* pela *rede do tempo*, rompendo com o paradigma tradicional de programação linear em fluxo. Ao transformar a cronologia rígida em uma rede de possibilidades, a *Timelink* permite que o telespectador navegue pela programação de forma não linear, escolhendo não apenas o conteúdo, mas também a sequência e o ritmo de consumo. Isso proporciona uma experiência interativa e personalizada, alinhada com as expectativas contemporâneas de interatividade, instantaneidade e participação coletiva.

A aplicação da *Timelink* visa atender a dois desafios principais na evolução de redes de difsuão, incluindo a TV digital:

- **Flexibilização da grade horária:** Adaptando-se ao *tempo de leitura* e ao *tempo de ação* de cada telespectador, que são subjetivos e variam conforme suas preferências individuais.

- **Inclusão digital e social:** Alinhando-se a um dos principais objetivos do Sistema Brasileiro de TV Digital Terrestre (SBTVD-T), que é minimizar a exclusão digital ao oferecer uma experiência interativa acessível e inclusiva para todos os perfis de público.

O conceito rompe com o modelo de transmissão em fluxo, transformando a experiência televisiva de algo passivo para uma navegação ativa, onde o usuário tem o controle sobre o tempo e o conteúdo consumido. Esse novo paradigma, inspirado na lógica das redes digitais, não apenas moderniza a forma de assistir à TV, como também cria novas oportunidades para narrativas interativas, publicidade segmentada e estratégias de engajamento.

2.5.1 Definição e Conceito

O conceito da *Timelink* foi desenvolvido como resposta às limitações do modelo linear de programação televisiva, que segue um fluxo rígido e inflexível, obrigando o telespectador a se adequar ao tempo definido pelas emissoras. Diferente desse paradigma, a *Timelink* propõe uma rede temporal que permite ao usuário:

- Escolher o que assistir, quando assistir e como consumir o conteúdo, de acordo com suas preferências e tempo disponível.
- Navegar não linearmente pela programação, acessando conteúdos de forma ramificada, semelhante à forma como navegamos na *web*.
- Interagir com a narrativa audiovisual, podendo selecionar desfechos alternativos, acessar conteúdos complementares ou explorar diferentes pontos de vista em uma mesma história.

O conceito se baseia na metáfora da rede, que substitui a visão tradicional do tempo como uma linha contínua (a *seta do tempo*) por uma malha interconectada de possibilidades. Assim, o tempo não é mais percebido como uma sequência rígida de eventos, mas como um conjunto de nós temporais conectados que podem ser navegados em diferentes direções.

2.5.2 Arquitetura

Na *Timelink*, o tempo é organizado como uma rede de nós interligados, onde cada nó representa um evento ou um segmento de conteúdo audiovisual. Esses nós podem ser navegados de forma ramificada, permitindo que o usuário escolha diferentes trajetórias de consumo.

Estrutura dos Nós Temporais

Cada nó temporal pode conter:

- **Conteúdo principal:** O evento audiovisual em si, como uma cena de um filme, um capítulo de uma série ou um segmento de um programa de TV.

- **Conteúdos complementares:** Informações adicionais, *making-of*, entrevistas ou até mesmo conteúdo gerado por outros usuários.
- **Interatividade contextual:** Opções para o usuário interagir com o conteúdo, como escolher o ponto de vista de um personagem ou acessar uma linha narrativa alternativa.
- **Conexões com outros nós:** Links para eventos passados, futuros ou paralelos, permitindo uma navegação não linear pela narrativa.

Organização da Rede Temporal

A organização da rede na *Timelink* pode seguir diferentes padrões, dependendo do objetivo da narrativa e da experiência do usuário:

- **Rede Linear Expandida:** Mantém a sequência narrativa principal, mas permite acessar conteúdos complementares sem interromper o fluxo principal.
- **Rede Não Linear:** Oferece múltiplos caminhos narrativos, permitindo que o usuário escolha a ordem dos eventos.
- **Rede Interativa Dinâmica:** Se adapta em tempo real às escolhas do usuário, personalizando a experiência narrativa.

Essa estrutura flexível rompe com a linearidade do tempo na programação televisiva tradicional, permitindo que o telespectador controle o ritmo e a ordem de consumo.

Desafios Técnicos na Implementação

A implementação da *Timelink* exige uma infraestrutura tecnológica avançada para suportar a flexibilidade temporal e a interatividade que caracterizam essa abordagem. Diferente do modelo linear tradicional de programação televisiva, o Timelink organiza o tempo como uma rede de nós interligados, permitindo que o usuário navegue não linearmente pela programação, acessando diferentes conteúdos em sequência personalizada e ritmo próprio.

Para viabilizar essa experiência, é necessário enfrentar três desafios técnicos fundamentais:

- **Armazenamento de Conteúdo:** A organização não linear e a necessidade de conteúdos complementares e ramificados exigem um modelo de armazenamento flexível e escalável.

- **Transmissão de Dados:** A distribuição de informações na rede do tempo requer um fluxo dinâmico e adaptativo de dados, ajustando-se às escolhas do usuário em tempo real.

- **Recuperação de Informações:** A navegação não linear demanda algoritmos avançados para indexação, busca e recuperação de conteúdos, garantindo uma experiência fluida e contínua.

3 Revisão Bibliográfica

Neste capítulo, são apresentados trabalhos relacionados à replicação e consistência de arquivos em redes P2P, um tema essencial para garantir escalabilidade, desempenho e disponibilidade em sistemas distribuídos. Esses estudos exploram diferentes estratégias de replicação, buscando distribuir a carga entre os nós da rede de forma eficiente e adaptativa, o que contribui para a escalabilidade da solução.

Além disso, os trabalhos abordam mecanismos de manutenção de consistência das réplicas, essenciais para assegurar que todos os nós tenham uma visão coerente e atualizada dos arquivos, mesmo em ambientes altamente dinâmicos e descentralizados. Para atingir esses objetivos, são propostas abordagens que minimizam a sobrecarga de comunicação, otimizam o uso de recursos e garantem consistência eventual ou consistência forte, conforme as necessidades da aplicação.

Cada uma das abordagens apresentadas neste capítulo adota estratégias específicas para equilibrar desempenho e consistência, utilizando técnicas como replicação ativa, reconciliação de conflitos e propagação de operações assíncronas.

Shen (2009) apresenta o IRM (*Integrated File Replication and Consistency Maintenance mechanism*), um mecanismo que integra replicação de arquivos e manutenção de consistência em sistemas P2P. Diferente de abordagens passivas, o IRM utiliza uma estratégia ativa de replicação, na qual nós que frequentemente requisitam um arquivo, ou que participam intensamente no tráfego desse arquivo, tendem a armazenar uma cópia local para melhorar o desempenho e reduzir a latência.

Para manter a consistência das réplicas, o IRM adota um modelo de validação centralizada, elegendo um proprietário para cada arquivo replicado. Esse nó é responsável por verificar se as cópias estão atualizadas e coordenar as atualizações quando necessário. Contudo, essa abordagem apresenta desafios, como a determinação da frequência de verificações para minimizar o atraso na propagação das atualizações e a prevenção da sobrecarga no nó dono, que pode se tornar um gargalo de desempenho caso seja excessivamente requisitado.

Os resultados experimentais indicam que o IRM é eficaz na redução da comunicação redundante, pois somente as réplicas desatualizadas são notificadas, economizando largura de banda. Além disso, o mecanismo se mostrou escalável e eficiente para o desenvolvimento de redes P2P de larga escala, apresentando um desempenho superior em comparação com métodos tradicionais de replicação e consistência. Dessa forma, o IRM se posiciona como uma solução promissora para a otimização de redes P2P, especialmente em cenários de alta demanda por consistência e disponibilidade de dados.

Martins et al. (2006) apresenta uma técnica de reconciliação de conflitos de atualização baseada na semântica da aplicação, aplicada à *Atlas Peer-to-Peer Architecture* (APPA). Essa arquitetura adota um modelo em camadas, no qual as camadas inferiores fornecem serviços genéricos e abstratos para as camadas superiores, enquanto a camada mais alta lida diretamente com a semântica da aplicação. Essa abordagem permite um gerenciamento contextualizado dos dados, adaptando-se às necessidades específicas de cada aplicação.

O APPA é projetado para um cenário colaborativo em que os participantes compartilham interesses em comum e recursos, além de participarem ativamente da rede por meio de ações repetitivas. Nesse modelo, cada nó possui autonomia na aplicação de políticas de compartilhamento e uso de recursos, o que promove uma colaboração distribuída e descentralizada.

A política de reconciliação é definida pelo usuário, permitindo um alto grau de personalização para atender aos requisitos específicos da aplicação. Essa política é então aplicada de forma distribuída por todos os nós na rede, utilizando um algoritmo dinâmico que organiza grupos de nós e grupos de réplicas com o objetivo de minimizar a comunicação e alcançar um estado consistente entre as réplicas. Esse modelo de agrupamento reduz a necessidade de sincronização global, permitindo que o consenso seja atingido de forma eficiente e escalável.

O algoritmo de reconciliação utilizado no APPA garante consistência eventual entre as réplicas, o que é ideal para aplicações que não exigem atualização imediata dos dados entre os nós. Os resultados apresentados indicam que a técnica é eficaz em manter réplicas consistentes em diferentes nós, mesmo em cenários de alta distribuição geográfica

e colaboração intensiva. Além disso, o modelo é capaz de adaptar-se dinamicamente às mudanças na topologia da rede e na disponibilidade dos nós, mantendo a eficiência na comunicação e a consistência dos dados.

Essa abordagem torna o APPA uma arquitetura promissora para redes P2P colaborativas, especialmente em aplicações que exigem distribuição dinâmica de dados, consistência eventual e personalização na reconciliação de conflitos.

Cart e Ferrie (2007) propõe um sistema de reconciliação assíncrona que permite a propagação de operações e garante a ordem consistente dessas operações em qualquer momento e entre quaisquer nós da rede. Esse modelo é particularmente útil em ambientes distribuídos e colaborativos, onde não há um componente central confiável e a comunicação entre os nós é arbitrária e dinâmica.

Para manter a ordem das operações, o sistema utiliza um histórico de operações realizado sobre as cópias distribuídas. Esse histórico é mantido por meio de um esquema de nomeação única, no qual cada nó e cada cópia recebem um identificador exclusivo. As informações de ordenação são incorporadas diretamente nos nomes, permitindo que as operações sejam serializadas de forma determinística. Isso garante que a ordem de execução das operações não altere o estado final do sistema, independentemente da sequência de propagação entre os nós.

O modelo proposto oferece uma alta flexibilidade de comunicação e é totalmente descentralizado, o que o torna adequado para redes colaborativas em que a topologia muda dinamicamente e os nós podem entrar ou sair do sistema a qualquer momento. Além disso, como não há dependência de um componente central para coordenar as atualizações, o sistema é resiliente a falhas e altamente escalável.

No entanto, um dos desafios desse sistema é a necessidade de preservar o histórico de operações para garantir a consistência. Esse requisito pode levar a uma sobrecarga de informações obsoletas, especialmente em redes colaborativas de longo prazo, onde o volume de operações aumenta continuamente. Para mitigar esse problema, estratégias de compactação e descarte de históricos podem ser necessárias, embora isso exija um balanceamento cuidadoso para não comprometer a consistência eventual do sistema.

Os resultados experimentais demonstram que o sistema é eficiente na reconciliação

de dados e mantém a consistência em ambientes altamente distribuídos. Dessa forma, o modelo é especialmente adequado para aplicações colaborativas que exigem consistência eventual, flexibilidade de comunicação e resiliência a falhas.

4 Arquitetura

Este capítulo apresenta a estrutura arquitetural da Timelink, detalhando suas principais camadas e módulos internos. Inicialmente, descreve-se a divisão entre front-end e back-end, ressaltando como essa separação contribui para a flexibilidade e manutenção do sistema. Em seguida, são apresentados os principais componentes de cada camada, destacando suas funções e interações para garantir comunicação eficiente, replicação de dados e consistência eventual.

Essa abordagem modular e hierárquica permite que a Timelink se adapte a diferentes cenários de replicação e sincronização, mantendo uma experiência de usuário interativa e responsiva. Além disso, a organização em camadas facilita a manutenção e evolução do sistema, permitindo a adição de novas funcionalidades sem impactar as demais partes da arquitetura.

Uma arquitetura para a *Timelink* deve atender a requisitos para garantir seu desempenho, robustez e flexibilidade em um ambiente distribuído. Para isso, é essencial que o sistema seja projetado com os seguintes objetivos:

- **Escalabilidade:** A arquitetura deve permitir que o aumento no número de nós resulte em um incremento proporcional da capacidade de armazenamento e processamento da rede. Isso assegura que o sistema possa acompanhar o crescimento da demanda sem comprometer o desempenho. A escalabilidade deve ser alcançada de maneira eficiente, evitando gargalos e mantendo a alta disponibilidade de dados.

- **Consistência Eventual:** Devido à natureza interativa da *Timelink* e à possibilidade de entrada e saída dinâmica de nós, podem surgir inconsistências temporárias entre as sub-redes. O sistema deve ser capaz de detectar e resolver conflitos de informações, convergindo gradualmente para um estado consistente. Para isso, é necessário implementar políticas de reconciliação que garantam a consistência eventual, sem comprometer a interatividade e a responsividade da aplicação.

- **Descentralização:** A arquitetura deve ser totalmente descentralizada, permitindo

que todos os nós operem de forma autônoma e garantindo que não existam pontos únicos de falha. Isso implica que não devem existir nós hierarquicamente superiores, assegurando um modelo igualitário e distribuído de compartilhamento e recuperação de dados. Essa abordagem não só aumenta a resiliência do sistema como também melhora sua robustez e tolerância a falhas.

- **Replicação Eficiente:** Como não é viável que cada nó armazene todo o conteúdo da rede, a arquitetura deve implementar uma replicação seletiva e eficiente. Para isso, é necessário que os nós conheçam quais conteúdos estão disponíveis na rede e como obtê-los. Além disso, é fundamental estabelecer um controle inteligente de redundância, para garantir alta disponibilidade sem desperdiçar recursos de armazenamento e largura de banda.

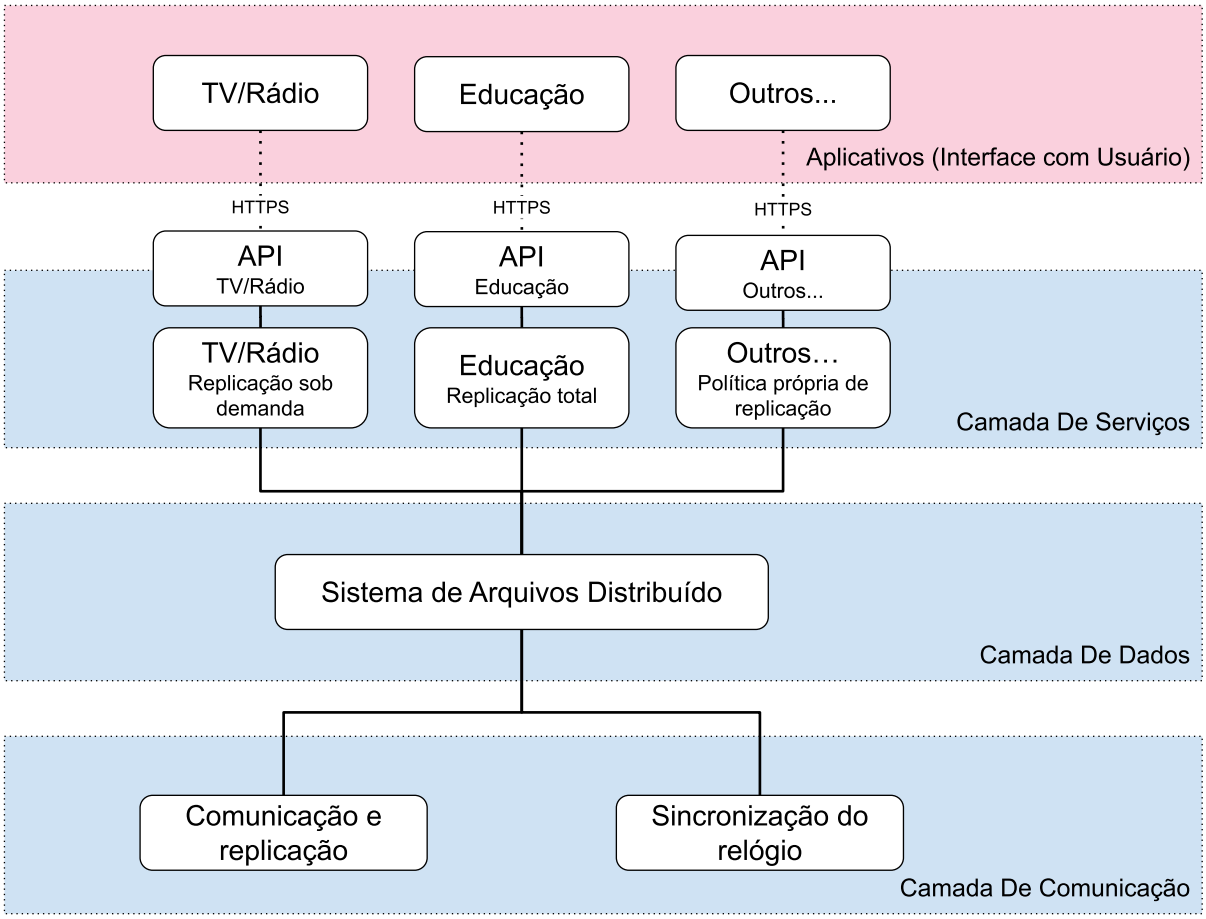

Figura 4.1: Arquitetura

O *software* da *Timelink* é organizado em uma arquitetura modular, dividida em *front-end* e *back-end*, conforme ilustrado na figura 4.1. Esse modelo visa proporcionar

flexibilidade, manutenção simplificada e escalabilidade, separando as responsabilidades de apresentação e interação do usuário da lógica de processamento e gerenciamento de dados.

No *front-end*, as interfaces são personalizadas para cada tipo de aplicação, garantindo uma experiência de usuário adaptada às necessidades específicas. O funcionamento do *front-end* depende apenas da conexão com a nuvem, o que permite flexibilidade na escolha dos dispositivos e uma navegação responsiva. Essa abordagem facilita a expansão para novas plataformas sem impactar o funcionamento interno do sistema.

O *back-end* é responsável por gerenciar as conexões entre os nós e replicar o conteúdo de forma eficiente e escalável. Para isso, ele é estruturado em camadas de abstração, cada uma com responsabilidades específicas que se alinham aos objetivos arquiteturais do Timelink. Essa organização em camadas permite um alto grau de modularidade, facilitando a manutenção e a evolução do sistema ao longo do tempo.

As camadas são organizadas conforme seus níveis de abstração e responsabilidade:

- Camadas inferiores fornecem serviços genéricos e abstratos, funcionando como fundamentos reutilizáveis para as camadas superiores. Elas são neutras em relação ao conteúdo, pois não têm conhecimento semântico dos dados.

- Camadas superiores contêm particularidades específicas de cada aplicação, pois estão mais próximas dos requisitos funcionais. Por estarem mais próximas do front-end, essas camadas interpretam o conteúdo e aplicam políticas específicas de replicação e consistência, adaptando-se às necessidades de cada aplicação.

Cada camada é subdividida em módulos, que são unidades independentes responsáveis por implementar um subconjunto de regras e funcionalidades. Essa divisão é feita para:

- **Isolar responsabilidades:** Cada módulo executa uma tarefa específica, mantendo alta coesão e baixo acoplamento com outros módulos.

- **Facilitar a manutenção e evolução:** Módulos podem ser atualizados ou substituídos sem afetar outras partes do sistema, desde que respeitem as interfaces definidas para comunicação.

Módulos de uma mesma camada não se comunicam entre si, pois possuem responsabilidades distintas. No entanto, um módulo pode consumir ou prover recursos para módulos de camadas diferentes, desde que mantenha compatibilidade de interface e respeite as hierarquias de abstração.

Uma regra fundamental na arquitetura é que camadas inferiores não utilizam serviços oferecidos por camadas superiores. Essa restrição visa:

- **Manter a neutralidade e reutilização:** Camadas inferiores são mais genéricas e independentes, o que permite que sejam reutilizadas em diferentes contextos sem adaptações.
- **Garantir o baixo acoplamento:** Ao evitar dependências cíclicas, a arquitetura reduz a complexidade e facilita a manutenção.

Essa hierarquia é fundamental para a flexibilidade da *Timelink*, permitindo que as camadas superiores adicionem novas funcionalidades ou políticas personalizadas sem impactar o funcionamento das camadas mais abstratas.

A seguir, segue o detalhamento dos objetivos, funcionalidades e limitações de cada uma das camadas do modelo.

4.1 Camada de Comunicação

A Camada de Comunicação é a camada mais inferior do modelo arquitetural da *Timelink*, sendo totalmente agnóstica ao conteúdo dos arquivos. Seu principal objetivo é estabelecer e manter a conectividade entre os nós da rede, garantindo que a troca de dados ocorra de maneira eficiente e confiável. Essa camada não possui conhecimento semântico sobre os dados transmitidos, o que a torna altamente reutilizável e independente das aplicações que utilizam a rede.

Por ser a base da arquitetura, a Camada de Comunicação é responsável por:

- Encontrar e identificar nós na rede
- Estabelecer conexões entre os nós

- Garantir a comunicação bidirecional, possibilitando tanto o envio quanto o recebimento de dados

- Fornecer serviços de comunicação genéricos, utilizados por camadas superiores para a replicação e sincronização de arquivos

Essa camada é composta por dois módulos: Comunicação e Replicação e Sincronização do Relógio.

4.1.1 Comunicação e replicação

O módulo de Comunicação e Replicação é responsável por gerenciar a troca de dados entre os nós e coordenar a replicação de arquivos de forma eficiente e escalável. As principais funções deste módulo incluem:

- Identificação de Nós Conectados: Manter uma lista atualizada da topologia da rede, informando às camadas superiores quais nós estão disponíveis para comunicação e replicação.

- Transferência de Arquivos: Realizar o download e upload de arquivos entre os nós. Esse processo deve ser assíncrono, para não impactar a responsividade do sistema.

Esse módulo não possui conhecimento semântico sobre o conteúdo dos arquivos, o que permite que ele seja utilizado para qualquer tipo de dado, independentemente do formato ou aplicação.

4.1.2 Sincronização do Relógio

O módulo de Sincronização do Relógio é responsável por manter os relógios dos nós sincronizados, garantindo um referencial de tempo consistente para todas as operações realizadas na rede. Isso é crucial para a Camada de Dados, pois as informações temporais são utilizadas para discriminar conteúdos mais recentes de versões antigas, garantindo a consistência eventual.

4.2 Camada de Dados

A Camada de Dados é responsável por abstrair os conceitos de baixo nível fornecidos pela Camada de Comunicação, oferecendo um sistema de gerenciamento de arquivos distribuído que serve como base para a Camada de Serviços. Essa camada proporciona uma interface transparente para o armazenamento, recuperação, replicação e sincronização de arquivos, permitindo que a Camada de Serviços manipule os dados sem precisar se preocupar com detalhes de comunicação e replicação.

4.2.1 Sistema de Arquivos Distribuído

O Sistema de Arquivos Distribuído é responsável por identificar e localizar os arquivos na rede, garantindo que todos os nós compartilhem a mesma estrutura de diretórios. Esse módulo permite que a Camada de Serviços explore o sistema de arquivos distribuído como se estivesse acessando um sistema de arquivos local, ocultando a complexidade da distribuição dos dados.

Entre as funções deste módulo, estão:

- Estrutura Uniforme de Diretórios: Garantir que todos os nós na rede tenham acesso à mesma estrutura hierárquica de diretórios e possam navegar pelos diretórios e arquivos de maneira uniforme e coerente.
- Descoberta Dinâmica: Implementar um algoritmo de descoberta dinâmica de conteúdo, permitindo que os nós anunciem e descubram novos arquivos conforme são criados ou atualizados.

Esse módulo é responsável por manter a consistência dos arquivos distribuídos, utilizando uma abordagem de consistência eventual. Para isso, é preciso gerenciar versões de arquivos e sincronizar atualizações entre diferentes nós, garantindo que a versão mais recente esteja sempre disponível na rede.

O mecanismo de versionamento funciona "marcando" cada arquivo com um *timestamp* que indica o instante da última alteração. Dessa forma, sempre que um nó solicita o acesso a um arquivo, o sistema assegura que a versão mais atual seja fornecida.

Entretanto, essa estratégia pode ocasionar problemas em cenários nos quais duas sub-redes perdem a comunicação, gerando divergências entre as versões dos arquivos. Como o módulo de Sistema de Arquivos Distribuído não interpreta o conteúdo dos arquivos, não é possível realizar um *merge* automático entre versões conflitantes. Por essa razão, a responsabilidade pela resolução de conflitos é delegada à Camada de Serviços.

Quando um nó detecta que a versão de um arquivo em seu armazenamento está desatualizada em relação àquela disponível em outro nó, o módulo de Sistema de Arquivos Distribuído dispara um *trigger* para notificar a Camada de Serviços. Essa notificação inclui as duas versões do arquivo para que, com base em uma política predefinida, a Camada de Serviços possa comparar os conteúdos. A partir dessa comparação, a Camada de Serviços decide se mantém a versão existente ou se gera um novo arquivo resultante da fusão (*merge*) das versões conflitantes. Caso seja efetuado o *merge*, o arquivo resultante receberá um novo *timestamp*, refletindo a atualização, e será replicado para toda a rede, garantindo assim a convergência para a consistência eventual.

4.3 Camada de Serviços

A Camada de Serviços situa-se na fronteira entre o *back-end* e o *front-end*, sendo a responsável por adaptar as funcionalidades de replicação e sincronização de dados às necessidades específicas de cada aplicação. Em outras palavras, esta camada contém um módulo de implementação que é customizado para cada serviço oferecido, de modo que os requisitos de replicação e consistência estejam alinhados com as expectativas de desempenho e usabilidade do front-end.

A Camada de Serviços desempenha três funções principais: estabelecer uma política de resolução de conflitos, implementar a lógica de negócios da aplicação e fornecer uma interface de comunicação segura com o *front-end*. Essas funções permitem que a camada adapte as políticas de replicação e sincronização conforme as demandas específicas de cada aplicação, oferecendo flexibilidade e escalabilidade na implementação de novos serviços.

- **Resolução de Conflitos:** Em conjunto com a Camada de Dados, a Camada de

Serviços é responsável por interpretar o conteúdo dos arquivos e, quando necessário, realizar o merge entre diferentes versões conflitantes. Essa resolução é baseada em regras específicas da aplicação, garantindo a consistência eventual do sistema.

- **Lógica de Negócios:** Implementa os recursos e serviços próprios de cada aplicação, definindo as regras de negócio e as funcionalidades que serão utilizadas pelos usuários. Essa função permite que a Camada de Serviços ofereça experiências personalizadas, adaptando-se às demandas específicas de cada aplicação sem impactar as demais camadas da arquitetura.

- **Interface com o *Front-end*:** Fornece uma interface de comunicação robusta e segura com o front-end, utilizando protocolos seguros (como HTTPS) para garantir a integridade e a confidencialidade dos dados. Essa interface abstrai a complexidade do back-end, oferecendo uma API intuitiva e eficiente para o desenvolvimento de interfaces responsivas e interativas.

4.4 Aplicativos

A Interface com o Usuário é a camada mais próxima do usuário final, responsável por estabelecer a comunicação visual e interativa com os participantes da rede. Essa camada é composta por aplicações que permitem aos usuários acessar, navegar e interagir com os conteúdos oferecidos pela Timelink.

Cada serviço oferecido pela Timelink possui uma implementação própria de aplicação, projetada para atender aos requisitos específicos da experiência do usuário. Essas implementações são altamente personalizadas, adaptando-se às características funcionais e visuais de cada serviço.

A Interface com o Usuário é desenvolvida para se comunicar diretamente com a Camada de Serviços, utilizando APIs seguras para acessar dados e funcionalidades. Essa comunicação é feita de maneira transparente e eficiente, garantindo que as interações do usuário sejam refletidas na rede.

5 Implementação

Durante o desenvolvimento deste trabalho, surgiu a oportunidade de colaborar em um projeto conjunto com a prefeitura de Lima Duarte, cujo objetivo era implementar uma versão adaptada da Timelink voltada para a educação. O aplicativo desenvolvido tinha como propósito facilitar a interação entre professores, alunos e responsáveis, oferecendo funcionalidades como:

- Disponibilização de materiais didáticos pelos professores.
- Envio de tarefas e respostas por parte dos alunos.
- Feedback e atribuição de notas pelos professores.
- Acompanhamento das atividades escolares pelos responsáveis.

Por conta do curto prazo para implementação e por se tratar do desenvolvimento de um protótipo, a arquitetura adotada partiu de premissas simplificadas, assumindo que:

- Todos os nós estariam constantemente disponíveis na rede, eliminando a necessidade de lidar com nós desconectados ou indisponibilidade temporária.
- Todo o conteúdo seria replicado integralmente entre os nós, garantindo alta disponibilidade e consistência uniforme.

Essas simplificações reduziram significativamente a complexidade da implementação, pois:

- Não foi necessário tratar cenários de inconsistências ou conflitos de versões, uma vez que a replicação integral garantiu que todas as cópias fossem idênticas.
- Não houve necessidade de controle de redundância, já que todos os arquivos estariam disponíveis em todos os nós, independentemente da demanda.

Ao eliminar a necessidade de gerenciamento de inconsistências e reduzir a complexidade da replicação, foi possível construir um protótipo viável e funcional. Os módulos da implementação foram organizados conforme ilustrado na Figura 5.2, mostrando as adaptações feitas em relação à arquitetura original da Timelink. Essas adaptações permitiram que o projeto educacional fosse concluído, oferecendo uma experiência de usuário simplificada e eficiente.

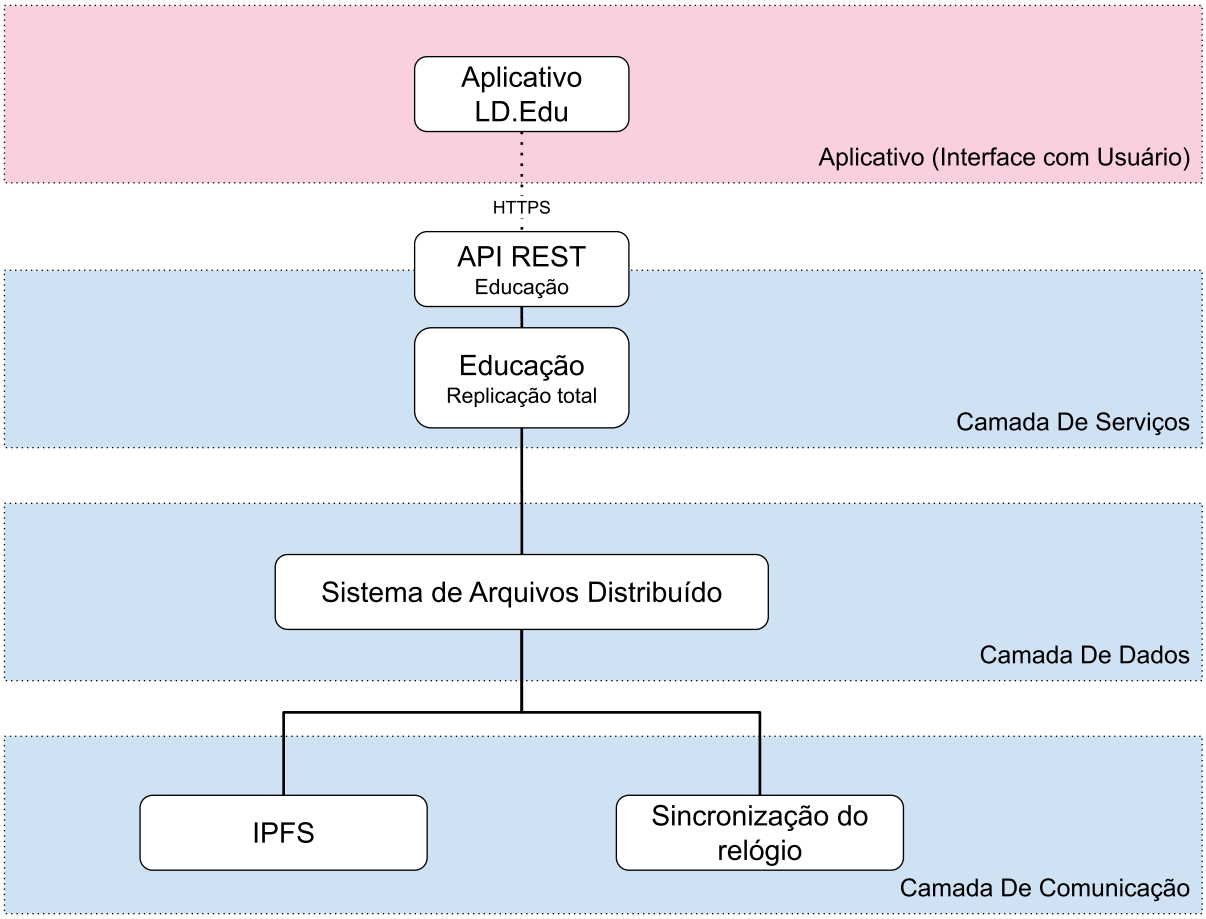

Figura 5.1: Modelo de Implementação

Entretanto, como não há uma infraestrutura dedicada para uma rede própria da *Timelink*, a comunicação entre os nós ocorre por meio da conexão com a Internet. No entanto, devido à má qualidade do acesso à Internet nas escolas, os nós estão sujeitos a conexões instáveis, podendo enfrentar lentidão ou até mesmo interrupções na conexão.

Essas limitações podem resultar em atualizações concorrentes, pois nós desconectados podem modificar os mesmos arquivos de maneira independente. Como não foi implementado um algoritmo de resolução de conflitos, esse cenário pode levar a perda de atualizações ou inconsistências nos dados replicados. Em situações mais críticas, as

inconsistências não tratadas podem comprometer o funcionamento do sistema.

É importante destacar que a adoção de premissas simplificadas se restringe à implementação realizada neste trabalho e não à arquitetura proposta. A implementação desenvolvida representa um subconjunto funcional da arquitetura da Timelink, e a introdução de novos requisitos, como resolução de conflitos, controle de redundância e replicação seletiva, pode ser feita de forma incremental aproveitando a estrutura já implementada.

Ou seja, a implementação atual não invalida ou restringe a arquitetura geral, mas sim serve como um primeiro estágio funcional, sobre o qual funcionalidades mais avançadas podem ser adicionadas. Assim, a expansão da implementação não exige uma reestruturação completa, mas sim a incorporação de novos algoritmos e mecanismos que complementam o que já foi desenvolvido. Dessa forma, o escopo reduzido da implementação não compromete a viabilidade da arquitetura proposta, apenas define um ponto inicial sobre o qual evoluções futuras podem ser construídas.

5.1 Topologia Piloto

Para atender a esses requisitos em um escopo reduzido, a rede *Timelink* foi implantada como um experimento piloto com apenas quatro nós, distribuídos da seguinte forma:

- Três nós localizados nas escolas, acessíveis pelos usuários (professores, alunos e responsáveis).
- Um nó público com endereço IP fixo, atuando como nó de *bootstrap*, permitindo que os demais nós se conectem e sincronizem com a rede.

Cada nó foi implementado utilizando um Raspberry Pi, responsável por armazenar e distribuir o conteúdo educacional. Esses dispositivos se comunicam entre si via Internet, garantindo a replicação dos dados entre os nós da rede.

No entanto, a conexão com a Internet nas escolas é limitada e precária, sujeita a indisponibilidades e lentidão frequentes. Em diversos momentos, a rede enfrenta instabilidades que interrompem a comunicação entre os nós, impedindo a sincronização imediata

do conteúdo. Como consequência, quando a conexão falha, os nós funcionam de forma isolada, armazenando e distribuindo apenas os dados disponíveis localmente até que a conectividade seja restabelecida.

A adoção de um modelo de consistência eventual permite que, assim que a conexão for retomada, mesmo que com baixa velocidade, os nós consigam gradualmente sincronizar as atualizações pendentes. Dessa forma, o sistema se ajusta dinamicamente às condições da rede, garantindo que todas as alterações realizadas durante o período de desconexão sejam eventualmente propagadas e incorporadas pelos demais nós.

Além disso, cada nó localizado nas escolas atua como um ponto de acesso Wi-Fi, criando uma rede local própria. Dessa forma, os usuários podem se conectar diretamente ao nó mais próximo, acessando o conteúdo educacional mesmo sem conexão com a Internet. Essa abordagem permite que os alunos e professores utilizem o sistema normalmente dentro do ambiente escolar, reduzindo a dependência de infraestrutura externa.

A Figura 5.2 ilustra o funcionamento da rede descentralizada utilizada na implementação do *LD.Edu* dentro das escolas de Lima Duarte. Na imagem, é possível visualizar:

- A estrutura da escola, onde está localizado um dos nós da rede descentralizada.
- A área de acesso criada na escola, representando a rede Wi-Fi fornecida pelo nó local.
- Um usuário conectado à rede local, acessando o aplicativo LD.Edu sem necessidade de Internet.
- A comunicação do nó local com os demais nós da rede através da Internet, garantindo a replicação de conteúdo quando a conectividade está disponível.

5.2 Camada de Comunicação

Os módulos da Camada de Comunicação foram implementados utilizando ferramentas de software já existentes, com o objetivo de acelerar o desenvolvimento do protótipo e simplificar a integração dos nós na rede.

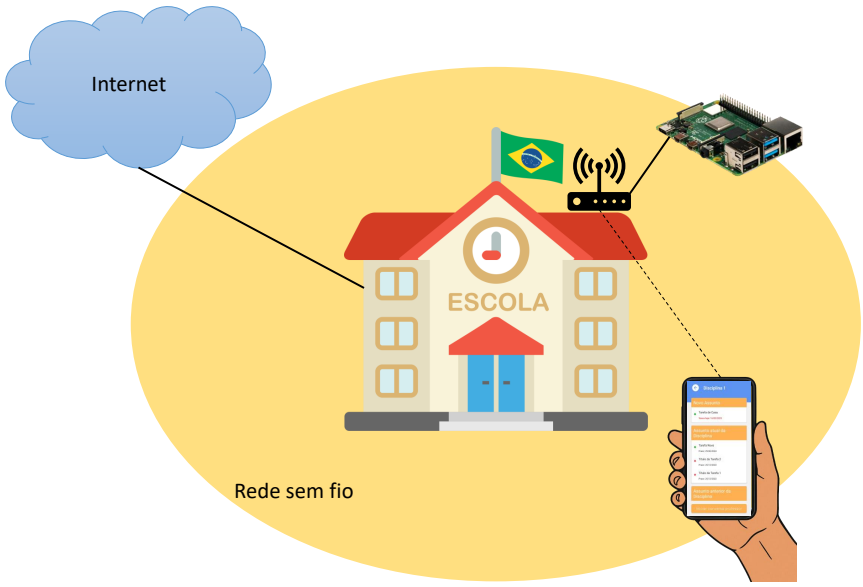

Figura 5.2: Aplicativo LD.edu conectado a Rede Descentralizada

Como os dispositivos físicos utilizados para suportar os nós eram *Raspberry Pis* conectados à Internet, assumiu-se que os relógios desses dispositivos estavam sincronizados e atualizados pelo Sistema Operacional, dispensando a necessidade de um módulo específico de sincronização de relógio.

O módulo de Comunicação e Replicação foi implementado utilizando o IPFS, uma tecnologia de armazenamento e compartilhamento de arquivos distribuída e descentralizada. O uso do IPFS foi fundamental para viabilizar a construção rápida do protótipo, proporcionando um meio eficiente de conectar e sincronizar os nós da rede.

5.2.1 Funcionalidades do IPFS na Implementação

O IPFS foi utilizado para encontrar e conectar os nós na rede, estabelecendo uma comunicação distribuída e descentralizada. Inicialmente, todos os dispositivos conectam-se ao nó de *bootstrap*, um nó público com endereço de IP fixo, que atua como ponto de partida para descoberta de nós.

- A partir do nó de *bootstrap*, os outros dispositivos se juntam à rede e conseguem encontrar todos os demais nós conectados.
- Uma vez estabelecida a conexão inicial, o nó de *bootstrap* não é mais necessário,

pois o IPFS mantém uma lista local dos dispositivos disponíveis, permitindo que os nós continuem se conectando diretamente.

O IPFS permite publicar conteúdos na rede e recuperá-los através de um CID (Content Identifier). Cada arquivo publicado recebe um CID único, gerado com base em um *hash* criptográfico do conteúdo.

- Novos arquivos podem ser disponibilizados na rede simplesmente publicando o CID correspondente.
- Atualizações de arquivos geram um novo CID, uma vez que o IPFS utiliza um modelo de conteúdo imutável, onde qualquer alteração cria uma nova versão do arquivo.
- O gerenciamento de versões e a identificação de atualizações são realizados pela Camada de Dados, utilizando os CIDs para comparar versões e verificar a necessidade de replicação.

Além do IPFS, foi utilizado o IPNS, um sistema de nomes descentralizado que permite associar um endereço fixo a um CID mutável.

- Cada nó publica um arquivo de metadados utilizando o IPNS, contendo informações sobre os arquivos disponíveis e suas versões.
- Esse arquivo de metadados é gerenciado pela Camada de Dados, permitindo que outros nós verifiquem as versões dos arquivos e sincronizem atualizações quando necessário.

5.2.2 Limitações do IPFS na Implementação

A utilização do IPFS foi importante para viabilizar a construção rápida do protótipo. No entanto, o IPFS apresentou limitações significativas em relação aos requisitos da arquitetura original.

Dependência de Nó de *Bootstrap*

Uma das principais limitações do IPFS é a necessidade de especificar explicitamente um nó de *bootstrap* para que um dispositivo possa encontrar e se conectar à rede. No IPFS, os nós não conseguem descobrir automaticamente outros dispositivos na rede sem o auxílio de um nó de *bootstrap*, que funciona como um ponto de partida para a descoberta de nós.

Essa dependência cria um ponto de falha na conexão, uma vez que:

- Se o nó de *bootstrap* ficar indisponível, novos nós não conseguirão se juntar à rede ou localizar dispositivos ativos, impactando a resiliência e a escalabilidade do sistema.
- Apesar de o IPFS permitir que os nós armazenem localmente as informações sobre os dispositivos previamente conectados, em cenários onde nenhum nó conhecido esteja disponível, a reconexão com a rede não será possível sem o nó de *bootstrap*.
- Em ambientes dinâmicos, onde nós entram e saem frequentemente da rede, a dependência de um nó central pode afetar a continuidade do serviço, comprometendo a disponibilidade e a experiência do usuário.

Essa limitação contraria um dos princípios de descentralização desejados na arquitetura da *Timelink*, exigindo que adaptações futuras busquem alternativas para descoberta de nós de maneira totalmente descentralizada.

É importante notar que embora o IPFS dependa de nós de *bootstrap* para descoberta inicial de pares, seria possível configurar dinamicamente esses nós antes da inicialização do sistema. Para isso, seria necessário utilizar algum algoritmo, protocolo ou serviço que permitisse encontrar nós ativos na rede e alimentar o IPFS com essas informações. Essa abordagem poderia mitigar a dependência de um conjunto fixo de nós de *bootstrap*, tornando a descoberta de pares mais flexível e resiliente. No entanto, essa solução não foi adotada na implementação realizada neste trabalho.

Operações Limitadas a *Download* e *Upload*

Outra limitação importante do IPFS é que suas operações de gerenciamento de dados são limitadas a *download* e *upload*. O modelo de endereçamento baseado em conteúdo

do IPFS é imutável, o que significa que qualquer alteração em um arquivo gera um novo CID, enquanto o CID antigo permanece inalterado.

Essa abordagem impõe algumas dificuldades na troca e atualização de metadados, pois:

- Não há um mecanismo nativo de atualização direta no IPFS, o que torna a modificação de metadados uma tarefa complexa e custosa. Sempre que um metadado é alterado, um novo CID é gerado, exigindo que a Camada de Dados atualize todas as referências anteriores para o novo identificador.

- A troca de metadados requer a publicação de um novo CID, o que implica em operações redundantes de upload e download. Além disso, a sincronização dessas atualizações entre todos os nós da rede aumenta o tráfego de comunicação e consome largura de banda, impactando a eficiência da rede.

- Para mitigar essa limitação, foi utilizado o IPNS para vincular um identificador fixo a um CID mutável, permitindo que os metadados fossem atualizados indiretamente. No entanto, o IPNS possui alta latência na propagação de atualizações, o que afeta o tempo de resposta da aplicação em cenários dinâmicos.

Essa limitação aumenta a complexidade do gerenciamento de versões e metadados, exigindo um esforço adicional na Camada de Dados para manter as referências atualizadas e sincronizadas entre os nós. Como resultado, a flexibilidade na atualização de conteúdos é limitada, o que pode impactar a experiência do usuário em aplicações que requerem alterações frequentes.

5.3 Camada de Dados

Na Camada de Dados foram implementadas operações comuns de um sistema de arquivos, incluindo:

- Listar arquivos e diretórios

- Escrever arquivos

- Ler arquivos
- Deletar arquivos

Essas operações são implementadas de forma transparente, permitindo que a Camada de Serviços manipule os arquivos como se estivesse interagindo com um sistema de arquivos local, ocultando os detalhes da replicação e sincronização.

Não foi implementada a operação de resolução de conflitos, uma vez que não era escopo deste projeto. Como resultado, não há políticas para reconciliar versões conflitantes quando diferentes nós atualizam o mesmo arquivo de forma independente.

Essa escolha foi feita com base na premissa de que o protótipo opera em um cenário de replicação total, em que todo o conteúdo é replicado integralmente em todos os nós, e todos os nós estão constantemente disponíveis na rede. Nesse contexto, não há atualizações concorrentes, pois não ocorrem partições na rede e todas as modificações se propagam rapidamente para os demais nós. Como não existem versões conflitantes, a necessidade de resolver conflitos é eliminada, o que simplifica a implementação e reduz a complexidade da Camada de Dados.

5.3.1 Metadados e Gerenciamento de Arquivos

Para fornecer as operações mencionadas, a Camada de Dados mantém uma lista de metadados em cada nó, contendo informações essenciais sobre os arquivos armazenados. Cada item da lista de metadados representa um arquivo, e as informações salvas para cada arquivo incluem:

- ***filename*:** Contém o nome do arquivo no sistema de arquivos, representado pelo caminho completo (*fullpath*) a partir da raiz do sistema de arquivos. Essa abordagem garante a unicidade dos nomes e permite navegação hierárquica na estrutura de diretórios distribuída.
- ***last_update*:** Armazena o timestamp da última atualização feita no nó. Esse campo é utilizado para comparar versões de arquivos ao sincronizar metadados com outros nós, garantindo que a versão mais recente seja utilizada.

- ***ipfsHash*:** Contém o CID do arquivo no IPFS, utilizado para recuperar o arquivo na rede. Como o IPFS utiliza endereçamento baseado em conteúdo, o CID é único para cada versão do arquivo, permitindo a identificação de alterações e controle de versões.

Para sincronizar os metadados entre os nós, a Camada de Dados utiliza o IPNS, que permite associar o CID mais atualizado ao endereço do nó. Dessa forma, outros nós podem consultar os metadados mais recentes de um nó sem precisar conhecer previamente o CID, uma vez que o IPNS vincula o endereço fixo do nó ao CID atual.

A sincronização de metadados é realizada ao consultar periodicamente os nós vizinhos, seguindo um fluxo de atualização definido para garantir a consistência eventual:

- **Inclusão de Novos Arquivos:** Se um *filename* presente nos metadados consultados não existe localmente, uma nova entrada é adicionada à lista de metadados local, indicando que o arquivo está disponível na rede.
- **Atualização de Arquivos:** Se o *filename* estiver presente em ambas as listas, o nó compara o campo *last_update*. Caso o *timestamp* do outro nó seja mais recente, a entrada local é atualizada com o CID mais recente, indicando que uma nova versão do arquivo foi publicada.
- **Manutenção de Arquivos Deletados:** Quando um arquivo é deletado, a entrada é mantida na lista de metadados, mas o campo *ipfsHash* é esvaziado, indicando que o arquivo não existe mais na rede.
- **Anúncio de Novas Versões:** Ao final do processo de sincronização, se os metadados locais foram atualizados, o nó anuncia uma nova versão utilizando o IPNS, permitindo que outros nós sincronizem as mudanças.

Essa abordagem garante que todas as versões dos arquivos sejam propagadas para os demais nós, mantendo a consistência eventual na rede. No entanto, como não há um mecanismo de resolução de conflitos, eventuais inconsistências podem ocorrer em cenários de atualizações concorrentes.

5.4 Camada de Serviços

A Camada de Serviços estabelece a política de resolução de conflitos, implementa as regras de negócio e gerencia a comunicação com o *front-end*.

No contexto desta prototipação, foi adotado um cenário de replicação total, assumindo-se, de forma simplificada, que todos os nós estão constantemente disponíveis na rede. Essa premissa elimina a possibilidade de partições na rede e minimiza a ocorrência de inconsistências, o que simplifica significativamente a gestão da consistência eventual.

Dessa forma, não foi necessário implementar uma política de resolução de conflitos, uma vez que:

- A consistência é rapidamente atingida, pois a disponibilidade constante dos nós permite que as atualizações se propaguem rapidamente por toda a rede.
- Conflitos de versões são inexistentes, pois não há atualizações concorrentes em nós isolados.
- Não houve necessidade de desenvolver algoritmos de merge para combinação semântica de arquivos incompatíveis, uma vez que não ocorrem divergências entre as versões.

Para permitir que múltiplos usuários se conectem simultaneamente a um nó, a comunicação entre o *back-end* e o *front-end* foi implementada utilizando uma API REST sobre HTTPS, garantindo segurança, escalabilidade e flexibilidade na interação com a Timelink.

A escolha de uma API REST foi motivada por:

- Simplicidade e flexibilidade na comunicação, permitindo que diferentes clientes (navegadores, aplicativos móveis e outras interfaces) acessem os mesmos serviços.
- Escalabilidade e facilidade de manutenção, permitindo que novos endpoints sejam adicionados sem afetar as operações existentes.

O uso de HTTPS foi essencial para garantir a segurança da comunicação entre o *front-end* e o *back-end*, criptografando os dados para proteger informações sensíveis como credenciais de login e dados escolares.

Na Camada de Serviços também foram implementadas as regras de negócio da aplicação, adaptando a Timelink ao contexto educacional. As funcionalidades desenvolvidas facilitam a comunicação entre professores, alunos e responsáveis, abrangendo:

- **Login baseado em perfis:** Foram criados três perfis de usuário (aluno, professor e responsável) com permissões específicas
- **Disponibilização de Material:** Professores podem publicar conteúdos educativos (como PDFs, vídeos e slides) na rede, permitindo que alunos acessem o material a qualquer momento.
- **Envio de Respostas às Tarefas:** Alunos podem enviar respostas, anexando arquivos.
- **Acompanhamento de Atividades:** Responsáveis podem acompanhar as atividades escolares.
- **Notas:** Professores fornecem feedback para as atividades entregues atribuindo notas.

Essas são algumas funcionalidades que foram desenvolvidas para atender às necessidades específicas do ambiente educacional, explorando as capacidades de replicação e sincronização da Timelink para fornecer uma experiência de usuário responsiva e colaborativa.

5.4.1 Exemplo de Sincronização na Rede

Para ilustrar o funcionamento do mecanismo de replicação e sincronização da *Timelink*, foi realizado um experimento envolvendo dois nós da rede:

- **Nó Local**: Criado para fins de teste, representando um nó descentralizado da rede.
- **Nó RePesq**: Nó que atua como *bootstrap* na rede implementada em Lima Duarte, permitindo a conexão inicial e a sincronização dos demais nós.

O cenário simula um processo de envio e replicação de arquivos dentro da rede, utilizando a seguinte operação:

- **Operação realizada**: Um aluno envia a resposta de uma atividade contendo um anexo de vídeo para o professor.
- **Arquivo**: Vídeo com 29,74 MB.
- **Tempo total de sincronização**: 123,087 segundos.

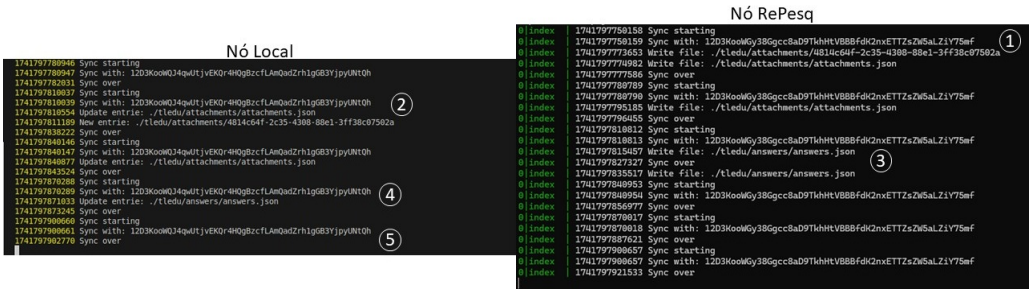

Figura 5.3: Logs Sincronização

A Figura 5.3 apresenta os logs registrados durante o processo de sincronização. Para facilitar a compreensão, os principais momentos do experimento foram numerados de 1 a 5, conforme descrito abaixo:

1. **Alterações iniciais no sistema de arquivos**: O Nó RePesq modifica dois arquivos localmente, registrando as alterações no sistema de arquivos.
2. **Identificação da atualização**: Durante uma rodada de verificação, o Nó Local detecta que um dos arquivos foi modificado e que um novo arquivo foi adicionado à rede.
3. **Novas modificações**: O Nó RePesq realiza novas alterações, modificando um terceiro arquivo e aplicando duas alterações consecutivas sobre ele.
4. **Reconhecimento das mudanças**: Em uma nova rodada de atualização, o Nó Local identifica as alterações realizadas pelo Nó RePesq, garantindo que as modificações sejam replicadas.
5. **Estado de consistência atingido**: Após a última verificação, o Nó Local não detecta mais nenhuma modificação pendente, indicando que a rede formada pelos dois nós alcançou um estado consistente.

5.5 Aplicativo

O aplicativo LD.Edu foi desenvolvido para a plataforma Android, com o objetivo de oferecer uma experiência de usuário intuitiva e transparente para alunos, professores e responsáveis, utilizando a Timelink como infraestrutura de replicação e sincronização de conteúdos educacionais.

Ao conectar-se à rede disponibilizada por um dos nós da *Timelink*, o usuário pode fazer *login* utilizando suas credenciais, acessando o conteúdo educacional de maneira simples e intuitiva. Todas as complexidades técnicas relacionadas à replicação de arquivos, sincronização de versões e comunicação distribuída são ocultadas do usuário, proporcionando uma experiência semelhante à de qualquer outro aplicativo educacional. Dessa forma, o usuário interage com o LD.Edu como se estivesse utilizando um sistema centralizado e tradicional, sem precisar se preocupar com detalhes de conectividade ou consistência de dados.

A Figura 5.4 apresenta algumas das telas desenvolvidas para o LD.Edu, incluindo a tela de login, a visão do perfil do usuário, a listagem das disciplinas disponíveis e a visualização detalhada de uma disciplina. Essas telas ilustram a interface simples projetada para facilitar a navegação dos usuários. O desenvolvimento do aplicativo foi realizado pelo Thomás Marques, garantindo uma implementação integrada à infraestrutura da rede descentralizada. Alguns fluxos de interação com o aplicativo são apresentados no Anexo B.

Para facilitar a divulgação e os testes do aplicativo, foi implementado um Modo de Demonstração, que pode ser ativado na tela de login ao marcar a opção "Habilitar modo de demonstração". Esse modo permite que usuários interessados explorem as funcionalidades do LD. Edu mesmo que não estejam conectados à rede distribuída da *Timelink*.

O Modo de Demonstração só foi possível graças ao nó utilizado para *bootstrap*, que é visível na internet e possui um endereço de IP fixo. Dessa forma, ao habilitar o modo de demonstração, o aplicativo se conecta a esse nó pela internet, estabelecendo uma ponte com a rede distribuída. Como o nó de *bootstrap* mantém uma lista atualizada de todos os dispositivos conectados, o aplicativo é capaz de acessar o conteúdo compartilhado na rede formada pelos quatro nós, simulando o funcionamento completo do sistema. Essa

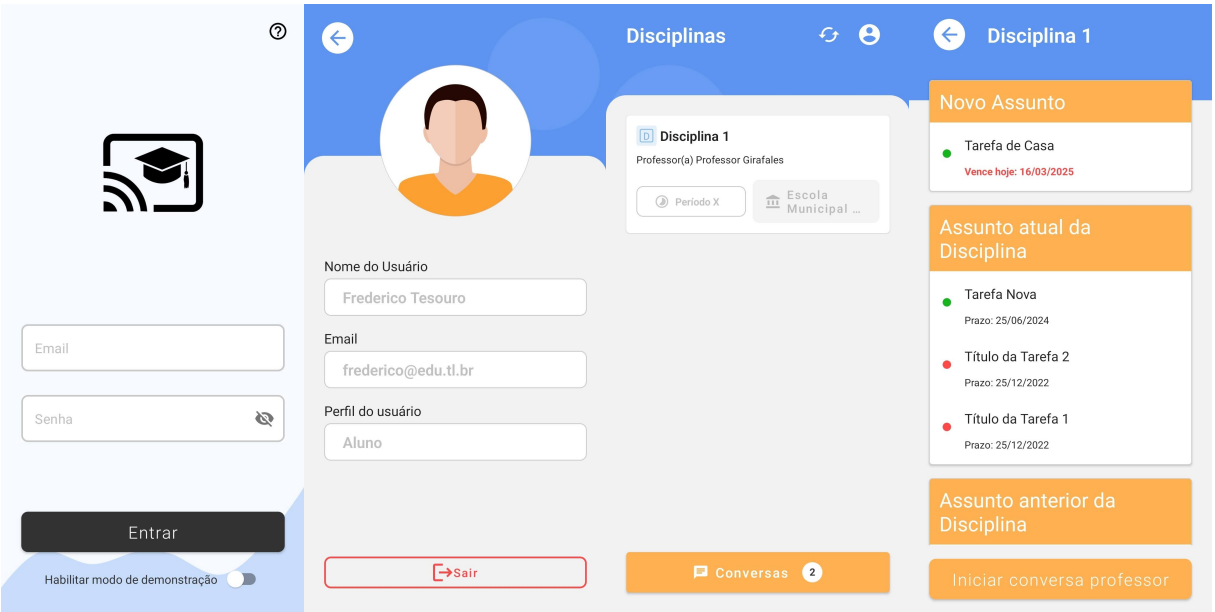

Figura 5.4: Interface do aplicativo LD.Edu: (a) Tela de login, (b) Visão do perfil do usuário, (c) Lista de disciplinas, (d) Visualização detalhada de uma disciplina.

abordagem permite que usuários explorem as funcionalidades do LD.Edu de forma realista e interativa, mesmo estando fora da rede local.

6 Considerações Finais

O principal objetivo deste trabalho foi propor um modelo de arquitetura para a *Timelink*, uma rede P2P descentralizada voltada para o compartilhamento eficiente de conteúdo. A arquitetura foi planejada para operar em um ambiente distribuído, sem hierarquia entre os nós, garantindo que todos os dispositivos tenham o mesmo nível de importância e possam funcionar de maneira independente. Além disso, a proposta visou criar uma estrutura escalável, flexível e confiável, capaz de alcançar consistência eventual e replicar conteúdo de forma eficiente em toda a rede.

Para alcançar esse objetivo, foi desenvolvida uma arquitetura organizada em camadas hierárquicas, proporcionando uma abstração modular que atende aos requisitos iniciais. Essa abordagem facilitou a separação de responsabilidades entre as camadas, permitindo que aspectos críticos como comunicação, sincronização de dados, replicação e resolução de conflitos fossem implementados de maneira flexível e independente. A organização modular também contribuiu para a escalabilidade da arquitetura, uma vez que novas funcionalidades podem ser incorporadas sem impactar as camadas existentes.

Além disso, foi possível construir um protótipo inicial ao flexibilizar alguns dos requisitos propostos, adotando premissas simplificadas como disponibilidade constante dos nós e replicação integral do conteúdo. Apesar dessas simplificações, o protótipo preservou os principais conceitos da arquitetura proposta, demonstrando a viabilidade prática da *Timelink*. O desenvolvimento do protótipo validou as ideias fundamentais da arquitetura e sinalizou a possibilidade de evoluir para um modelo mais completo, capaz de atender aos requisitos gerais da rede.

O trabalho desenvolvido contribui para o avanço na criação de redes P2P descentralizadas com consistência eventual e replicação eficiente. Além disso, os resultados obtidos sugerem que é possível implementar um modelo inteiramente baseado na arquitetura proposta, oferecendo flexibilidade e escalabilidade para diferentes cenários de compartilhamento de conteúdo.

Como proposta para trabalhos futuros, algumas melhorias e expansões podem ser

exploradas para aprimorar a arquitetura da *Timelink* e ampliar seu potencial de aplicação.

Uma das principais evoluções seria a implementação de replicação seletiva e eficiente, permitindo que cada nó armazene apenas os conteúdos que sejam relevantes para seus usuários. Essa abordagem reduziria o consumo de recursos, como espaço de armazenamento e largura de banda, além de aumentar a escalabilidade da rede, viabilizando seu uso em cenários com grandes volumes de dados.

Outra proposta é a implementação da resolução automatizada de conflitos, utilizando políticas customizadas para diferentes tipos de conteúdo. Essa funcionalidade seria essencial para manter a consistência eventual em cenários complexos, onde atualizações concorrentes ocorrem e nós podem ser desconectados temporariamente. A utilização de regras semânticas específicas para cada tipo de dado permitiria a reconciliação inteligente e eficiente.

Também é recomendada a exploração de alternativas ao IPFS, visando superar suas limitações quanto à dependência de nós de *bootstrap* e à imutabilidade do conteúdo. Entre as possibilidades estão a utilização de protocolos de descoberta de nós totalmente descentralizados, o uso de troca direta de mensagens entre nós e até mesmo o desenvolvimento de protocolos personalizados. Essas abordagens poderiam reduzir os problemas de alta latência na propagação do conteúdo e metadados.

Além das melhorias arquiteturais e técnicas, existe a oportunidade de expandir o uso da *Timelink* para outros contextos. A arquitetura modular é flexível e adaptável a diferentes cenários de compartilhamento de conteúdo, incluindo a captura de conteúdo televisivo e posterior disponibilização para consumo não linear. Esse caso de uso inovador exploraria todo o potencial da rede no tempo, permitindo uma experiência interativa e personalizada de consumo de mídia.

Bibliografia

BENET, J. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

CART, M.; FERRIE, J. Asynchronous reconciliation based on operational transformation for p2p collaborative environments. In: IEEE. *2007 International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2007)*. [S.l.], 2007. p. 127–138.

CRAIDE, S. *Apenas 22% dos brasileiros têm boas condições de conectividade*. 2024. Disponível em: ⟨https://agenciabrasil.ebc.com.br/economia/noticia/2024-04/apenas-22-dos-brasileiros-tem-boas-condicoes-de-conectividade⟩. Acesso em: 28 fev. 2025.

IBGE (Ed.). *Acesso à internet e à televisão e posse de telefone móvel celular para uso pessoal 2019*. IBGE, Coordenação de Trabalho e Rendimento, 2021. ISBN 9786587201566. Disponível em: ⟨https://biblioteca.ibge.gov.br/index.php/biblioteca-catalogo?view=detalhes&id=2101794⟩. Acesso em: 08 de junho 2021.

IPFS Docs. *Content Identifiers (CIDs)*. 2025. Disponível em: ⟨https://docs.ipfs.tech/concepts/content-addressing/#what-is-a-cid⟩. Acesso em: 25 fev. 2025.

IPFS Docs. *InterPlanetary Name System (IPNS)*. 2025. Disponível em: ⟨https://docs.ipfs.tech/concepts/ipns/⟩. Acesso em: 25 fev. 2025.

LI, J. On peer-to-peer (p2p) content delivery. *Peer-to-Peer Networking and Applications*, Springer, v. 1, n. 1, p. 45–63, 2008.

MARTINS, V. et al. Reconciliation in the appa p2p system. In: IEEE. *12th International Conference on Parallel and Distributed Systems-(ICPADS'06)*. [S.l.], 2006. v. 1, p. 10–pp.

MORENO, M. F. et al. R&D progress on TV 3.0 application coding layer. *SET International Journal of Broadcast Engineering*, v. 2023, n. 1, 2023. Disponível em: ⟨https://dx.doi.org/10.18580/setijbe.2023.1⟩.

SHEN, H. Irm: Integrated file replication and consistency maintenance in p2p systems. *IEEE transactions on Parallel and Distributed systems*, IEEE, v. 21, n. 1, p. 100–113, 2009.

TANENBAUM, A. S.; STEEN, M. V. Sistemas distribuídos: Princípios e paradigmas. In: *Sistemas Distribuídos: Princípios e Paradigmas*. 2. ed. São Paulo: Pearson Universidades, 2007. cap. 11, p. 296–328. ISBN 978-85-7605-142-8.

TEIXEIRA, S. C. *Timelink: um novo "tempo" para a TV digital aberta*. Tese (Doutorado) — Pontifícia Universidade Católica de São Paulo, 2018.

A API REST

Este anexo apresenta os *endpoints* da API REST desenvolvida para a implementação do LD.Edu. A API REST foi projetada para permitir a comunicação entre o *back-end* e o *front-end*, fornecendo serviços para autenticação de usuários, envio e recuperação de tarefas, troca de mensagens, entre outras funcionalidades.

A.0.1 /api/v0.1/user/login

Método	POST
Descrição	Autentica o usuário e retorna um token.
Parâmetros	**Corpo:** {*email, password*}
Resposta	**Sucesso (200):** {*email, name, role, _id, token*}
Erro	**401 Unauthorized** - Credenciais inválidas.

A.0.2 /api/v0.1/class/get_all

Método	GET
Descrição	Retorna todas as turmas disponíveis.
Autenticação	Sim
Parâmetros	Nenhum
Resposta	**Sucesso (200):** {*Lista com: chat_id, title, start_date, last_update_time, last_update_username*}

A.0.3 /api/v0.1/attachment/get_attachment/:attachment_id

Método	GET
Descrição	Retorna o arquivo solicitado.
Autenticação	Sim

Parâmetros	Nenhum
Resposta	**Sucesso (200):** {*Arquivo raw*}

A.0.4 /api/v0.1/assignment/get_answer/:assignment_id

Método	GET
Descrição	Retorna a resposta solicitada.
Autenticação	Sim
Parâmetros	Nenhum
Resposta	**Sucesso (200):** {*Lista com: user_id, assignement_id, submit_time, marked_as_done, attachment_id*}

A.0.5 /api/v0.1/assignment/mark_as_done/:assignment_id

Método	POST
Descrição	Marca o exercício como informado
Autenticação	Sim
Parâmetros	**Corpo:** {*submitTime, markAsDone*}
Resposta	**Sucesso (200):** {*Lista com: user_id, assignement_id, submit_time, marked_as_done*}

A.0.6 /api/v0.1/assignment/submit_attachment/:assignment_id

Método	POST
Descrição	Faz o upload do arquivo de resposta para a tarefa
Autenticação	Sim
Parâmetros	**Corpo:** {*attachmente_file, submit_time, mark_as_done*}
Resposta	**Sucesso (200):** {*Lista com: user_id, assignement_id, submit_time, marked_as_done, attachment_id*}

A.0.7 /api/v0.1/chat/get_chat_list

Método	GET
Descrição	Retorna todos as conversas e mensagens
Autenticação	Sim
Parâmetros	Nenhum
Resposta	**Sucesso (200):** {*Lista com: chatId, title, startDate, lastUpdateTime, lastUpdateUsername, messages*}

A.0.8 /api/v0.1/chat/get_chat_messages/:chat_id

Método	GET
Descrição	Retorna as mensagens de uma conversa
Autenticação	Sim
Parâmetros	Nenhum
Resposta	**Sucesso (200):** {*Lista com: message_id, content, sent_time, sent_by_username, readed_by_user*}

A.0.9 /api/v0.1/chat/mark_messages_as_read

Método	POST
Descrição	Marca as mensagens como lidas
Autenticação	Sim
Parâmetros	**Corpo:** {*Lista de mensagens*}
Resposta	**Sucesso (200)**

A.0.10 /api/v0.1/chat/send_message/:chat_id

Método	POST
Descrição	Envia uma mensagem em uma conversa
Autenticação	Sim

Parâmetros	**Corpo:** {*content, sentTime*}
Resposta	**Sucesso (200):** {*message_id, content, sent_time, sent_by_username, readed_by_user*}

A.0.11 /api/v0.1/chat/send_messages/:chat_id

Método	POST
Descrição	Envia uma lista de mensagens em uma conversa
Autenticação	Sim
Parâmetros	**Corpo:** {*Lista com: content, sent_time_client*}
Resposta	**Sucesso (200):** {*Lista com: message_id, content, sent_time, sent_by_username, readed_by_user*}

A.0.12 /api/v0.1/class/:class_id/get_students

Método	GET
Descrição	Retorna a lista de alunos de uma turma
Autenticação	Sim
Parâmetros	Nenhum
Resposta	**Sucesso (200):** {*Lista com: user_id, name, email, parent_id*}

A.0.13 /api/v0.1/class/:assignment_id/get_answers

Método	GET
Descrição	Retorna a lista de respostas para uma tarefa
Autenticação	Sim
Parâmetros	Nenhum

Resposta	**Sucesso (200):** {*Lista com: user_id, user_name, submit_time, marked_as_done, file_name, content_type, attachment_id, grade*}

A.0.14 /api/v0.1/class/:assignment_id/update_grade

Método	PATCH
Descrição	Atualiza a nota de uma resposta
Autenticação	Sim
Parâmetros	**Corpo:** {*user_id, grade*}
Resposta	**Sucesso (200):** {*user_id, user_name, submit_time, marked_as_done, file_name, content_type, attachment_id, grade*}

A.0.15 /api/v0.1/class/:assignment_id/update_grades

Método	PATCH
Descrição	Atualiza as nota de várias resposta
Autenticação	Sim
Parâmetros	**Corpo:** {*Lista com: user_id, grade*}
Resposta	**Sucesso (200):** {*Lista com: user_id, user_name, submit_time, marked_as_done, file_name, content_type, attachment_id, grade*}

A.0.16 /api/v0.1/class/:class_id/create_topic

Método	POST
Descrição	Cria um tópico
Autenticação	Sim
Parâmetros	**Corpo:** {*title, timestamp*}

Resposta	**Sucesso (200):** {*topic_id, title, timestamp, class_id*}

A.0.17 /api/v0.1/class/:class_id/create_topics

Método	POST
Descrição	Cria vários tópicos
Autenticação	Sim
Parâmetros	**Corpo:** {*Lista com: title, timestamp*}
Resposta	**Sucesso (200):** {*Lista com: topic_id, title, timestamp, class_id*}

A.0.18 /api/v0.1/class/:class_id/get_topics

Método	GET
Descrição	Retorna os tópicos de uma turma
Autenticação	Sim
Parâmetros	Nenhum
Resposta	**Sucesso (200):** {*Lista com: _id, title, timestamp*}

A.0.19 /api/v0.1/class/:topic_id/create_assignment

Método	POST
Descrição	Cria uma tarefa
Autenticação	Sim
Parâmetros	**Corpo:** {*title, deadline, description, timestamp*}
Resposta	**Sucesso (200):** {*assignment_id, title, deadline, description, timestamp, topic_id, attachments*}

A.0.20 /api/v0.1/class/:assignment_id/update_assignment

Método	PUT
Descrição	Atualiza uma tarefa e faz upload de arquivos
Autenticação	Sim
Parâmetros	**Corpo:** {*title, deadline, description, timestamp, topic_id, lista de _attachments: content, file_name*}
Resposta	**Sucesso (200):** {*assignment_id, title, deadline, description, timestamp, topic_id, attachments*}

A.0.21 /api/v0.1/class/:topic_id/get_assignments

Método	GET
Descrição	Retorna as tarefas
Autenticação	Sim
Parâmetros	Nenhum
Resposta	**Sucesso (200):** {Lista com: *assignment_id, title, deadline, description, timestamp, topic_id, attachments*}

A.0.22 /api/v0.1/chat/get_open_chat

Método	GET
Descrição	Retorna chat do tipo aberto
Autenticação	Sim
Parâmetros	Nenhum
Resposta	**Sucesso (200):** {*chat_id, title, start_date, user_list, messages*}

A.0.23 /api/v0.1/chat/:message_to_user_id/create_direct_chat

Método	POST
Descrição	Cria chat privado com outro usuário

Autenticação	Sim
Parâmetros	Nenhum
Resposta	**Sucesso (200):** {*chat_id, title, start_date, user_list, messages*}

A.0.24 /api/v0.1/chat/:user_id/get_direct_chat

Método	GET
Descrição	Retorna chat privado com outro usuário
Autenticação	Sim
Parâmetros	Nenhum
Resposta	**Sucesso (200):** {*chat_id, title, start_date, user_list, messages*}

A.0.25 /api/v0.1/chat/:class_id/create_class_chat

Método	POST
Descrição	Cria chat privado com uma turma
Autenticação	Sim
Parâmetros	Nenhum
Resposta	**Sucesso (200):** {*chat_id, title, start_date, user_list, messages*}

A.0.26 /api/v0.1/chat/:class_id/get_class_chat

Método	GET
Descrição	Retorna chat privado com uma turma
Autenticação	Sim
Parâmetros	Nenhum

Resposta	**Sucesso (200):** {*chat_id, title, start_date, user_list, messages*}

A.0.27 /api/v0.1/helloServer

Método	GET
Descrição	Testa conexão com o servidor
Autenticação	Sim
Parâmetros	Nenhum
Resposta	**Sucesso (200)**

B Fluxos do Aplicativo

Neste anexo são apresentados alguns dos fluxos de interação do usuário com o aplicativo LD.Edu. As imagens ilustram os passos seguidos pelos diferentes perfis de usuários — aluno, professor e responsável — ao utilizar o sistema.

B.1 Fluxo de Login – Perfil de Aluno

Na imagem B.1, podemos observar:

1. Tela inicial do aplicativo
2. O usuário digita as suas credenciais e seleciona o botão *Entrar*
3. Visão do aluno ao fazer login

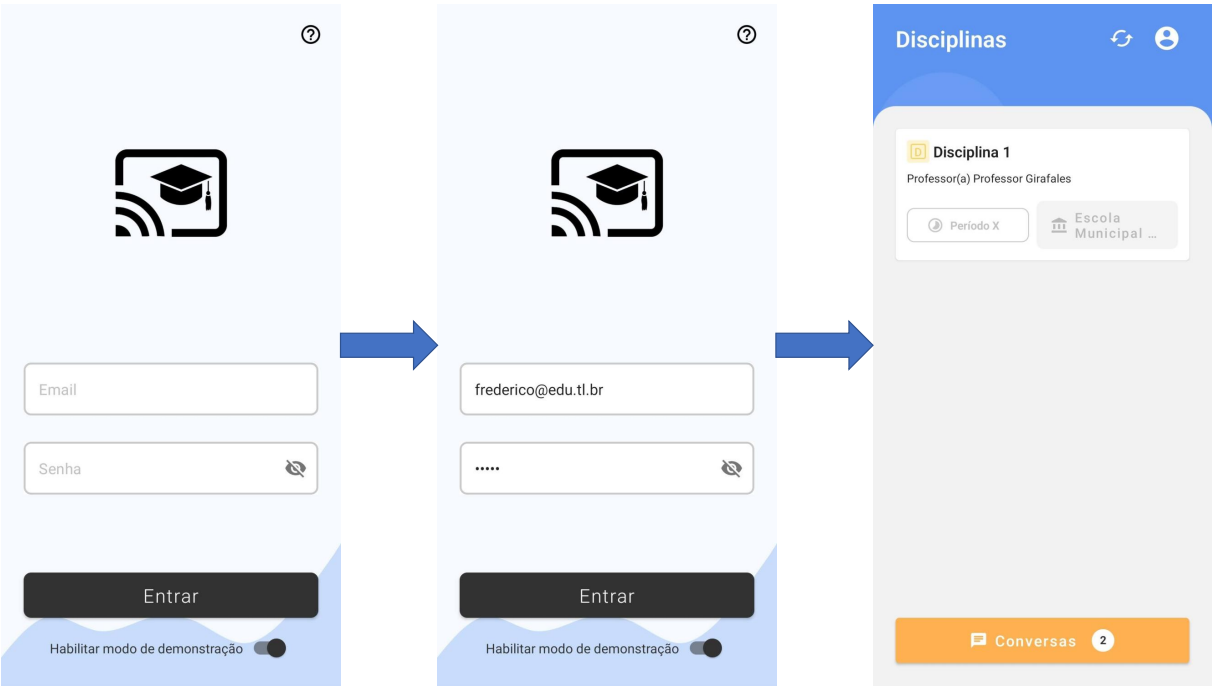

Figura B.1: Fluxo de Login – Perfil de Aluno

B.2 Aluno Inicia Conversa com Professor

Na imagem B.2, podemos observar:

1. Visão do aluno na página inicial
2. Ao selecionar *Disciplina 1*, o aluno consegue ver os assuntos e as disciplinas criadas pelo professor. Também vê o botão *Iniciar conversa professor* e selecioná-lo
3. Abre um *pop-up* para que se insira o título da conversa
4. O aluno preenche o título e seleciona *Confirmar*
5. É direcionado para uma conversa em branco com o professor e pode enviar mensagens

B.3 Professor Visualiza Conversa com Aluno

Na imagem B.3, podemos observar:

1. Tela inicial do aplicativo com as credenciais do professor
2. Visão das disciplinas ministradas pelo professor
3. Ao selecionar *Conversas*, o professor pode ver as conversas disponíveis
4. Ao selecionar uma conversa, é possível ver as mensagens com suas devidas informações

B.4 Professor Cria Assunto e Publica Tarefa

Na imagem B.4, podemos observar:

1. Visão de uma disciplina, com as tarefas já criadas e as opções de *Assuntos*, *Nova tarefa* e *Lista de Alunos*
2. Ao selecionar *Assuntos*, é mostrada uma tela com opções de criar novos assuntos
3. Ao selecionar *Adicionar assunto*, o usuário escreve o nome desejado e seleciona *Salvar*

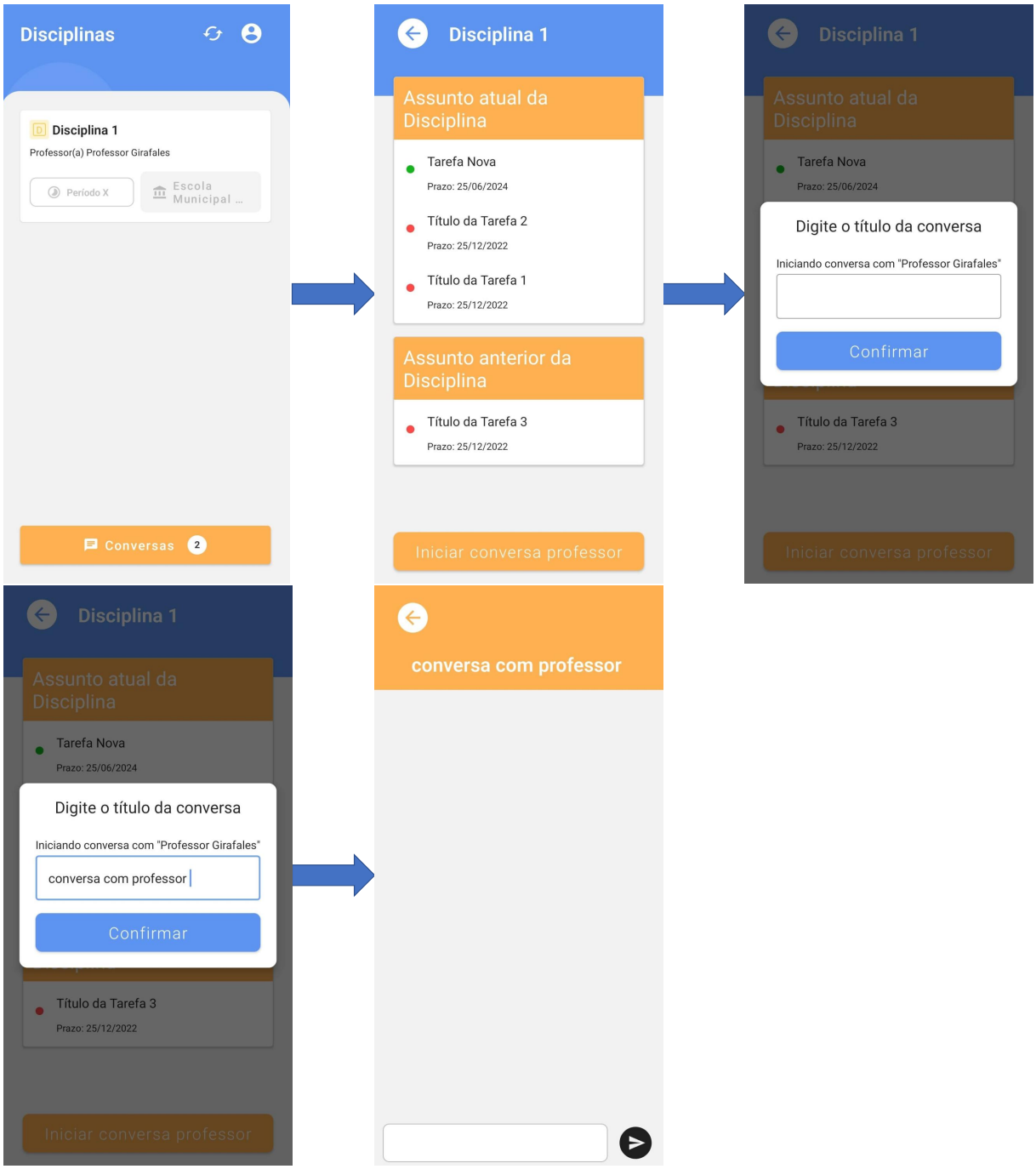

Figura B.2: Aluno Inicia Conversa com Professor

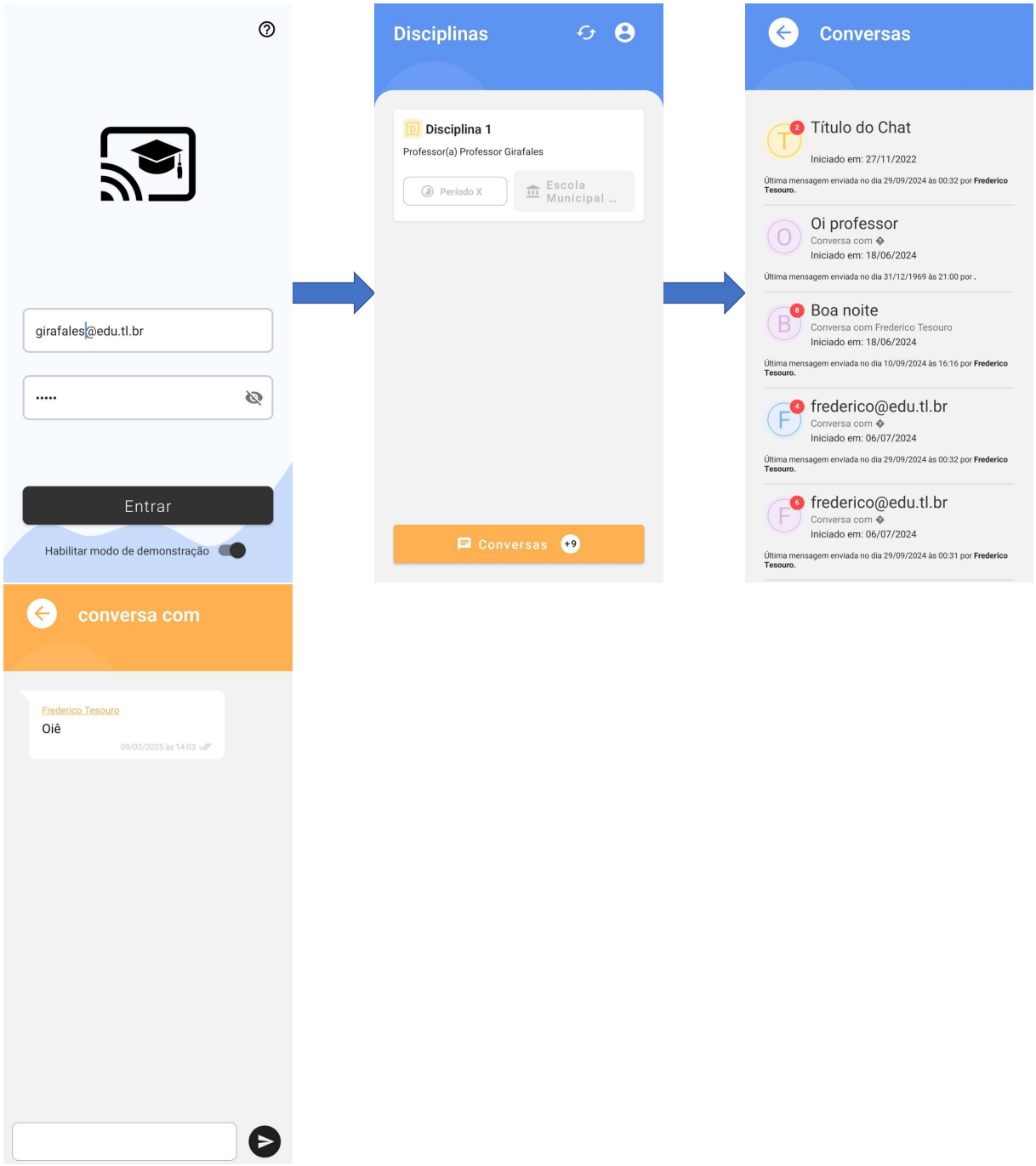

Figura B.3: Professor Visualiza Conversa com Aluno

4. De volta a visão da disciplina o usuário pode ver o assunto criado

5. Ao selecionar *Nova tarefa* o usuário é capaz de colocar as informações de uma nova tarefa e salvá-la

6. Após salvar, o usuário pode editar as informações e adicionar conteúdos anexos caso desejar

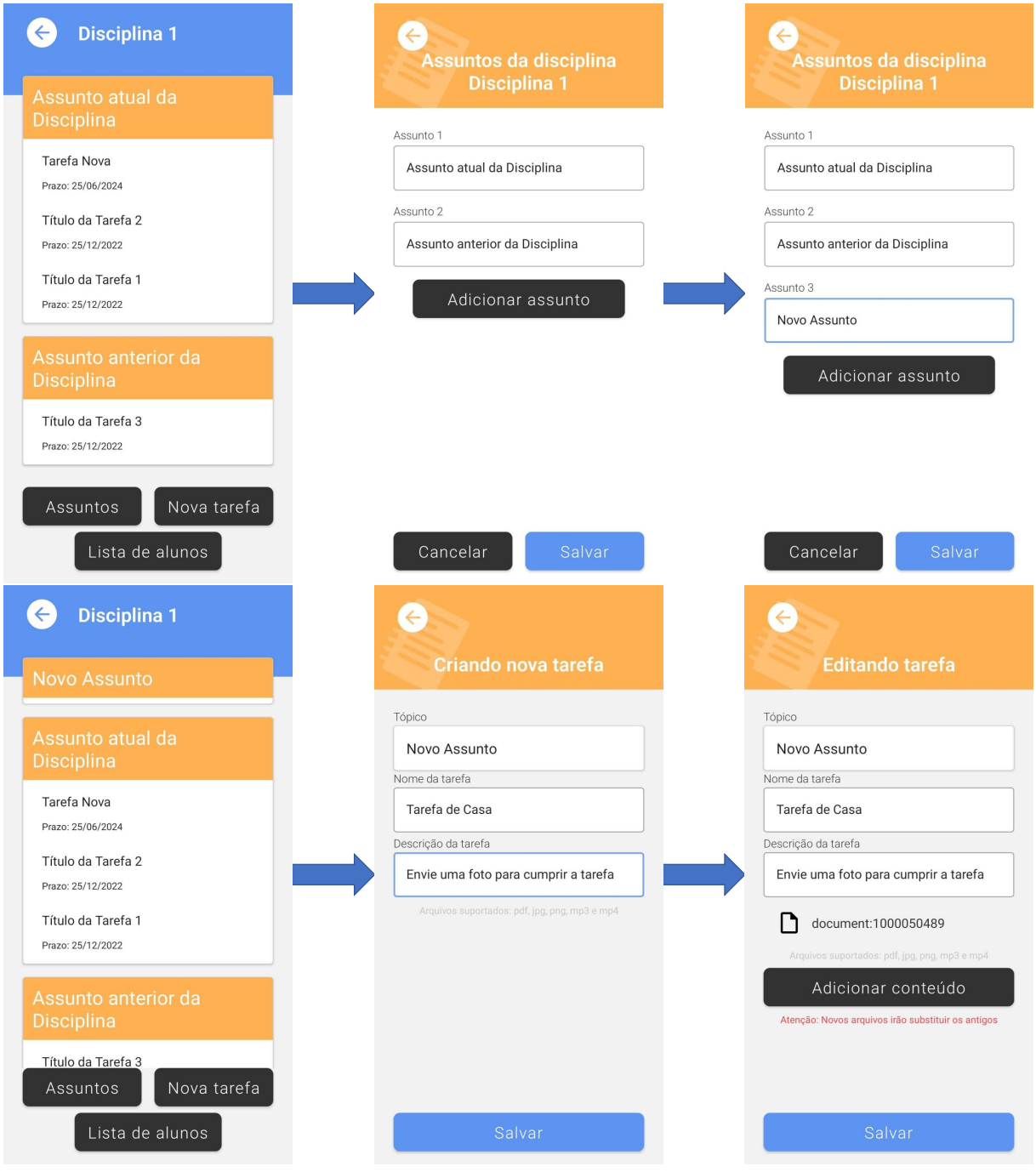

Figura B.4: Professor Cria Assunto e Publica Tarefa

B.5 Aluno Responde à Tarefa

Na imagem B.5, podemos observar:

1. Na visão da disciplina, o aluno pode selecionar a tarefa desejada
2. Ao abrir a tarefa e tentar visualizar o anexo, o usuário tem a opção de baixar o conteúdo
3. Após isso, o usuário tem o conteúdo disponível localmente no seu dispositivo
4. Ao selecionar o anexo, o dispositivo exibe opções para visualização
5. Ao selecionar *Responder tarefa*, o usuário pode escolher só marcá-la como concluída ou enviar uma resposta em anexo
6. Após selecionar o anexo, o usuário pode submeter a resposta

B.6 Professor Avalia Resposta do Aluno

Na imagem B.6, podemos observar:

1. Na visão da disciplina, o professor pode selecionar a tarefa desejada
2. Ao selecionar a tarefa, o professor pode ver as respostas dos alunos
3. Ao selecionar o aluno, o professor é capaz de fazer o download do anexo para avaliação
4. Após a avaliação, o professor retorna para a visão dos alunos que enviaram as tarefas e pode escolher avaliar
5. Ao avaliar a tarefa, é aberto um *pop-up* para que o professor possa dar uma nota para o aluno
6. Após atualizar as avalições, o professor pode salvar o progresso das avaliações

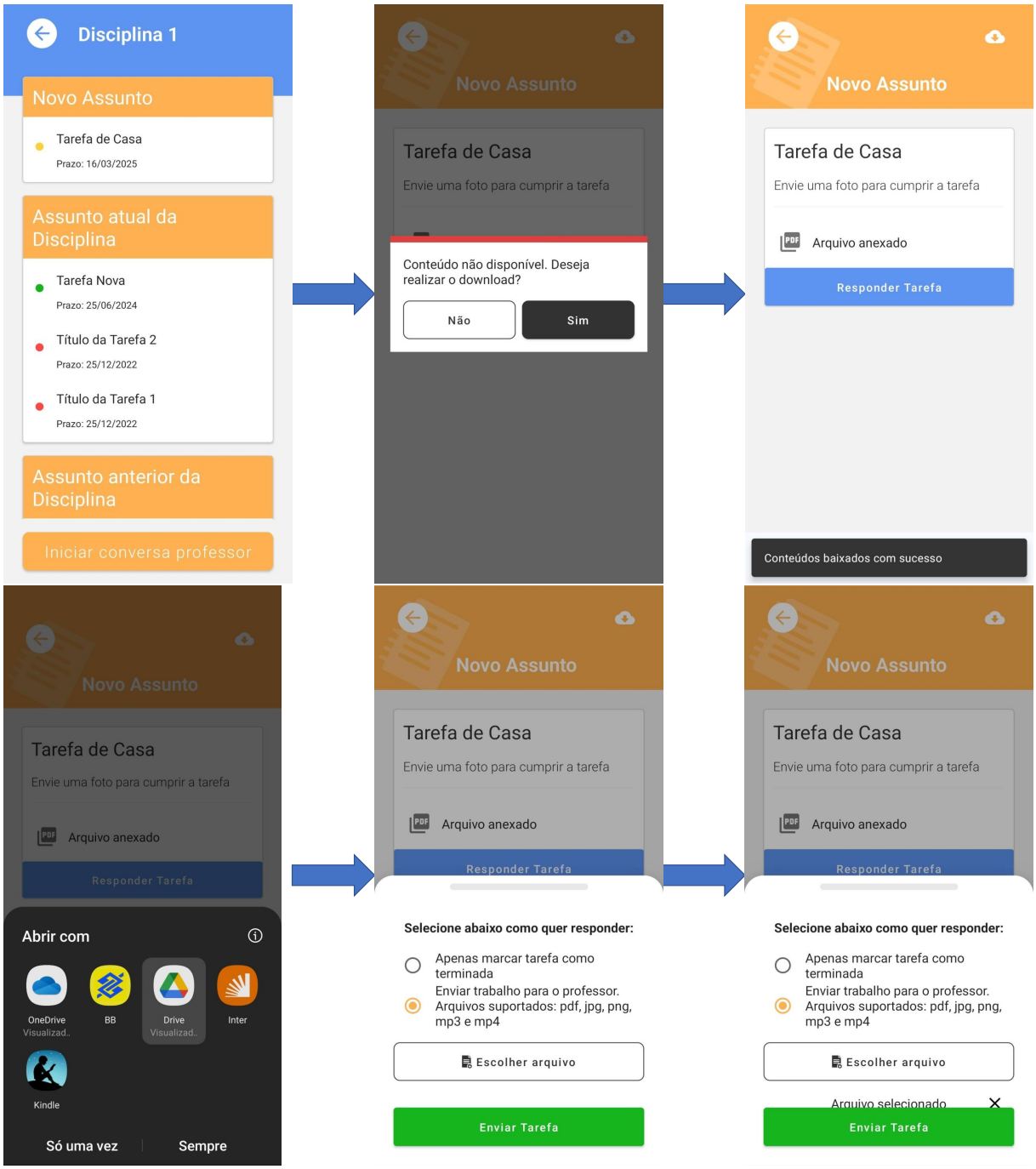

Figura B.5: Aluno Responde à Tarefa

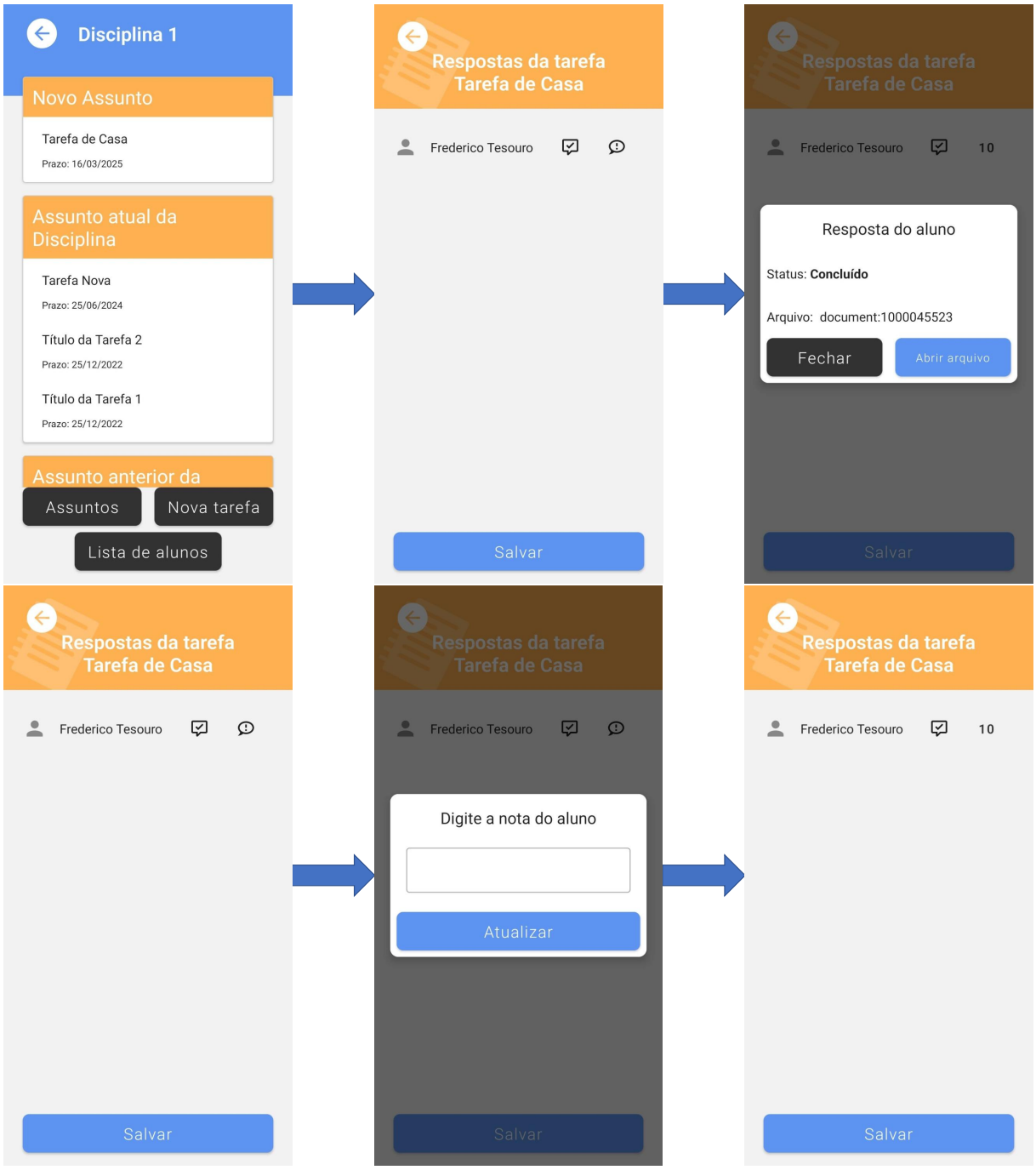

Figura B.6: Professor Avalia Resposta do Aluno

B.7 Visões Disponíveis para o Perfil de Responsável

Na imagem B.6, podemos observar:

1. Quais disciplinas o aluno está matriculado
2. As tarefas passadas pelo professor
3. O perfil de usuário

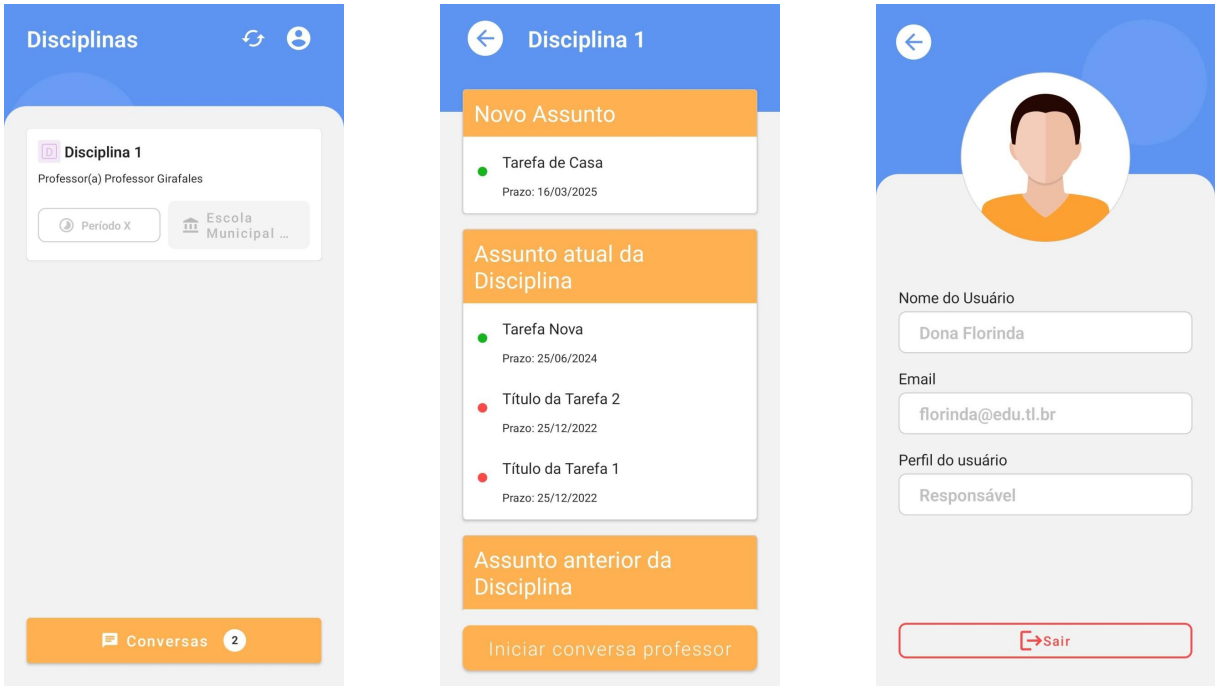

Figura B.7: Visões Disponíveis para o Perfil de Responsável