FACULDADE DE CIÊNCIAS APLICADAS E SOCIAS DE PETROLINA – FACAPE

SISTEMA DE REGISTRO DE PONTO

PARA PROFESSORES

LISTA DE TABELAS

LISTA DE QUADROS

SUMÁRIO

1. REQUISITOS FUNCIONAIS

Requisitos funcionais são as funções que o sistema realiza ou que é capaz de realizar. Para demonstrar os requisitos, foram usados casos de uso para cada requisito funcional.

Nome:	[RF01] – Registrar Ponto Entrada
Atores:	O professor
Prioridade:	Essencial
Pré-Condição:	-
Pós-Condição:	Ponto registrado com sucesso.
Fluxo de eventos	
Navegação para o fluxo principal:	1. Tela ***Principal.***
Fluxo principal:	1. Professor seleciona o curso. 2. Professor seleciona a disciplina. 3. Professor informa seu código. 4. Professor clica em ***Enviar***.
Fluxo Alternativo:	1. Professor fecha o navegador.

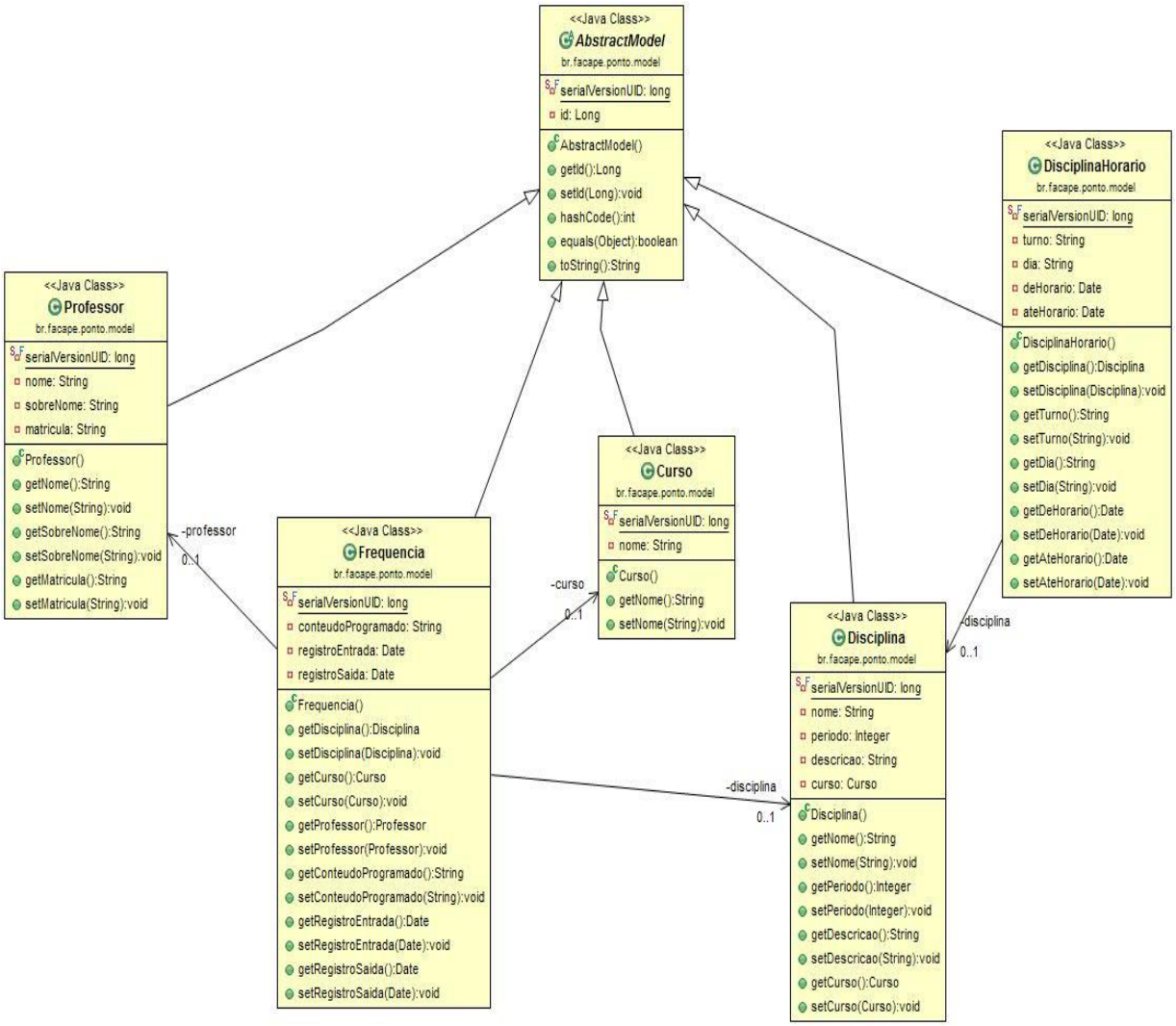
Tabela 1 - RF Registrar Ponto (Entrada)

Nome:	[RF02] – Registrar Ponto Saída
Atores:	O professor
Prioridade:	Essencial
Pré-Condição:	Ter registrado ponto de entrada.
Pós-Condição:	Ponto registrado com sucesso.
Fluxo de eventos	
Navegação para o fluxo principal:	1. Tela ***Principal.***
Fluxo principal:	1. Professor seleciona o curso. 2. Professor seleciona a disciplina. 3. Professor informa seu código. 4. Professor clica em ***Enviar***.
Fluxo Alternativo:	1. Professor fecha o navegador.

Tabela 2 - RF Registrar Ponto (Saída)

1.1 DIAGRAMA DE CLASSES

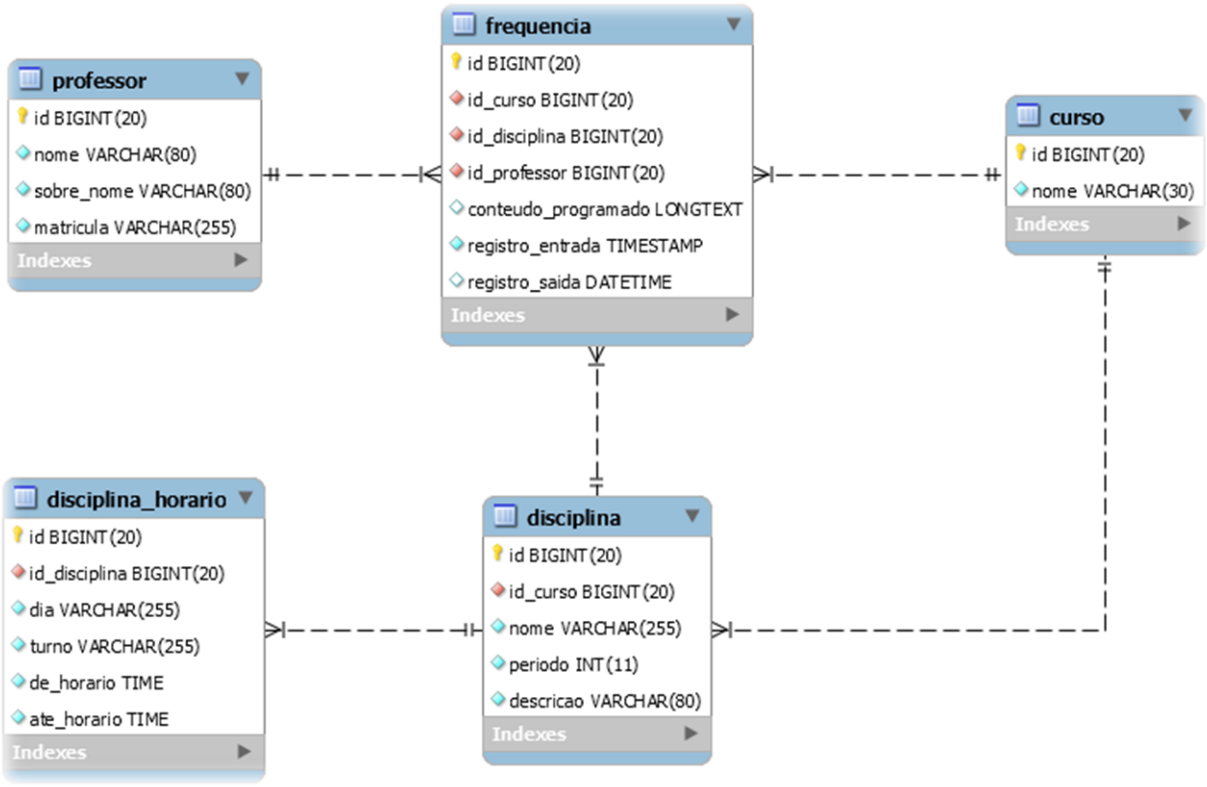
Uma representação do sistema através de ilustrações que representam classes é chamado de Diagrama de Classes.



Quadro 1 - Diagrama de Classes

1.2 MODELO DE DADOS

1.2.1 MODELO LÓGICO DO BANCO DE DADOS



Quadro 1 - Diagrama do Modelo Lógico do Banco de Dados

1.2.2 MODELO FÍSICO DO BANCO DE DADOS

NOME	TIPO	DOMINIO	NÃO NULO	CHAVE-PRIMARIA	CHAVE-ESTRANGEIRA
TABELA – PROFESSOR					
ID	BIGINT(20)		X	X	
MATRÍCULA	VARCHAR(10)		X		
NOME	VARCHAR(40)		X		
SOBRENOME	VARCHAR(40)		X		
TABELA – DISCIPLINA					
ID	BIGINT(20)		X	X	
IDCURSO	BIGINT(20)		X		
NOME	VARCHAR(40)		X		
PERIODO	INT(11)		X		
DESCRICAO	VARCHAR(80)				
TABELA – CURSO					
ID	SMALLINT		X	X	
NOME	VARCHAR(40)		X		
NUMERO	SMALLINT		X		
MATERIA	SMALLINT		X		
TABELA – FREQUENCIA					
ID	BIGINT(20)		X	X	
IDCURSO	BIGINT(20)		X		
IDDISCIPLINA	BIGINT(20)		X		
IDPROFESSOR	BIGINT (20)		X		
CONTEUDOPROGRAMADO	LONGTEXT		X		
REGISTROENTRADA	TIMESTAMP		X		
REGISTROSAIDA	TIMESTAMP				
TABELA – DISCIPLINA_HORARIO					
ID	BIGINT(20)		X		
IDDISCIPLINA	BIGINT(20)		X		
DIA	VARCHAR(20)				
TURNO	VARCHAR(20)				
DEHORARIO	TIME				
ATEHORARIO	TIME				

Tabela 3 - Modelo Físico do Banco de Dados

2 IMPLEMENTAÇÃO

O sistema foi construído usando o padrão de arquitetura MVC que é um padrão de arquitetura de software que separa a informação da interface com a qual o usuário interage é o padrão mais utilizado atualmente por desenvolvedores WEB.

Foram usados os seguintes pacotes:

- Model – contém as classes de "domínio" do sistema.
- Controller – Processa e responde os eventos (requisições) da View.
- Repository – responsável pela persistência dos dados.
- Service – contém a lógica mais especifica do sistema.
- Messages – contém mensagens.
- Pasta Deployed Resources (View), parte do sistema que o usuário interage e manipula.

As classes utilizadas foram:

- FrequenciaMB do pacote Controller, contém o método que recebe os dados da View e envia para o Service onde serão processado e enviados para o pacote Repository para serem persistido.
- AbstractRepository, do pacote Repository, e uma classe abstrata que contém os métodos de persistência padrões como inserir, alterar, excluir, etc. Essa classe é herdada pelos repositórios individuais.
- Repository do pacote Repository, interface contendo os métodos padrões que são implementados pela classe AbstractRepository.
- Curso, Disciplina, Frequência, Professor e Disciplina Horário do pacote Model, são as classes quem contém os dados que serão manipulados e persistidos.
- AbstractModel do pacote Model, contém dados comuns a todoas as classes e é usado para mapeamento na classe Repository e AbstractRepository.
- FrequenciaService do pacote Service, contém a lógica especifica do sistema e repassa os dados para o pacote Repository onde serão persistidos no banco de dados.

Para as View's:

- index.xhtml, da pasta Deployed Resources implementado usado componentes gráficos do Freamwork PrimeFaces versão mobile.

Outros:

- persistence, contém as configurações de comunicação com o banco de dados, bem como usuário e senha, etc.

3 CONHECENDO MELHOR OS IDES

3.1 JAVA

A linguagem Java foi desenvolvida pela Sun Microsystems, sendo que sua característica mais marcante é a possibilidade de programas escritos em Java serem executados virtualmente em qualquer plataforma, isto é, em qualquer Sistema Operacional – SO – (exemplo Windows). O responsável por isso é a JVM (Java Virtual Machine) que faz a tradução do programa para a plataforma sobre a qual está executando.

Uma grande vantagem dessa linguagem sobre as demais é que ela foi criada voltada para o desenvolvimento web enquanto as demais tiveram que se adaptar ao ambiente *www*.

Características Técnicas:

- Orientação a Objetos: Suporte ao paradigma de programação orientada a objetos;
- *Multithreading*: possibilidade de desenvolvimento utilizando *threads*;
- Suporte à comunicação: classes para programação em rede;
- Acesso remoto a banco de dados - Dados recuperados e/ou armazenados de qualquer ponto da Internet;
- Segurança: mecanismos de segurança que a linguagem oferece para a realização de processos pela Internet.

3.2 ECLIPSE

Eclipse é um IDE para desenvolvimento Java, porém suporta várias outras linguagens a partir de plugins como C/C++,PHP,ColdFusion, Python, Scala e plataforma Android. Ele foi feito em Java e segue o modelo open source de desenvolvimento de software. Atualmente faz parte do kit de desenvolvimento de software recomendado para desenvolvedores Android.

O projeto Eclipse foi iniciado na IBM que desenvolveu a primeira versão do produto e doou-o como software livre para a comunidade. O gasto inicial da IBM no produto foi de mais de 40 milhões de dólares. Hoje, o Eclipse é o IDE Java mais utilizado no mundo. Possui como característica marcante o uso da SWT e não do Swing como biblioteca gráfica, a forte orientação ao desenvolvimento baseado em plug-ins e o amplo suporte ao desenvolvedor com

centenas de plug-ins que procuram atender as diferentes necessidades de diferentes programadores.

O software Eclipse tem a licença EPL (Eclipse Public License).

3.3 MYSQL

O MySQL é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL (Linguagem de Consulta Estruturada, do inglês Structured Query Language) como interface. É atualmente um dos bancos de dados mais populares, com mais de 10 milhões de instalações pelo mundo.

Entre os usuários do banco de dados MySQL estão: NASA, Friendster, Banco Bradesco, Dataprev, HP, Nokia, Sony, Lufthansa, U.S. Army, U.S. Federal Reserve Bank, Associated Press, Alcatel, Slashdot, Cisco Systems, Google, entre outros.

O MySQL foi criado na Suécia por suecos e um finlandês: David Axmark, Allan Larsson e Michael "Monty" Widenius, que têm trabalhado juntos desde a década de 1980. Hoje seu desenvolvimento e manutenção empregam aproximadamente 400 profissionais no mundo inteiro, e mais de mil contribuem testando o software, integrando-o a outros produtos, e escrevendo a respeito dele.

3.4 OUTRAS FERRAMENTAS UTILIZADAS

- Weld (para injeção de dependências , CDI).
- Hibernate (orm, validações, manipulação de dados).
- JPA (API de persistência dados).
- Primefaces (biblioteca de componentes Web, versão mobile usado no Front-End).
- Omnifaces (biblioteca de utilitários, facilitar o desenvolvimento JSF. para navegação, mensagens, validações de tela, sessões, etc).
- Tomcat (container de aplicações (servlets), usado para deploy da aplicação. Versão 8.5).
- Java Server Faces - JSF (freamework que facilita o desenvolvimento de aplicações web com Java).
- Java JDK 1.8 (linguagem de programação usada no Back-End).

4 CÓDIGO DO BANCO DE DADOS

```
-- MySQL Script generated by MySQL Workbench
-- Mon Jun 19 21:53:02 2017
-- Model: New Model    Version: 1.0
-- MySQL Workbench Forward Engineering
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
-- -----------------------------------------------------
-- Schema ponto-professor
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `ponto-professor` DEFAULT CHARACTER SET utf8 ;
USE `ponto-professor` ;
-- -----------------------------------------------------
-- Table `ponto-professor`.`curso`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ponto-professor`.`curso` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(30) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
AUTO_INCREMENT = 3
DEFAULT CHARACTER SET = utf8;
-- -----------------------------------------------------
-- Table `ponto-professor`.`disciplina`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ponto-professor`.`disciplina` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `id_curso` BIGINT(20) NOT NULL,
  `nome` VARCHAR(255) NOT NULL,
  `periodo` INT(11) NOT NULL,
  `descricao` VARCHAR(80) NOT NULL,
  PRIMARY KEY (`id`),
```

```
  INDEX `FKncelm81kvdx8m08q5rbpn2jru` (`id_curso` ASC),
  CONSTRAINT `FKncelm81kvdx8m08q5rbpn2jru`
    FOREIGN KEY (`id_curso`)
    REFERENCES `ponto-professor`.`curso` (`id`))
ENGINE = InnoDB
AUTO_INCREMENT = 4
DEFAULT CHARACTER SET = utf8;
-- -----------------------------------------------------
-- Table `ponto-professor`.`disciplina_horario`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ponto-professor`.`disciplina_horario` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `id_disciplina` BIGINT(20) NOT NULL,
  `dia` VARCHAR(255) NOT NULL,
  `turno` VARCHAR(255) NOT NULL,
  `de_horario` TIME NOT NULL,
  `ate_horario` TIME NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `FKjlgbmrk7equrbo6cja16cdokx` (`id_disciplina` ASC),
  CONSTRAINT `FKjlgbmrk7equrbo6cja16cdokx`
    FOREIGN KEY (`id_disciplina`)
    REFERENCES `ponto-professor`.`disciplina` (`id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
-- -----------------------------------------------------
-- Table `ponto-professor`.`professor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ponto-professor`.`professor` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(80) NOT NULL,
  `sobre_nome` VARCHAR(80) NOT NULL,
  `matricula` VARCHAR(255) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB
```

```
AUTO_INCREMENT = 4
DEFAULT CHARACTER SET = utf8;
-- -----------------------------------------------------
-- Table `ponto-professor`.`frequencia`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ponto-professor`.`frequencia` (
  `id` BIGINT(20) NOT NULL AUTO_INCREMENT,
  `id_curso` BIGINT(20) NOT NULL,
  `id_disciplina` BIGINT(20) NOT NULL,
  `id_professor` BIGINT(20) NOT NULL,
  `conteudo_programado` LONGTEXT NULL DEFAULT NULL,
  `registro_entrada` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `registro_saida` DATETIME NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  INDEX `FKs0l1upem7tbsji5xiupjr267s` (`id_curso` ASC),
  INDEX `FKqx8kr59tfkhriyjtoepiwq0kw` (`id_disciplina` ASC),
  INDEX `FKtbvetdyj05xfexv8qr41v52d6` (`id_professor` ASC),
  CONSTRAINT `FKqx8kr59tfkhriyjtoepiwq0kw`
    FOREIGN KEY (`id_disciplina`)
    REFERENCES `ponto-professor`.`disciplina` (`id`),
  CONSTRAINT `FKs0l1upem7tbsji5xiupjr267s`
    FOREIGN KEY (`id_curso`)
    REFERENCES `ponto-professor`.`curso` (`id`),
  CONSTRAINT `FKtbvetdyj05xfexv8qr41v52d6`
    FOREIGN KEY (`id_professor`)
    REFERENCES `ponto-professor`.`professor` (`id`))
ENGINE = InnoDB
AUTO_INCREMENT = 3
DEFAULT CHARACTER SET = utf8;
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

5 CÓDIGO FONTE DO SISTEMA

```java
@MappedSuperclass
public abstract class AbstractModel implements Serializable {

	private static final long serialVersionUID = 4177637941540158455L;

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

	public Long getId() {
		return id;
	}

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

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((id == null) ? 0 : id.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		AbstractModel other = (AbstractModel) obj;
		if (id == null) {
			if (other.id != null)
				return false;
		} else if (!id.equals(other.id))
			return false;
		return true;
	}

	@Override
	public String toString() {
		return getClass().getName() + "[id=" + id + "]";
	}

}


@Entity
@Table(name = "curso")
public class Curso extends AbstractModel {

	private static final long serialVersionUID = -311777165242302544L;

	@Column(nullable = false, length = 40)
	private String nome;
```

```
	public String getNome() {
		return nome;
	}

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

}


@Entity
@Table(name = "disciplina")
public class Disciplina extends AbstractModel {

	private static final long serialVersionUID = -5283667011015245358L;

	@Column(nullable = false, length=40)
	private String nome;

	@Column(nullable = false, length=40)
	private Integer periodo;

	@Column(nullable = false, length=80)
	private String descricao;

	@ManyToOne
	@JoinColumn(name="id_curso", nullable=false)
	private Curso curso;

	// get e set

	public String getNome() {
		return nome;
	}

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

	public Integer getPeriodo() {
		return periodo;
	}

	public void setPeriodo(Integer periodo) {
		this.periodo = periodo;
	}

	public String getDescricao() {
		return descricao;
	}

	public void setDescricao(String descricao) {
		this.descricao = descricao;
	}

	public Curso getCurso() {
		return curso;
	}
```

```
	public void setCurso(Curso curso) {
		this.curso = curso;
	}


}


@Entity
@Table(name="disciplina_horario")
public class DisciplinaHorario extends AbstractModel {

	private static final long serialVersionUID = -2235482068183870279L;


	@ManyToOne
	@JoinColumn(name="id_disciplina", nullable=false)
	private Disciplina disciplina;

	@Column(nullable = false, length=20)
	private String turno;

	@Column(nullable = false, length=20)
	private String dia;

	@Column(name = "de_horario", nullable=false)
	@Temporal(TemporalType.TIME)
	private Date deHorario;

	@Column(name = "ate_horario", nullable=false)
	@Temporal(TemporalType.TIME)
	private Date ateHorario;

	// get e set
	public Disciplina getDisciplina() {
		return disciplina;
	}

	public void setDisciplina(Disciplina disciplina) {
		this.disciplina = disciplina;
	}

	public String getTurno() {
		return turno;
	}

	public void setTurno(String turno) {
		this.turno = turno;
	}

	public String getDia() {
		return dia;
	}

	public void setDia(String dia) {
		this.dia = dia;
	}

	public Date getDeHorario() {
		return deHorario;
	}
```

```
	public void setDeHorario(Date deHorario) {
		this.deHorario = deHorario;
	}

	public Date getAteHorario() {
		return ateHorario;
	}

	public void setAteHorario(Date ateHorario) {
		this.ateHorario = ateHorario;
	}
}


@Table
@Entity(name = "frequencia")
public class Frequencia extends AbstractModel {

	private static final long serialVersionUID = 1915074354213364037L;

	@ManyToOne
	@JoinColumn(name="id_disciplina", nullable=false)
	private Disciplina disciplina;

	@ManyToOne
	@JoinColumn(name="id_curso", nullable=false)
	private Curso curso;

	@ManyToOne
	@JoinColumn(name="id_professor", nullable=false)
	private Professor professor;

	@Lob
	@Column(name="conteudo_programado")
	private String conteudoProgramado;

	@Column(name = "registro_entrada", nullable=false, insertable=false,
updatable=false, columnDefinition="TIMESTAMP DEFAULT CURRENT_TIMESTAMP")
	@Temporal(TemporalType.TIMESTAMP)
	@CreationTimestamp
	private Date registroEntrada;

	@Column(name = "registro_saida", nullable=true)
	@Temporal(TemporalType.TIMESTAMP)
	private Date registroSaida;

	// get e set
	public Disciplina getDisciplina() {
		return disciplina;
	}

	public void setDisciplina(Disciplina disciplina) {
		this.disciplina = disciplina;
	}

	public Curso getCurso() {
		return curso;
	}

	public void setCurso(Curso curso) {
```

```
		this.curso = curso;
	}

	public Professor getProfessor() {
		return professor;
	}

	public void setProfessor(Professor professor) {
		this.professor = professor;
	}

	public String getConteudoProgramado() {
		return conteudoProgramado;
	}

	public void setConteudoProgramado(String conteudoProgramado) {
		this.conteudoProgramado = conteudoProgramado;
	}

	public Date getRegistroEntrada() {
		return registroEntrada;
	}

	public void setRegistroEntrada(Date registroEntrada) {
		this.registroEntrada = registroEntrada;
	}

	public Date getRegistroSaida() {
		return registroSaida;
	}

	public void setRegistroSaida(Date registroSaida) {
		this.registroSaida = registroSaida;
	}
}


@Entity
@Table(name = "professor")
public class Professor extends AbstractModel {


	private static final long serialVersionUID = -2272347706786120539L;

	@Column(nullable=false, length=40)
	private String nome;

	@Column(name = "sobre_nome", nullable = false, length = 40)
	private String sobreNome;

	@Column(nullable = false)
	private String matricula;


	// get e set
	public String getNome() {
		return nome;
	}

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

```
	}

	public String getSobreNome() {
		return sobreNome;
	}

	public void setSobreNome(String sobreNome) {
		this.sobreNome = sobreNome;
	}

	public String getMatricula() {
		return matricula;
	}

	public void setMatricula(String matricula) {
		this.matricula = matricula;
	}
}
```