Checklist for Interfaces

The Interfaces category covers all user-facing components of Orgo, including APIs, dashboards, and web clients. This checklist ensures interfaces are secure, user-friendly, and seamlessly integrated with the system.

---

1. Structure

[ ] API Directory:

Separate files for each functionality (e.g., email\_endpoints.py, task\_endpoints.py).

[ ] Frontend Directory:

Store templates (HTML/CSS/JS) and static assets (e.g., images, icons) in dedicated folders.

[ ] Separation of Concerns:

Ensure backend logic and frontend templates are decoupled.

---

2. API Design

[ ] RESTful Principles:

Use clear and consistent endpoints (e.g., /api/emails/send, /api/tasks/create).

[ ] Descriptive HTTP Methods:

Use appropriate methods:

GET for fetching data.

POST for creating data.

PUT for updating data.

DELETE for deleting data.

[ ] Versioning:

Include versioning in endpoint URLs (e.g., /api/v1/tasks/create).

[ ] Pagination:

Implement pagination for endpoints returning large datasets.

---

3. API Security

[ ] Authentication:

Require API tokens for all endpoints.

Use OAuth 2.0 or JWT for secure authentication.

[ ] Role-Based Access Control (RBAC):

Restrict access based on user roles (e.g., admin, user).

[ ] Input Validation:

Validate all incoming payloads for required fields and correct types.

[ ] Rate Limiting:

Prevent abuse by limiting the number of requests per user/IP.

---

4. API Documentation

[ ] Auto-Generate Documentation:

Use tools like Swagger or Postman to generate API documentation.

[ ] Include Examples:

Provide example requests and responses for each endpoint.

[ ] Error Codes:

Document error codes and their meanings (e.g., 400 Bad Request, 401 Unauthorized).

---

5. Frontend Design

[ ] Responsiveness:

Ensure dashboards and web clients are mobile-friendly.

[ ] Intuitive Navigation:

Use a clear and consistent navigation structure.

[ ] Accessibility:

Follow WCAG guidelines for accessibility (e.g., keyboard navigation, alt text for images).

[ ] Dynamic Updates:

Use AJAX or WebSockets for real-time updates where necessary.

---

6. Notifications

[ ] Multichannel Notifications:

Support email, SMS, and in-app notifications.

[ ] Customizable Preferences:

Allow users to configure notification settings (e.g., frequency, channel).

[ ] Tracking:

Log notifications sent and track delivery status.

---

7. Logging and Monitoring

[ ] API Logs:

Log all API requests and responses, including timestamps and IP addresses.

[ ] Frontend Logs:

Capture frontend errors (e.g., JavaScript exceptions) for debugging.

[ ] Alerting:

Generate alerts for API failures or excessive error rates.

---

8. Testing

[ ] Unit Tests:

Test individual API endpoints for functionality and edge cases.

[ ] Integration Tests:

Validate interactions between APIs and backend services.

[ ] Frontend Tests:

Use tools like Selenium or Cypress for testing user interfaces.

[ ] Security Tests:

Test for vulnerabilities like SQL injection or cross-site scripting (XSS).

---

9. Performance

[ ] API Optimization:

Minimize response times (<200ms for most requests).

Cache frequently requested data.

[ ] Frontend Optimization:

Compress assets (e.g., images, CSS, JS).

Minify and bundle static files.

[ ] Load Testing:

Simulate high traffic to ensure interfaces handle peak loads.

---

10. Scalability

[ ] Modular Design:

Ensure new endpoints or features can be added without affecting existing ones.

[ ] API Gateway:

Use an API gateway (e.g., Kong, AWS API Gateway) for routing and load balancing.

[ ] Stateless APIs:

Design APIs to be stateless for horizontal scaling.

---

Example Application for Email Interfaces

API Endpoints

/api/v1/emails/send:

Sends an email with required fields (recipient, subject, body).

/api/v1/emails/status:

Fetches the delivery status of an email.

Frontend Components

Email Dashboard:

Displays sent and received emails.

Allows filtering by recipient, date, or subject.

Checklist Applied

[ ] API Validation:

Check that all required fields (recipient, subject, body) are present in the payload.

[ ] Role Restrictions:

Allow only admins to view email logs.

[ ] Error Logging:

Log all failed email sends with detailed error messages.

---

Optimized Workflow

1. Define Core API Endpoints:

Start with essential functionality (e.g., task creation, email sending).

2. Test APIs Incrementally:

Write unit tests for each endpoint before adding complexity.

3. Design Frontend with Wireframes:

Create simple wireframes to align on dashboard structure.

4. Iterate and Optimize:

Gather feedback and improve API performance and UI usability.

---

This checklist ensures the interfaces for Orgo are secure, scalable, and user-friendly. Would you like to focus on APIs, dashboards, or both for implementation?