

COIN STUDY

SISTEMA DE MÉRITO ESTUDANTIL

INTEGRANTES

Ana Luiza Machado Alves

Lucas Henrique Chaves de Barros

Matheus Martins da Silva Porto

Yan Mariz Magalhães Cota

ARQUITETURA DO SISTEMA

Arquitetura em Camadas
Padrão Model View Controller (MVC)

MVC – MODEL

Possuí a lógica da aplicação;

Responsável pelas regras de negócios, persistência com o banco de dados e as classes de entidades;

Recebe as requisições da camada Controller e as responde a partir da lógica implementada;

Na nossa aplicação mapeamos as classes do domínio para tabelas no banco de dados usando anotações JPA e Hibernate.

MVC - MODEL

```
@Entity
@Table(name="aluno")
public class Aluno extends Pessoa{

    private String rg;

    @ManyToOne
    @JoinColumn(name="instituicao_id")
    private Instituicao instituicao;

    @ManyToOne
    @JoinColumn(name="curso_id")
    private Curso curso;

    public String getRg() {
        return this.rg;
    }

    public void setRg(String rg) {
        this.rg = rg;
    }

    public Instituicao getInstituicao() {
        return this.instituicao;
    }

    public void setInstituicao(Instituicao instituicao) {
        this.instituicao = instituicao;
    }
}
```

```
@Entity
@Table(name="empresa")
public class Empresa {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;

    private String nome;
    private String cnpj;

    @ManyToOne
    private Instituicao instituicao;

    public Long getId() {
        return this.id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getNome() {
        return this.nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getCnpj() {
        return this.cnpj;
    }
}
```

MVC - MODEL

```
public String login(String email, String password) {  
    Pessoa pessoa = getPessoaByLogin(email);  
    if (pessoa != null) {  
        if (pessoa.getSenha().equals(password)) {  
            if (pessoa instanceof Aluno) {  
                return "Login bem-sucedido como Aluno!";  
            } else if (pessoa instanceof Professor) {  
                return "Login bem-sucedido como Professor!";  
            }  
        } else {  
            if (pessoa instanceof Aluno) {  
                return "Senha incorreta para Aluno!";  
            } else if (pessoa instanceof Professor) {  
                return "Senha incorreta para Professor!";  
            }  
        }  
    }  
    return "Email não encontrado!";  
}
```


MVC - VIEW

Camada de visualização, interagindo diretamente com o usuário;

Essa camada é utilizada para se comunicar com a nossa API, enviando e recebendo dados;

Não é responsável por executar lógica de negócios.

Dashboard

Início

Saldo

Configurações

Editar Perfil

Nome

Yan cota

CPF

999.999.999-99

RG

Atualize seu RG

Atualizar

Cancelar

Deletar Conta

Dashboard

Início

Saldo

Configurações

Editar Perfil

Nome

Yan cota

CPF

999.999.999-99

RG

Atualize seu RG

Atualizar

Cancelar

Deletar Conta

MVC - CONTROLLER

Atua como intermediador entre Model e View;

Recebe as requisições da View e as encaminha para o Model;

Facilita a troca de dados entre a interface e a lógica de negócios;

Não contém lógica de negócios, apenas media a comunicação entre as camadas.

MVC - CONTROLLER

```
@RestController
@RequestMapping("/Alunos")
public class AlunoController{

    @Autowired
    AlunoService alunoService;

    @GetMapping
    public List<Aluno> findAll(){
        return this.alunoService.getAll();
    }

    @GetMapping("/{login}")
    public ResponseEntity<Aluno> findByLogin(@PathVariable String login){
        Aluno aluno= this.alunoService.getAlunoByLogin(login);
        if(aluno!=null){
            return ResponseEntity.ok().body(aluno);
        }else{
            return ResponseEntity.notFound().build();
        }
    }

    @PostMapping("/register")
    public ResponseEntity<Aluno> create(@RequestBody Aluno newAluno){
        try {
            this.alunoService.createAluno(newAluno);
            return ResponseEntity.ok().body(newAluno);
        } catch (Exception e) {
            return ResponseEntity.badRequest().build();
        }
    }
}
```

```
@RestController
@RequestMapping("/Professor")
public class ProfessorController {

    @Autowired
    ProfessorService professorService;

    @GetMapping
    public List<Professor> findAll() {
        return this.professorService.findAll();
    }

    @GetMapping("/{login}")
    public ResponseEntity<Professor> findByLogin(@PathVariable String login) {
        Professor Professor = this.professorService.findByLogin(login);
        if (Professor != null) {
            return ResponseEntity.ok().body(Professor);
        } else {
            return ResponseEntity.notFound().build();
        }
    }

    @PostMapping("/register")
    public ResponseEntity<Professor> create(@RequestBody Professor newProfessor) {
        try {
            this.professorService.createProfessor(newProfessor);
            return ResponseEntity.ok().body(newProfessor);
        } catch (Exception e) {
            return ResponseEntity.badRequest().build();
        }
    }
}
```

MVC - VANTAGENS

Separação de responsabilidades por camadas;

**Pode acelerar o desenvolvimento de aplicações,
com um desenvolvimento simultâneo das camadas;**

**Reutilização de camadas para diferentes interfaces
sem manutenção na lógica de negócios;**

Manutenção do sistema.

MVC - DESVANTAGENS

Complexidade em sistema menores;

Curva de aprendizado para desenvolvedores iniciantes pode ser desafiador;

Disciplina para respeitar a implementação por camadas.

CAMADA DE PERSISTÊNCIA

Para essa camada, utilizamos JPA e Hibernate, junto com o padrão Repository. Optamos pelo Repository em vez de DAO por sua maior integração com o Spring Data, facilitando a criação de consultas, deixando-as mais simples e eliminando a repetição de código desnecessária

REPOSITORY

Como Funciona?

O Repository simplifica o acesso aos dados, gerando automaticamente operações como salvar, buscar e deletar. Com o Spring Data, não é necessário implementar essas operações manualmente, e ainda é possível criar consultas personalizadas de forma fácil usando nomes de métodos ou anotações.

REPOSITORY V.S DAO

Vantagens:

- **Abstração mais alta:** O Repository foca em coleções de objetos de domínio, ocultando detalhes de persistência.
- **Separação de preocupações:** Desacopla a lógica de negócio da camada de acesso a dados.
- **Melhor testabilidade:** Mais fácil de mockar e testar a lógica de negócio sem dependências do banco de dados.
- **Flexibilidade:** Permite trocar a implementação de persistência (SQL, NoSQL) sem afetar a lógica de domínio.
- **Adequado para DDD:** O Repository é mais alinhado aos princípios de Domain-Driven Design.

REPOSITORY V.S DAO


Desvantagens:

- **Complexidade adicional:** O Repository pode ser mais abstrato e complexo de implementar.
- **Menos controle sobre consultas:** A abstração pode dificultar a otimização de queries específicas, enquanto DAO permite maior controle.
- **Possível duplicidade:** Quando combinado com ORM, pode gerar redundância com funcionalidades já providas.

REPOSITORY

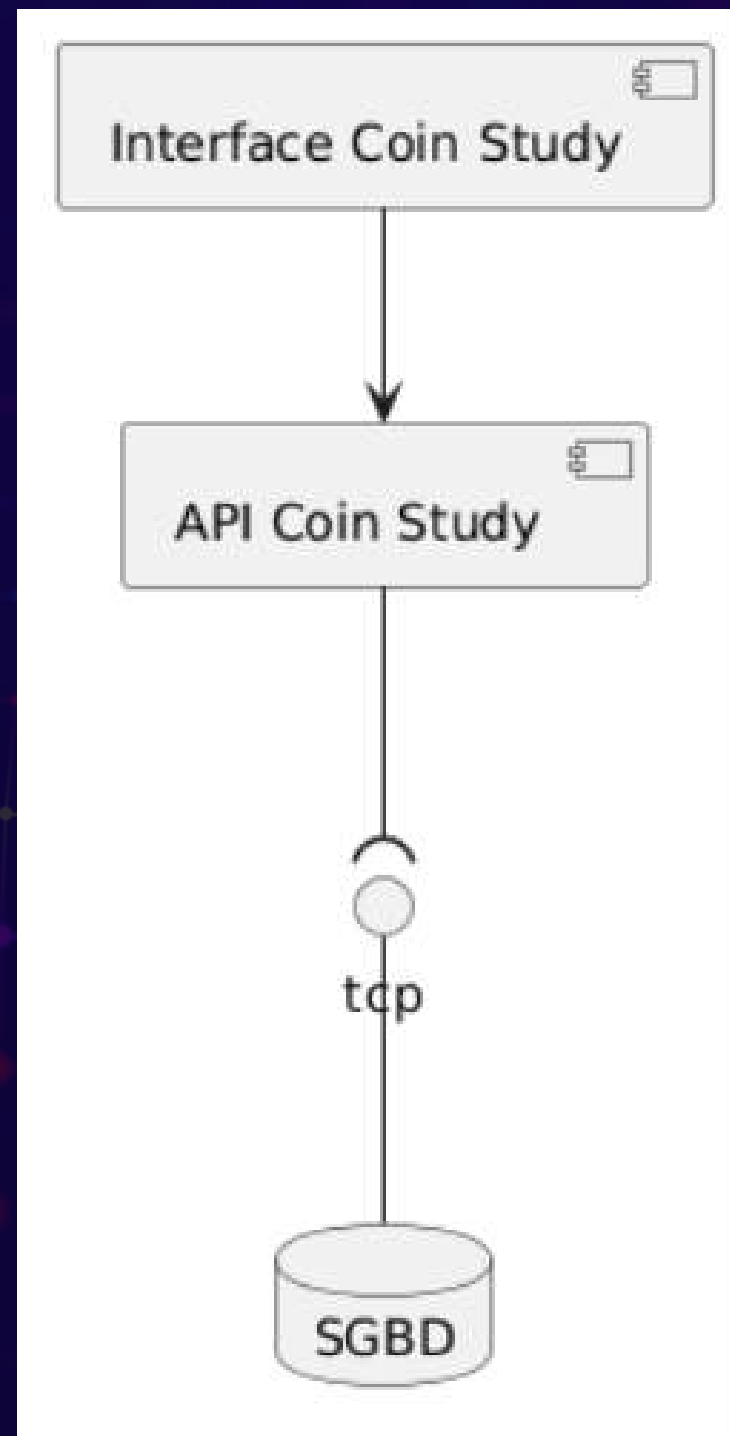
```
@Repository
public interface AlunoRepository extends JpaRepository<Aluno, String> {
    public Optional<Aluno> findByCpf(String cpf);
    public Optional<Aluno> findByLogin(String login);

    @Query("SELECT u FROM Aluno u WHERE u.login = :login")
    public Aluno getByLogin(String login);
}
```

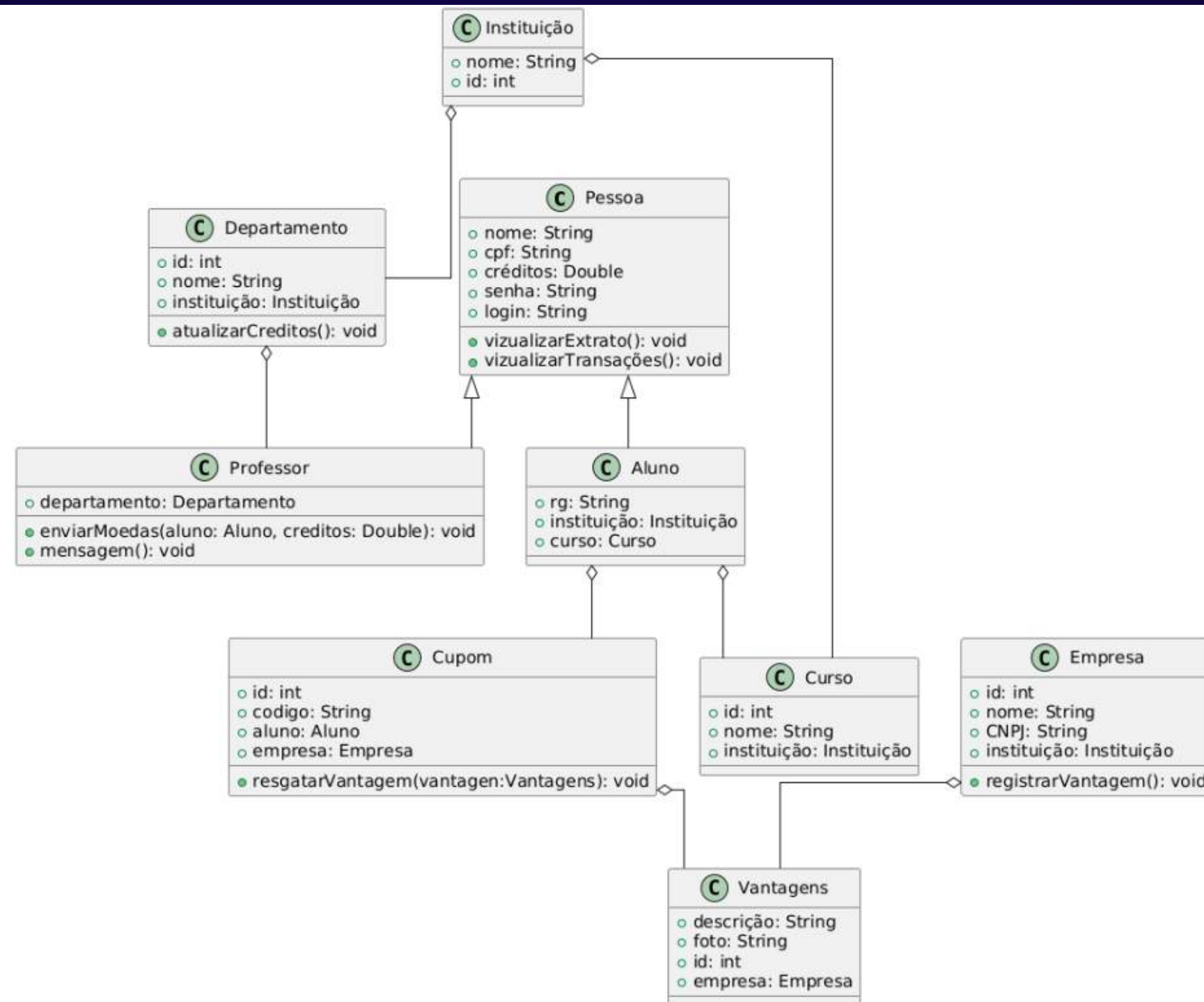


```
@Repository
public interface DepartamentoRepository extends JpaRepository<Departamento, Long>{
}
```

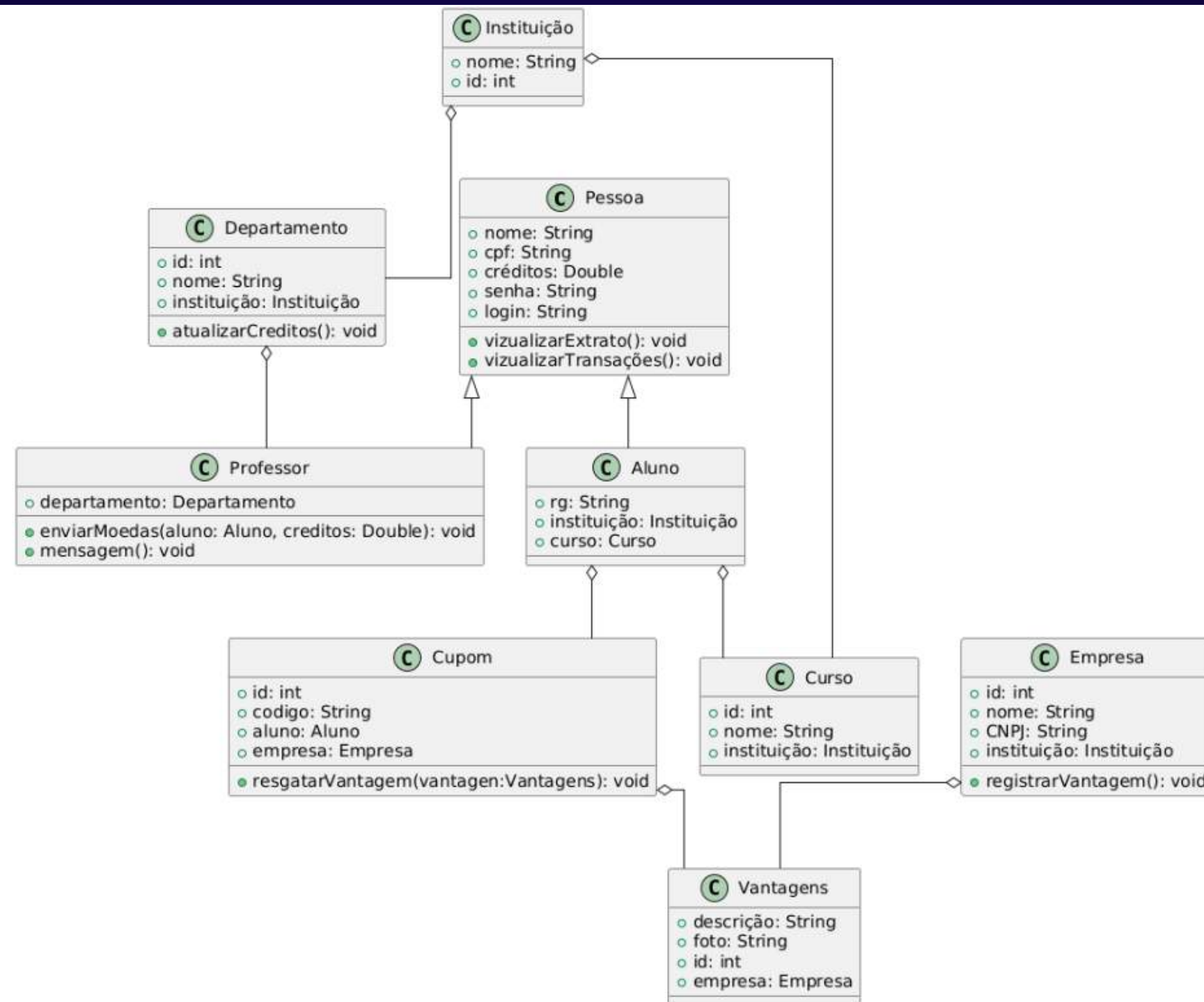

DIAGRAMAS



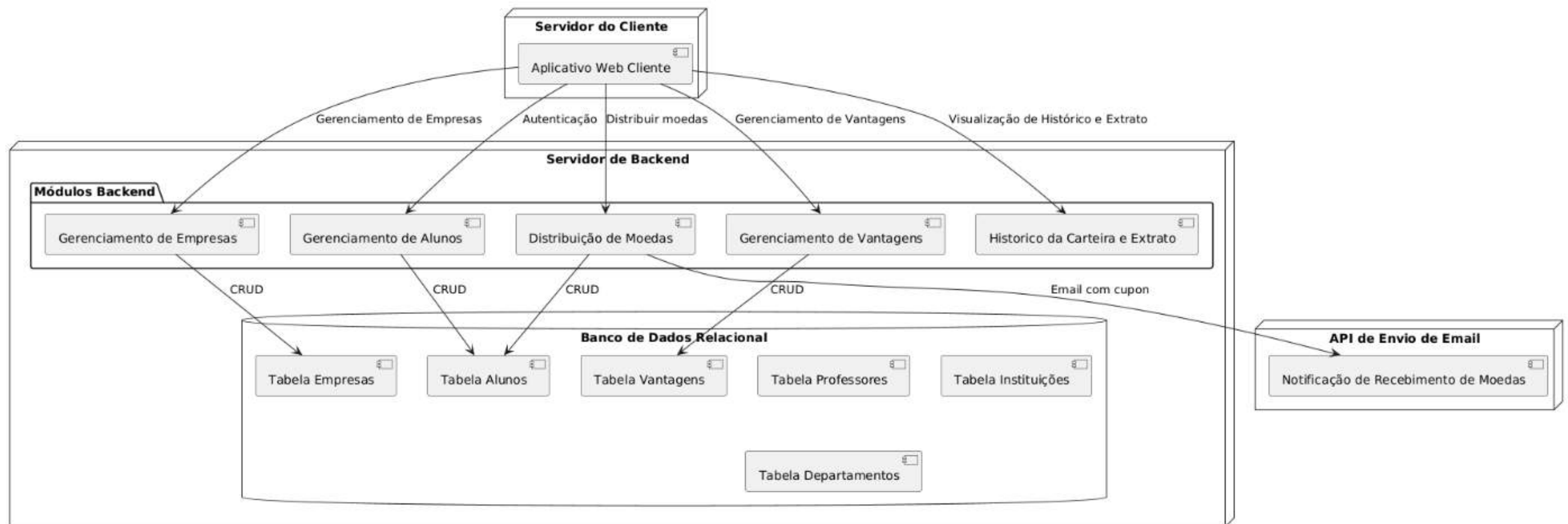
DIAGRAMAS



DIAGRAMAS



DIAGRAMAS





OBRIGADO