The challenge is to build a RESTful API for a news aggregator service that pulls articles from various sources and provides endpoints for a frontend application to consume.

1. Database Schema (Tables/Models & Relationships)

- **User**
    - Fields: id, email, password, created_at, updated_at
    - Relationships: one-to-Many with **UserPreference**
- **UserPreference**
    - Fields: id, user_id (FK), title
    - Relationships: many-to-one with **User,** Many-to-many with **ApiSource**, Many-to-many **Category**, and Many-to-many **Author** (via pivot tables or direct references)
- **Article**
    - Fields: id, title, content, source_id, published_at
    - Relationships: Many-to-one with **ApiSource**, Many-to-many **Category**, and Many-to-many **Author** (via pivot tables or direct references)
- **ApiSource**
    - Fields: id, name, api_url
    - Relationships: one-to-many with **Articles** and many-to-many with **UserPreferences**
- **Category**
    - Fields: id, title
    - Relationships: Many-to-many with **Articles and UserPreferences**
- **Author**
    - Fields: id, title
    - Relationships: Many-to-many with **Articles and UserPreferences**

User as u
-
id PK uuid
email varchar(255)
password varchar(255)
created_at DATETIME
updated_at DATETIME

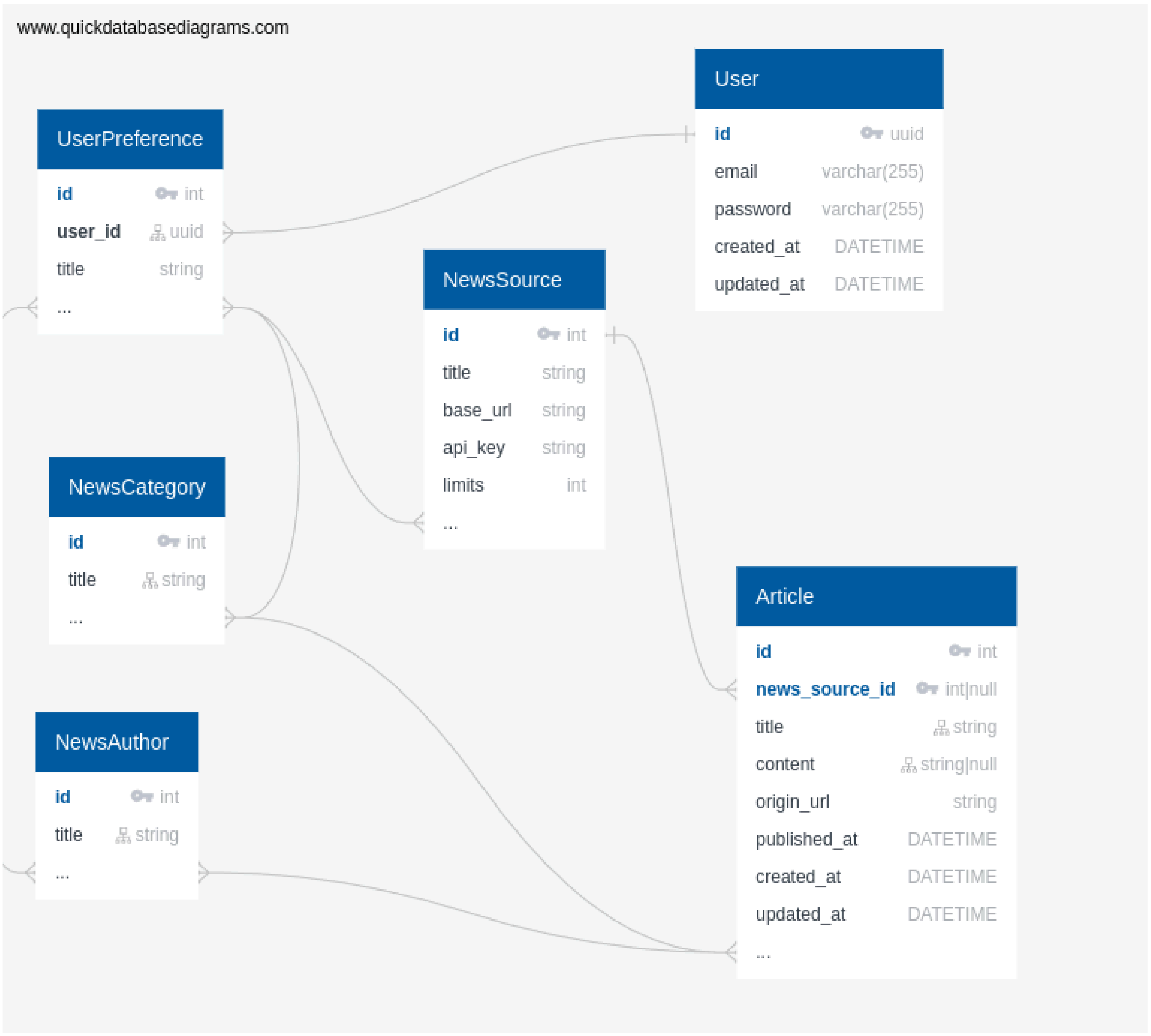
UserPreference as upr
rel >-< apis
rel >-< ath
rel >-< ct
-
id int PK
user_id uuid INDEX FK >- u.id
title string

Article as ar
rel >-< ct
rel >-< ath
-
id PK int
news_source_id int|null PK FK >- apis.id
title string INDEX
content string|null INDEX
published_at DATETIME
created_at DATETIME
updated_at DATETIME

ApiSource as apis
-
id PK int
title string
api_url string
api_key string
limits int

Category as ct
-
id PK int
title string INDEX

Author as ath
-
id PK int
title string INDEX



2. Tools & Laravel Packages

- **Database**:
  - **MySQL**
  - Docker Image: `mysql:latest`
- **Database Management**:
  - **Laravel Migrations/Seeder**
  - Laravel package: `php artisan migrate`, `php artisan db:seed`
- **Authentication**:
  - **Laravel Sanctum**
  - Laravel package: `laravel/sanctum`
- **Pagination**:
  - **Laravel Pagination**
  - Laravel package: Built-in
- **Search**:
  - **Laravel Scout & Meilisearch**
  - Laravel package: `laravel/scout`
  - Docker Image: `getmeili/meilisearch:latest`
- **Queueing**:
  - **Laravel Queues** (for background jobs like fetching articles)
  - Laravel package: laravel/horizon
  - Docker Image: `redis:latest` `(or rabbitmq:management)`
- **Caching**:
  - **Redis**
  - Docker Image: `redis:latest`
- **Testing**:
  - **PHPUnit**
  - Laravel package: Built-in (no extra package required)
- **OpenApi**:
  - Swagger
  - Laravel package: zircote/swagger-php, wotz/laravel-swagger-ui
- **Mailing**:
  - Mailpit
  - Docker Image: axllent/mailpit:latest
- **Monitoring**:
  - MailpTelescopeit
  - Laravel package: `laravel/telescope`
- **Docker**:
  - Docker Image: `php:8.3-fpm`, `nginx:latest`, `mysql:latest`, `redis:latest`, `getmeili/meilisearch:latest`

3. API URLs

- ✅**POST**: `/api/v1/auth/register`
- ✅**POST**: `/api/v1/auth/login`
- ✅**GET**: `/api/v1/auth/logout`
- ✅**POST**: `/api/v1/auth/password-forgot`
- ✅**POST**: `/api/v1/auth/password-update`
- ✅**POST,GET, PUT, DELETE**: `/api/v1/user/preferences`
- ✅**GET**: `/api/v1/user/preferences/{preference_id}`
- ✅**GET**: `/api/v1/articles`
- ✅**GET**: `/api/v1/articles/{article_id}`
- ✅**GET**: `/api/v1/news-categories`
- ✅**GET**: `/api/v1/news-sources`
- ✅**GET**: `/api/v1/news-authors`
- ✅**GET**: `/api/v1/user/feed`
- ✅**GET**: `/api/v1`
- **DELETE**: `/api/v1/user`

4. Web URLs

- ✅**POST**: `/password-reset`
- ✅**POST**: `/password-reset/{token}/{email}`
- ✅**GET**: `/api/docs`

5. Backend jobs

CreateFetchNewsSourceJob
Take source details on the local database. Create a new Job to fetch data from the specific source. Can create Jobs from multiple sources and store them in Redis.

FetchNewsSourceArticlesJob
Fetch news from a specific source and create multiple Jobs to convert received data into local DTOs.

PrepareToStoreFetchedArticleJob
Convert each source grabber DTO into a specific ArticleDTO.

StoreFetchedArticleJob
Take unified ArticleDTO and build ArticleModel. Store data in a local database.