Architectural Choices Analysis

1. **Microservices Architecture with API Gateways:**
   - **Pros:**
     - Isolation and Scalability: Independent deployment and scaling of microservices, enhanced by API gateways.
     - Simplified Client Interaction: Clients interact via a unified API gateway, reducing complexity.
     - Security and Rate Limiting: API gateways enforce consistent security measures and rate limiting across services.
   - **Cons:**
     - Single Point of Failure: Potential single point of failure in the API gateway.
     - Learning Curve: Requires effort for effective implementation and management.
   - **Reasoning:** Chosen for scalability, isolation, and centralized policy enforcement, simplifying client-server communication.
2. **Modular Plugin System:**
   - **Pros:**
     - Extensibility: Easy addition of new features without core system modifications.
     - Reusability: Plugins can be reused in different contexts, promoting code reusability.
     - Separation of Concerns: Plugins handle specific functionalities, ensuring clear separation.
   - **Cons:**
     - Versioning Challenges: Managing plugin compatibility during upgrades.
   - **Reasoning:** Chosen for adaptability to future requirements and promoting code reusability without affecting core services.
3. **Database Isolation:**
   - **Pros:**
     - Security: Restricts direct database access, enhancing security.
     - Data Integrity: Maintains data integrity by enforcing service-specific data control.
   - **Cons:**
     - Increased Latency: Indirect database access through services may introduce slight latency.
   - **Reasoning:** Chosen to enhance security and maintain clear data ownership and control.

Areas for Improvement

1. **Request Validation:** Either json schema validation or validation using language library.
2. **Code Quality Improvement:** The code needs to separated into modules, separating concerns, and writing efficient code.
3. **Phone Numbers and Emails Verification:** Both needs to be verified.
4. **Enhanced Role Management:** Additional features to teams should be given to improve this platform.

Additional Considerations

1. **Use of Existing Auth Providers:** Either using auth0(cloud based) or authentik(self managed) for authentication. Enhances security, reduces development time, and offers advanced authentication features.
2. **Distribution using Kubernetes:** Right now the services are deployed using docker-compose. Efficient orchestration, scaling, and management of containerized applications for microservices.

Conclusion

Chosen architectural decisions aim for scalability, maintainability, and security. Continuous improvements in request validation, code quality, and role management are crucial. Integration with existing auth providers and potential migration to Kubernetes could further enhance efficiency and scalability.