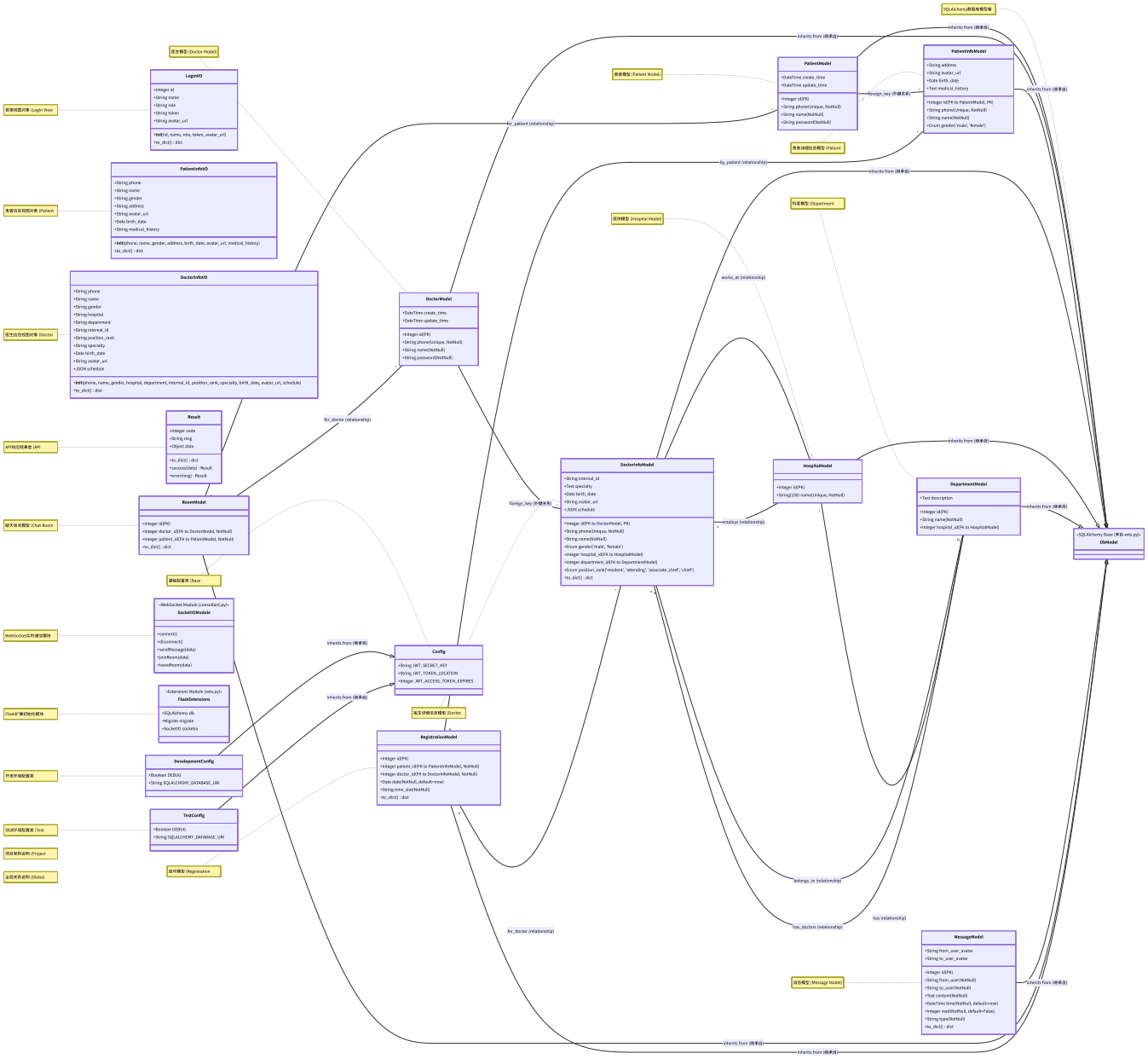
类图



整体架构概览

这个类图展现了医疗系统的项目结构：

- **数据模型层**：负责数据持久化（Models，继承自db.Model，图中显示为DbModel）
- **视图对象层**：负责数据传输（VOs，独立类，无继承关系）
- **工具层**：提供通用功能（Result、配置类、WebSocket等）
- **Blueprint函数层**：处理HTTP请求（函数式架构，非面向对象）

1. 数据库模型基础架构

DbModel（代表db.Model - SQLAlchemy基类）

```
<<SQLAlchemy Base (来自 exts.py)>>
```

- 这是整个数据模型的根基，所有数据库模型都直接继承自它
- 来自Flask-SQLAlchemy，通过 `from exts import db` 引入
- 图中使用DbModel代表实际的db.Model（因为Mermaid语法不支持带点的类名）

2. 用户管理子系统

患者相关模型

PatientModel（患者基础模型）

- 存储患者的账户信息：登录凭证、基本身份信息
- 字段分析：
  - `id (PK)`：主键，系统内唯一标识
  - `phone (Unique, NotNull)`：手机号作为登录用户名，确保唯一性
  - `name`：真实姓名
  - `password`：登录密码（实际应用中会加密存储）
  - `create_time/update_time`：审计字段，记录数据生命周期

PatientInfoModel（患者详细信息模型）

- 存储患者的详细个人和医疗信息
- 关键设计：
  - `id` 既是主键又是外键，与PatientModel形成一对一关系
  - `gender` 使用枚举类型，确保数据一致性
  - `medical_history` 使用Text类型，支持长文本病史记录
  - `avatar_url` 存储头像链接，可能指向云存储服务

医生相关模型

DoctorModel（医生基础模型）

- 结构与PatientModel相似，体现了统一的用户账户设计模式

DoctorInfoModel（医生详细信息模型）

- 最复杂的模型之一，包含丰富的职业信息：
  - `hospital_id/department_id`：建立医生与医疗机构的关联
  - `internal_id`：院内工号，便于医院内部管理
  - `position_rank`：职称枚举（住院医师→主治医师→副主任医师→主任医师）
  - `schedule`：JSON格式的排班表，灵活存储复杂的时间安排
  - `to_dict()` 方法：提供对象序列化能力，便于API响应

3. 医疗机构管理子系统

HospitalModel（医院模型）

- 简洁的设计，只存储核心信息
- `name` 字段唯一性约束，确保医院名称不重复

DepartmentModel（科室模型）

- 通过 `hospital_id` 与医院建立多对一关系
- `description` 字段支持科室详细介绍
- 与医院形成树状结构：医院→科室→医生

4. 业务流程子系统

通信功能

MessageModel（消息模型）

- 支持用户间的实时通信：
  - `from_user/to_user`：发送方和接收方
  - `from_user_avatar/to_user_avatar`：头像信息，优化UI显示
  - `read` 字段：消息状态管理
  - `type` 字段：支持不同类型的消息（文本、图片、语音等）

RoomModel（聊天房间模型）

- 管理医患一对一咨询会话
- 通过 `doctor_id` 和 `patient_id` 建立三方关系
- 可能用于会话状态管理、历史记录归档等

预约挂号功能

RegistrationModel（挂号模型）

- 核心业务模型，连接患者和医生：
  - 关联 `PatientInfoModel` 和 `DoctorInfoModel`
  - `date` 和 `time_slot`：精确的时间管理
  - `time_slot` 使用预定义值（'morning'/'afternoon'），简化时间段管理

5. 数据传输层（VO系统）

VO类设计（无继承关系）

项目中的VO类都是独立的普通类，没有基类继承关系

具体VO类

LoginVO（登录视图对象）

- 构造函数：`__init__(self, id, name, role, token, avatar_url)`
- 登录成功后的响应数据结构
- 包含 `token` 字段，支持JWT认证机制
- `role` 字段：区分用户类型（患者/医生/管理员）

PatientInfoVO/DoctorInfoVO

- 分别对应患者和医生的信息展示格式
- DoctorInfoVO 构造函数参数是 `hospital` 和 `department`，不是 `hospital_id` 和 `department_id`
- 可能过滤敏感信息（如密码），只传输必要数据

6. 系统支撑层

Result（统一响应类）

```
+Integer code (状态码)
+String msg (消息文本)
+Object data (数据负载)
+success(data) Result (成功静态方法)
+error(msg) Result (失败静态方法)
```

- 标准化API响应格式

- 静态工厂方法简化响应对象创建
- 支持RESTful API设计规范

配置管理系统

Config（基础配置类）

- JWT相关配置：安全认证的核心参数
- 为不同环境提供配置基础

DevelopmentConfig/TestConfig

- 环境特定配置：
  - `DEBUG`：控制调试模式
  - `SQLALCHEMY_DATABASE_URI`：不同环境使用不同数据库

7. WebSocket实时通信系统

SocketIOModule（consultant.py）

- 医患实时通信的核心实现
- 主要功能：
  - `connect()`/`disconnect()`：连接管理
  - `sendMessage(data)`：实时消息发送
  - `joinRoom(data)`/`leaveRoom(data)`：房间管理
- 使用Flask-SocketIO实现WebSocket通信
- 与MessageModel配合，提供完整的聊天功能

8. Flask扩展系统

FlaskExtensions（exts.py）

- **核心基础模块**：初始化所有Flask扩展
- 关键组件：
  - `db = SQLAlchemy()`：数据库ORM
  - `migrate = Migrate()`：数据库迁移
  - `socketio = SocketIO(cors_allowed_origins="*")`：WebSocket支持

9. 项目架构特点

函数式Blueprint架构

项目采用函数式而非面向对象架构

- **Blueprint文件**：patient.py, doctor.py, registration.py, chat.py, upload_file.py, ai.py, doctor_manage.py, patient_manage.py
- **路由处理**：每个文件包含多个函数，直接处理HTTP请求
- **无Service层**：业务逻辑直接在Blueprint函数中实现
- **无Controller类**：使用Flask的函数式路由模式

代码问题标注

RoomModel.to_dict()方法存在bug：

```
# 错误代码
"doctor_name": self.doctor_name,  # 属性不存在
"patient_name": self.patient_name  # 属性不存在
# 应该是
"doctor_name": self.doctor.name,
"patient_name": self.patient.name
```

10. 关系模式分析

一对一关系

- `PatientModel ↔ PatientInfoModel`
- `DoctorModel ↔ DoctorInfoModel`
- **设计意图**：分离账户信息和详细信息，支持渐进式信息完善

多对一关系

- `DoctorInfoModel → HospitalModel`：医生归属医院
- `DoctorInfoModel → DepartmentModel`：医生归属科室
- `DepartmentModel → HospitalModel`：科室归属医院
- `RegistrationModel → PatientInfoModel/DoctorInfoModel`：挂号关联医患

继承关系

- 所有Model类直接继承自 `db.Model`（图中显示为DbModel）：统一数据库操作接口
- **无VO基类**：所有VO类都是独立的普通类，没有继承关系
- Config类的继承：环境配置的层次化管理

11. 设计模式和最佳实践

1. **单一职责原则**：每个模型专注于特定的业务领域
2. **开闭原则**：通过继承支持功能扩展
3. **函数式设计**：Blueprint采用函数式而非面向对象设计
4. **工厂模式**：Result类的静态方法
5. **数据传输对象模式**：VO系统分离内部模型和外部接口

12. 系统优势与特点

1. **轻量级架构**：函数式Blueprint减少了代码复杂度
2. **实时通信能力**：WebSocket支持医患即时沟通
3. **数据安全**：分离的用户模型保护敏感信息
4. **业务灵活性**：JSON字段（如schedule）支持复杂业务需求
5. **开发效率**：标准化的响应格式和配置管理
6. **维护性**：明确的关系定义便于理解和维护