Gran
applic

1. List of Services

- **Hosting**: AWS (EC2, ECS), Azure, or Google Cloud Platform (GCP).
- **Database**: PostgreSQL (Amazon RDS, Azure SQL Database) for relational data, Redis for caching.
- **Authentication**: Auth0 or AWS Cognito
- **File Storage**: AWS S3, Azure Blob Storage, or Google Cloud Storage for storing uploaded documents.
- **CI/CD Pipeline**: GitHub Actions, Jenkins, GitLab CI, or CircleCI for automating deployments.
- **Monitoring & Logging**: AWS CloudWatch, Prometheus + Grafana, or ELK (Elasticsearch, Logstash, Kibana) stack.
- **Message Queues/Background Jobs**: Kafka, SQS, or Celery handles background processing and event-driven communication between microservices, essential for scalability.
- **Search Functionality**: The search functionality will be implemented directly within the application service. It will likely perform SQL queries against the primary relational database (e.g., PostgreSQL) to retrieve search results. For more robust search service, Elasticsearch can be used for grant search and filtering.
- **Caching**: Redis or Memcached for session storage, frequently accessed data, and API rate-limiting.

ıtlN
ation

Database Schema

```
-- Users Table
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    hashed_password VARCHAR(255) NOT NULL,
    role VARCHAR(50) CHECK (role IN ('admin', 'applicant')) DEFAULT 'applicant',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);


-- Grants Table
CREATE TABLE grants (
    id SERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    description TEXT NOT NULL,
    category VARCHAR(255) NOT NULL,
    deadline TIMESTAMP NOT NULL,
    requirements TEXT,
    created_by INT REFERENCES users(id) ON DELETE SET NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);


-- Applications Table
CREATE TABLE applications (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id) ON DELETE CASCADE,
    grant_id INT REFERENCES grants(id) ON DELETE CASCADE,
    status VARCHAR(50) CHECK (status IN ('submitted', 'pending review', 'approved', 'rejected')) DEFAULT 'submitted',
    submitted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, grant_id)  -- Ensures that a user can apply to a grant only once
);


-- Documents Table
CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    application_id INT REFERENCES applications(id) ON DELETE CASCADE,
    document_url TEXT NOT NULL,
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);


-- Audit Logs Table
CREATE TABLE audit_logs (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id) ON DELETE SET NULL,
    action VARCHAR(255) NOT NULL,
    details TEXT,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

1. Functional Requirements

- **User Management**: Users can register as applicant or admins, log in, manage their profiles.
- **Application Management**: Admins can post grants and Users can submit applications, upload required documents, and track the status of their applications.
- **Notification System**: Users receive notifications regarding their application status or new grants.
- **Search and Filter**: Users can search and filter grants based on categories, deadlines, and other criteria.

2. Non-Functional Requirements

- **Scalability**: The system must scale to handle increasing numbers of users and data efficiently.
- **Performance**: The system should provide low-latency responses (< 200ms for most requests).
- **Availability**: The system should be highly available (99.9% uptime), considering failover and redundancy.
- **Security**: User data and documents must be secured using encryption at rest and in transit, with robust authentication and authorization mechanisms.
- **Maintainability**: The system should be modular and easy to maintain, allowing for independent updates and deployments.
- **Compliance**: The system should comply with regulations such as GDPR if handling sensitive user data.

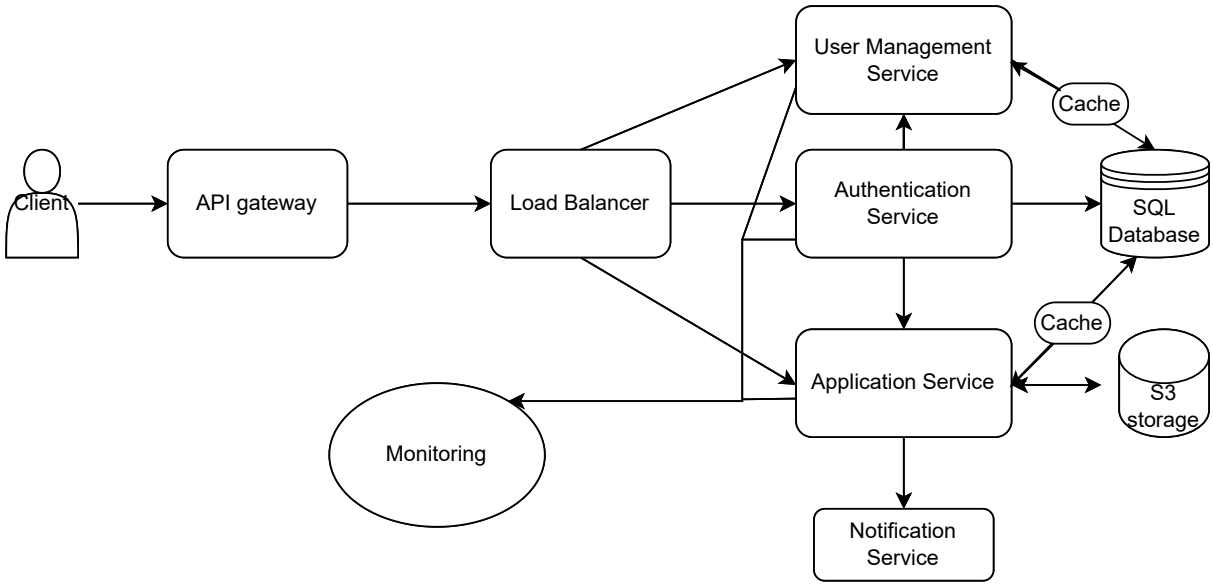
3. Scalability Considerations

I am assuming 100k active users per day and 10k active grants. if Each user makes 10 requests per day on average The RPS would be approximately 12 RPS. If we assume an average of 5 queries per request. it will be 58 QPS.

- **Horizontal Scaling**: The application should be designed to scale horizontally. Services should be stateless and easily deployable across multiple instances.
- **Database Sharding**: For a large number of users and grants, consider sharding or partitioning the database to distribute the load.
- **Caching**: Use caching for frequently accessed data like user sessions, popular grants, and search results.
- **Load Balancing**: A load balancer should be used to distribute incoming requests evenly across instances.

4. System Design Justification

- **Microservices vs. Monolithic Architecture**:
    - **Microservices Architecture** is preferred here due to the modularity, scalability, and independent deployability of different components (e.g., user management, Application management). It allows for more flexibility when scaling specific components independently.
- **Service Interactions**:
    - **Authentication Service**: Handles user authentication (JWT-based) and interacts with the user management service.
    - **Application Service**: Handles user applications, status tracking, and documents uploads.
    - **Notification Service**: Sends email notifications and updates to users.
- **API Gateway/Load Balancer**:
    - An API Gateway is placed in front of all microservices to handle routing, load balancing, and rate limiting. It also enforces security policies like authentication and authorization.
- **Data Flow**:
    1. **User Login/Register**: The user submits credentials via the API Gateway, which routes the request to the Authentication Service. The service validates the credentials, generates a JWT token, and returns it.
    2. **Grant Browsing**: The user searches for grants via the API Gateway, which forwards the request to the Application Service. The Application Service queries the database and returns the results.
    3. **Application Submission**: The user submits an application via the API Gateway, which routes the request to the Application Service. The Application Service handles file uploads and then saves the application data.
    4. **Notification Trigger**: Upon application submission or status change, the Notification Service is triggered to send an email to the user.
- **Security Measures**:
    - **Data Encryption**: Use TLS for data in transit and AES-256 encryption for data at rest.
    - **Role-Based Access Control (RBAC)**: Implement different access levels for users (applicants, admins).
    - **Audit Logs**: Keep track of all critical operations (e.g., grant creation, application submission) for compliance and troubleshooting.
    - **DDoS Protection**: Implement rate limiting and IP filtering at the API Gateway level.


User Management Service
Cache
Client
API gateway
Load Balancer
Authentication Service
SQL Database
Cache
Application Service
S3 storage
Monitoring
Notification Service