GESTÃO E QUALIDADE DE SOFTWARE

EVELLYN CRUZ SOUZA RA:823213551
AMANDA AGUSTINHO COSTA RA:823150503
JOÃO PEDRO AGUSTINHO COSTA RA:823223417
ALLAN PAULUK MEDEIROS RA:822142483
GABRIEL CRISPINO TORRES RA: 8221410678

PROF. ROBSON CALVETTI

EVOLUÇÃO DO PROJETO

Os commits mostram o ciclo completo de evolução do sistema:

- **Initial commit** e estrutura básica do projeto
- **Upload dos arquivos iniciais**
- **Ajustes de dependências** e criação do *.gitignore*
- **Refatoração POO no Node e no Python** para separar responsabilidades
- **Organização do código em camadas** (Repository, Client, Predictor, API)
- **Entrada dos testes unitários** em ambas as linguagens
- **Melhorias recentes nos testes** (API, predictor, prediction service, CRUD, mocks)

Resumo: os commits revelam a transformação do protótipo inicial em um projeto modular, estável e testável.

TRABALHO EM GRUPO E DISTRIBUIÇÃO DE TAREFAS

O histórico de commits mostra como a equipe trabalhou:

- Divisão clara de responsabilidades
- Cada commit cobre um módulo específico
- Redução de conflito de código
- Organização contínua do repositório
- Evolução gradual e colaborativa

Resumo: o projeto evoluiu de forma organizada porque cada membro contribuiu em partes isoladas e revisadas.

REFATORAÇÃO NO NODE.JS

Principais commits relacionados

- **“refactor: aplicar POO no servidor Node e separar responsabilidades em classes”**

→ **Entrada do** *DiagnosisRepository* e *PredictionClient*.

- **“refactor: separação de responsabilidades e implementação de POO”**

→ **Rotas Express ficaram leves, agindo apenas como orquestradoras.**

- **“chore: ajustes em dependências e adição do .gitignore”**

→ **Preparação do ambiente.**

Resumo: o Node saiu de um código acoplado para uma arquitetura modular e testável.

REFATORAÇÃO NO PYTHON

Commits importantes no backend Python:

- **“Mudanças no código de testes para o prediction service”**

→ **Ajustes no** *HepatitisPredictor* e no *PredictionRepository*.

- **“Adicionando unit tests em OOP para predição”**

→ **Adaptação do serviço Python para ser testável.**

- **“Adicionar testes para o model_api.py”**

→ **Melhorias nos endpoints do Flask e carregamento do modelo.**

Impactos principais:

- API Flask reorganizada
- Predictor em POO
- Validações e erros mais robustos
- Fluxo de predição mais claro

Resumo: o Python consolidou uma estrutura limpa, orientada a objetos e fácil de testar.

TESTES UNITÁRIOS E COBERTURA

Commits de testes:

- Testes Python (predictor, API, validações)
- Testes Node (CRUD, erros, URLs, file system)
- Mocks de rede, arquivos e logs
- Testes com Supertest
- Cenários 200, 400, 500, 502
- Aumento de cobertura com pytest-cov e Jest

Resumo: a cobertura cresceu muito e o comportamento do sistema ficou previsível e confiável.

CONCLUSÃO

O histórico de commits mostra:

- A evolução do código de um protótipo para uma arquitetura limpa
- Aplicação clara de Clean Code (POO, SRP, modularização)
- O impacto da separação de responsabilidades na testabilidade
- Como o trabalho em grupo foi organizado: cada commit focado em uma parte do sistema
- A importância do Clean Code para crescimento, manutenção e colaboração

Conclusão geral:

O sistema se transformou em um backend moderno, organizado e sustentável. A refatoração orientada por Clean Code tornou tudo mais fácil de evoluir, testar e manter.

OBRIGADO