AI DOCTOR - COMPE561
Backend Wireframe and Models

PROJECT OVERVIEW

Project title: AI Doctor

Project Description: This web application acts as a website for a hospital or medical service. Which also provides an AI chat box to allow users to ask for simple advice. There would also be a form to book appointments for services at the center.

API ENDPOINTS

Endpoint	***HTTP Method***	***Description***	***Request Parameters***	***Response Structure***
/api/news	GET	Retrieve the latest 6 news article	None	[{id, title, date, summary, category,content}]
/api/news/{id}	GET	Retrieve a specific news article	Id (path)	{id, title, date, summary, category,content}
api/news	POST	Create new article	{title, date, summary, category}	{id, title, date, summary, category,content}
/api/news/{id}	PUT	Update an existing news article	Id (path), {title, date, summary, category}	{id, title, date, summary, category,content}
/api/news/{id}	DELETE	Delete a news article	Id (path)	{success : true}
/api/users	GET	Retrieve a list of all users	None	[{id, name, email, created_at}]
/api/users	POST	Create new user	{name, email, password}	{success: true}
/api/users/{id}	PUT	Update login information	{email, password}	{success: true}
/api/users/{id}	DELETE	Delete a user	None	{success: true}
/api/auth/login	POST	Authenticate user and get Token	{email, password}	{success: true, token}
/api/auth/me	GET	Get current user info	None	{id, name, email, created_at}
/api/admin/login	POST	Admin authentication	{username, password}	{success: true}

/api/admin/news	GET	Get all news articles (with pagination)	{token}	{id, title, summary, content, category, image_url, date, created_at}
/api/admin/news	POST	Create a new article	{token, title, summary, content,category, image_url, date}	{success: true}
/api/admin/news/{id}	PUT	Update an existing news article	{token, title, summary, content, category, image_url, date}	{success: true}
/api/admin/news/{id}	DELETE	Delete a news article	{token}	{success: true}
/api/admin/profile	GET	Get admin profile information	{token}	{id, username, email, created_at, last_login}
/api/admin/profile	PUT	Update admin profile	{token, email, current_password, new_password}	{success: true}
/api/admin/news/categories	GET	Get all news categories	{token}	{name, count}
/api/admin/dashboard	GET	Get statistics	{token}	{total_articles, articles_by_category {name, count}, recent_articles{id, title, date}}

DATA MODELS

NewsArticle:

- Attributes:
    + id: int, primary key, auto-increment
    + title: string, required
    + summary: str, required
    + content: text, required
    + category: str, required
    + image_url: str, required
    + date: datetime, required
    + status: str, default “draft” (draft, published, archived)
    + created_at: dateTime
    + updated_at: datetime
    + published_at: datetime, nullable
    + views_count: int, default 0
    + admin_id: int, foreign key

- Relationships: Belongs to Admin (admin_id references admins.id)

User:

- Attributes:
    + id: primary key, int
    + email: str, unique, required
    + name: str, required
    + password_hash: str, required
    + created_at: DateTime
    + updated_at: datetime, auto-set
    + last_login: DateTime
    + is_active: Boolean
    + verification_token: string, nullable
    + email_verified: boolean, default false
    + email_verified_at: datetime, nullable Relationships:
- Relationships: One user can have multiple UserSessions and have multiple RefreshToken

UserSession:

- Attributes:
    + id: int, primary key, auto-increment
    + user_id: int, foreign key
    + token: string, required, unique
    + expires_at: datetime, required
    + created_at: datetime, auto-set
    + ip_address: string, nullable
    + user_agent: string, nullable
- Relationships: Belongs to User

Token:

- Attributes:
    + token: str, primary key
    + token_type: str, required
    + expires_at: DateTime
    + user_id: Foreign Key to Users, nullable
- Relationships: Token can be acquired by user or admin.

Admin:

- Attributes:
    + id: primary key, int
    + username: str, unique, required
    + email: str, unique, required
    + password_hash: str, required
    + created_at: DateTime
    + updated_at: datetime, auto-set
    + last_login: datetime, nullable
    + is_active: boolean, default true
    + role: string, default 'editor' (super_admin, editor, viewer)
    + permissions: text (JSON), nullable
- Relationships: Have many NewsArticles, AdminSessions, RefreshTokens

AdminSession:

- Attributes:
    + id: int, primary key, auto-increment
    + admin_id: int, foreign key
    + token: string, required, unique
    + expires_at: datetime, required
    + created_at: datetime, auto-set
    + ip_address: string, nullable
    + user_agent: string, nullable
- Relationships: Belongs to Admin

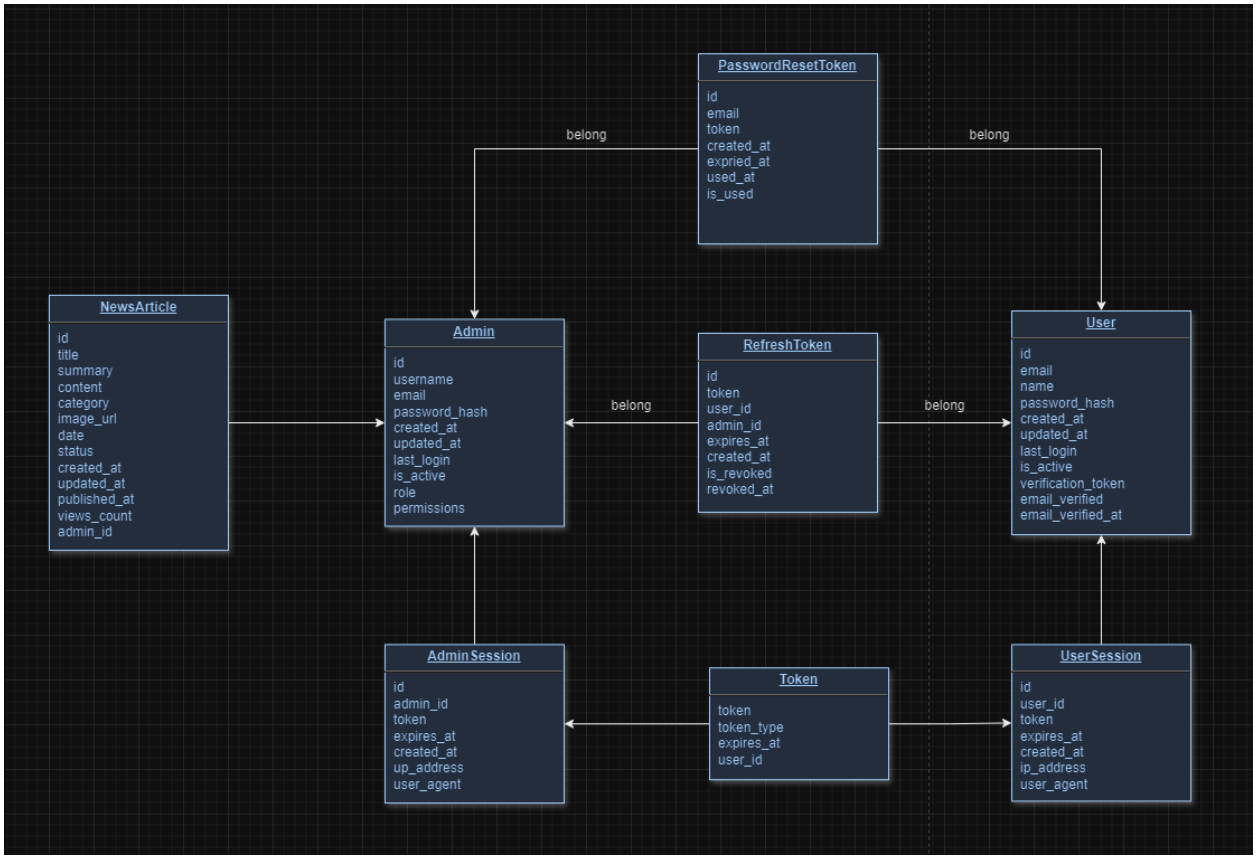
RefreshToken:

- Attributes:
    + id: int, primary key, auto-increment
    + token: string, required, unique
    + user_id: int, foreign key, nullable
    + admin_id: int, foreign key, nullable
    + expires_at: datetime, required
    + created_at: datetime, auto-set
    + is_revoked: boolean, default false
    + revoked_at: datetime, nullable
- Relationships: Belong to User/Admin (optional)

PasswordResetToken:

- Attributes:
    + id: int, primary key, auto-increment
    + email: string, required
    + token: string, required, unique
    + created_at: datetime, auto-set
    + expires_at: datetime, required
    + used_at: datetime, nullable
    + is_used: boolean, default “false”
- Relationships: No direct relationship

DATABASE SCHEMA



ADDITIONAL CONSIDERATIONS

1. *Authentication:*
   After some consideration, we decided to go with JWT (JSON Web Token) for authentication and password hashing for security. We will be implementing security features:
   - Password hashing using bcrypt
   - JWT tokens for authentication
   - Token expiration
   - Protected routes
   - User registration with duplicate checking
   - Login endpoint with token generation
   - User details endpoint
2. *Middleware:*
   We are planning to implement the middlewares as follow:
   - CORS - Handles cross-origin requests securely
   - Logging - Tracks all requests and their performance
   - Error Handling - Handles errors
   - Authentication - Protects routes requiring authentication
   - Admin Protection - Secures admin routes
   - Rate Limiting - Prevents abuse

   We will also look into more details with consideration of:
   - Middlewares order
   - Performancing impact - Each middleware will adds processing time

- Error handling - Proper error propagation
- Security - Careful implementation of auth check to ensure security
- Logging - Maybe consider using a proper logging framework
- Rate limiting - We considering using Redis

3. *Error handling:*
   - Standard Success Respsonses:

```
// GET /api/news
{
   "success": true,
   "data": [
      {
         "id": 1,
         "title": "Sample News",
         "summary": "Summary text",
         "date": "2024-10-28"
      }
   ],
   "meta": {
      "total": 100,
      "page": 1,
      "per_page": 10
   }
}

// POST /api/auth/login
{
   "success": true,
   "data": {
      "token": "eyJhbGc...",
      "user": {
         "id": 1,
         "name": "John Doe",
         "email": "john@example.com"
      }
   }
}
```

- Standard Error Repsonses:

```
// 400 Bad Request
{
   "success": false,
   "error": {
      "code": "VALIDATION_ERROR",
      "message": "Invalid input data",
      "details": {
         "email": "Invalid email format",
         "password": "Password must be at least 9 characters"
      }
   }
```

```
}

// 401 Unauthorized
{
  "success": false,
  "error": {
    "code": "UNAUTHORIZED",
    "message": "Invalid credentials"
  }
}

// 403 Forbidden
{
  "success": false,
  "error": {
    "code": "FORBIDDEN",
    "message": "You don't have permission to access this resource"
  }
}

// 404 Not Found
{
  "success": false,
  "error": {
    "code": "NOT_FOUND",
    "message": "Article not found"
  }
}

// 429 Too Many Requests
{
  "success": false,
  "error": {
    "code": "RATE_LIMIT_EXCEEDED",
    "message": "Too many requests",
    "details": {
      "wait_time": 60,
      "limit": 100,
      "reset_at": "2024-10-28T15:30:00Z"
    }
  }
}

// 500 Internal Server Error
{
  "success": false,
  "error": {
    "code": "INTERNAL_ERROR",
    "message": "An unexpected error occurred"
```

```
    }
}
```

4. *Testing:*

   We are currently considering these following tools for testing:

   - pytest for testing
   - coverage.py for code coverage
   - pytest-cov for coverage reporting
   - faker for generating test data
   - pytest-asyncio for async tests
   - locust for load testing