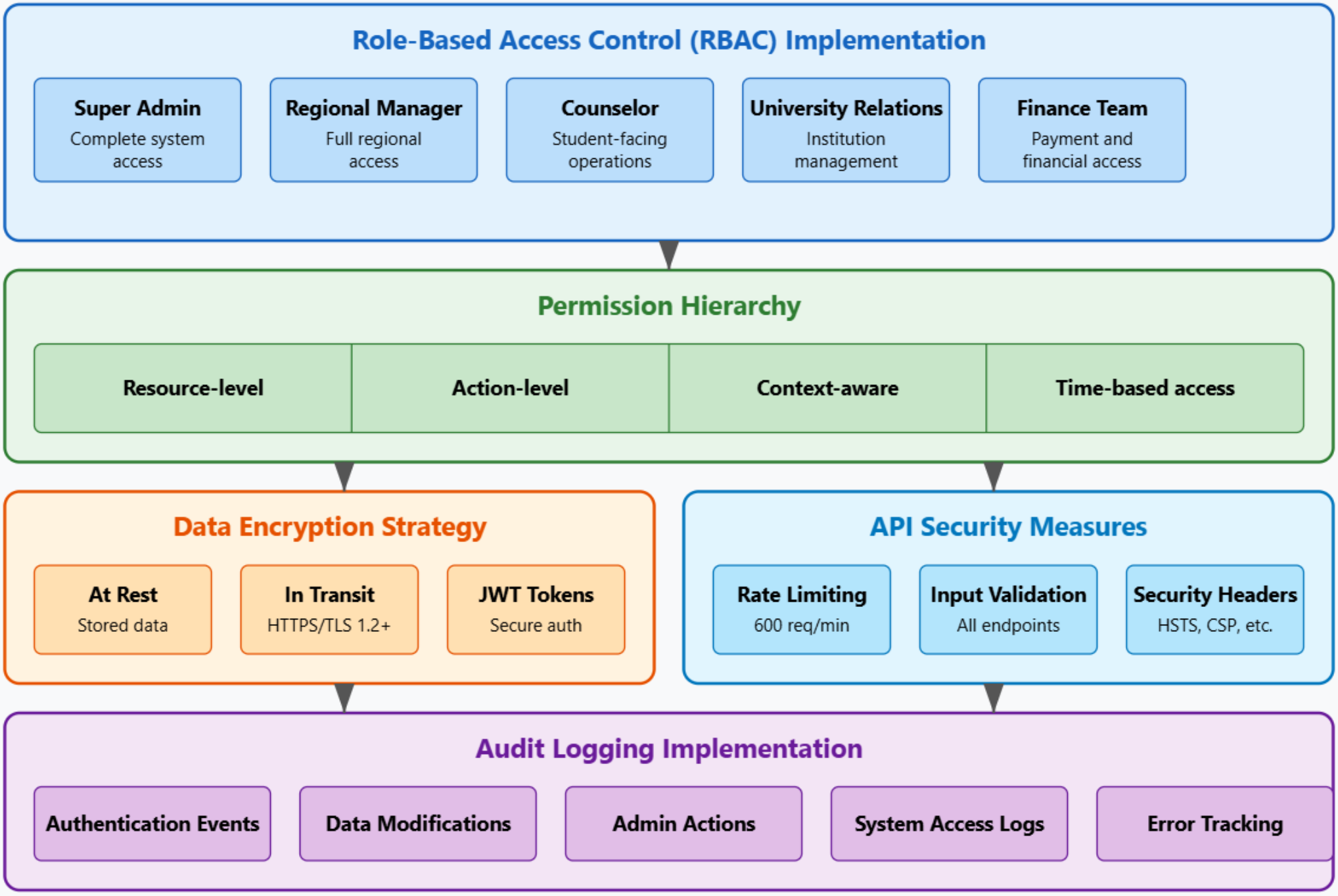
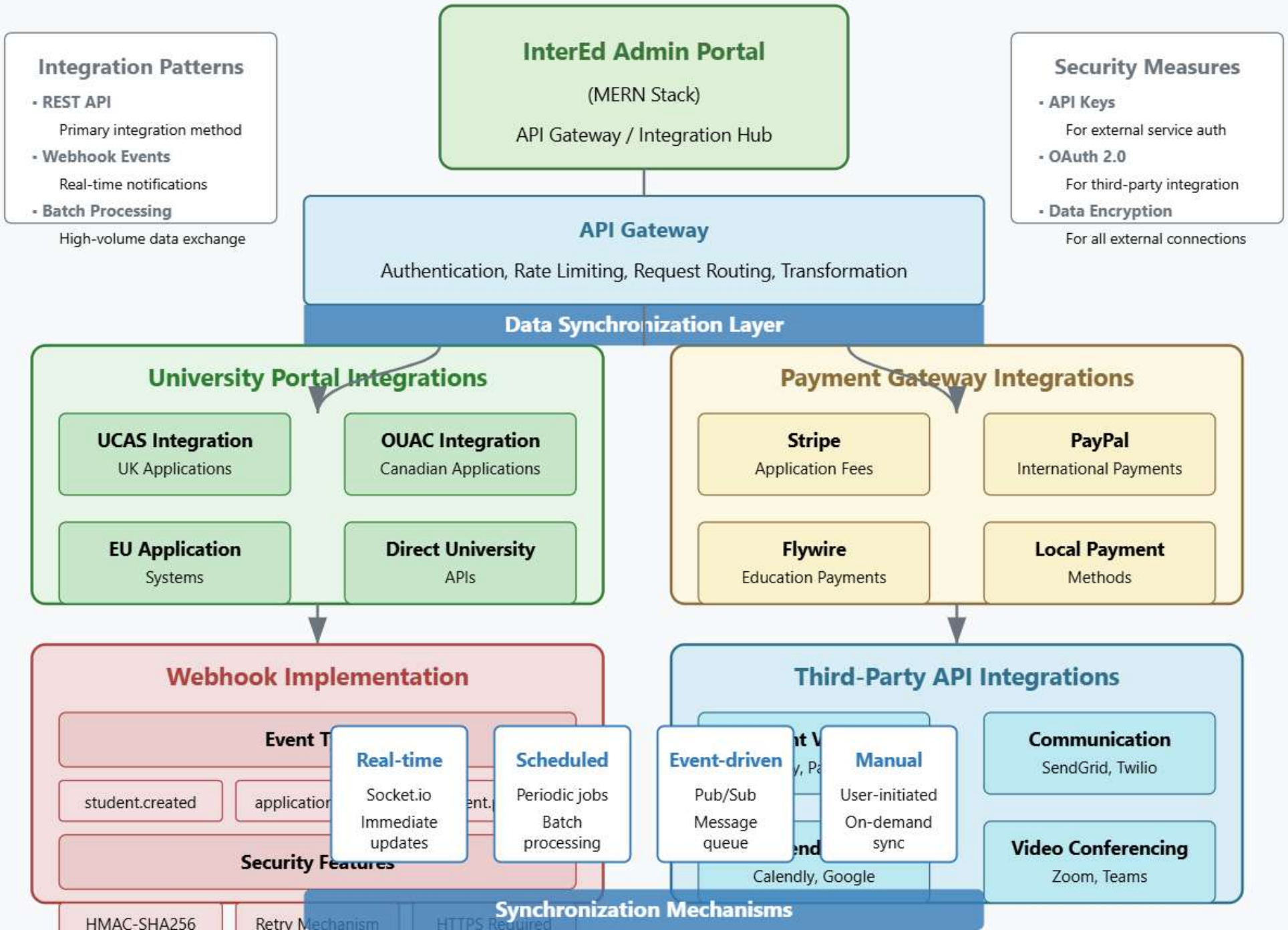
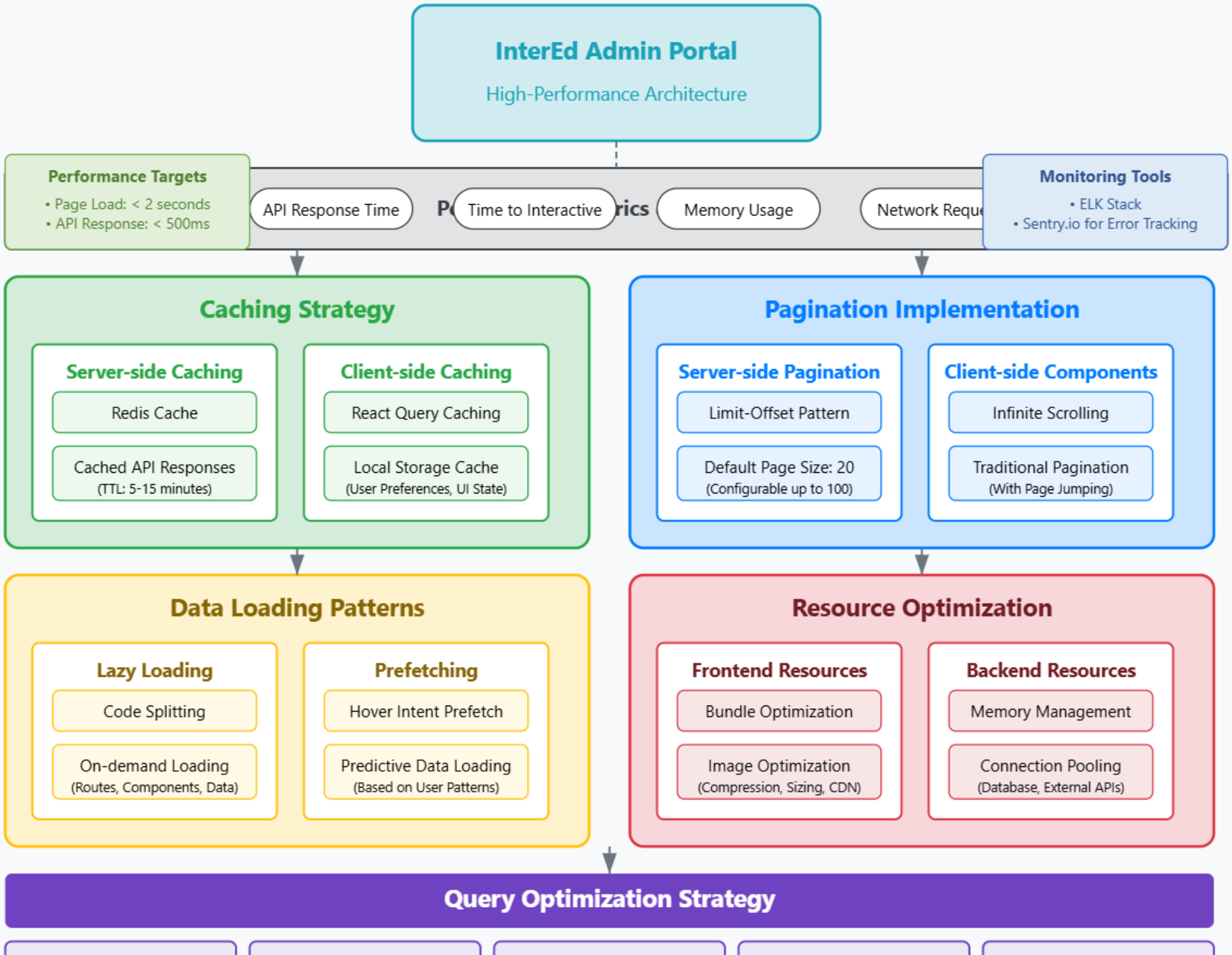
InterEd Admin Portal Security Architecture



InterEd Admin Portal Integration Architecture

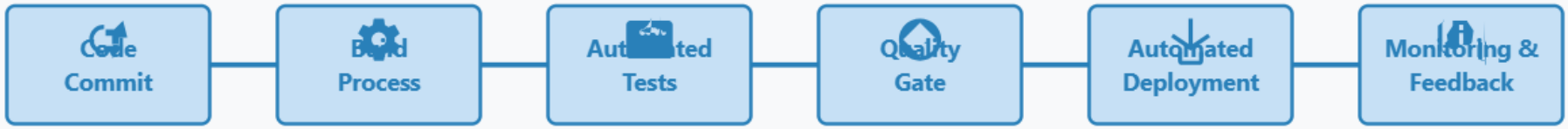


InterEd Admin Portal Performance Optimization Strategy



InterEd Admin Portal DevOps Pipeline

Continuous Integration / Continuous Deployment Workflow



Environment Configuration

Development
- Docker-based Local Env
- Continuous Integration
- Feature Branches

QA/Testing
- Isolated Test Environment
- Test Data Management
- Automated Test Suite

Staging
- Production-like Environment
- UAT Testing
- Performance Testing

Production
- Highly Available Infra
- Load Balancing
- Auto-scaling

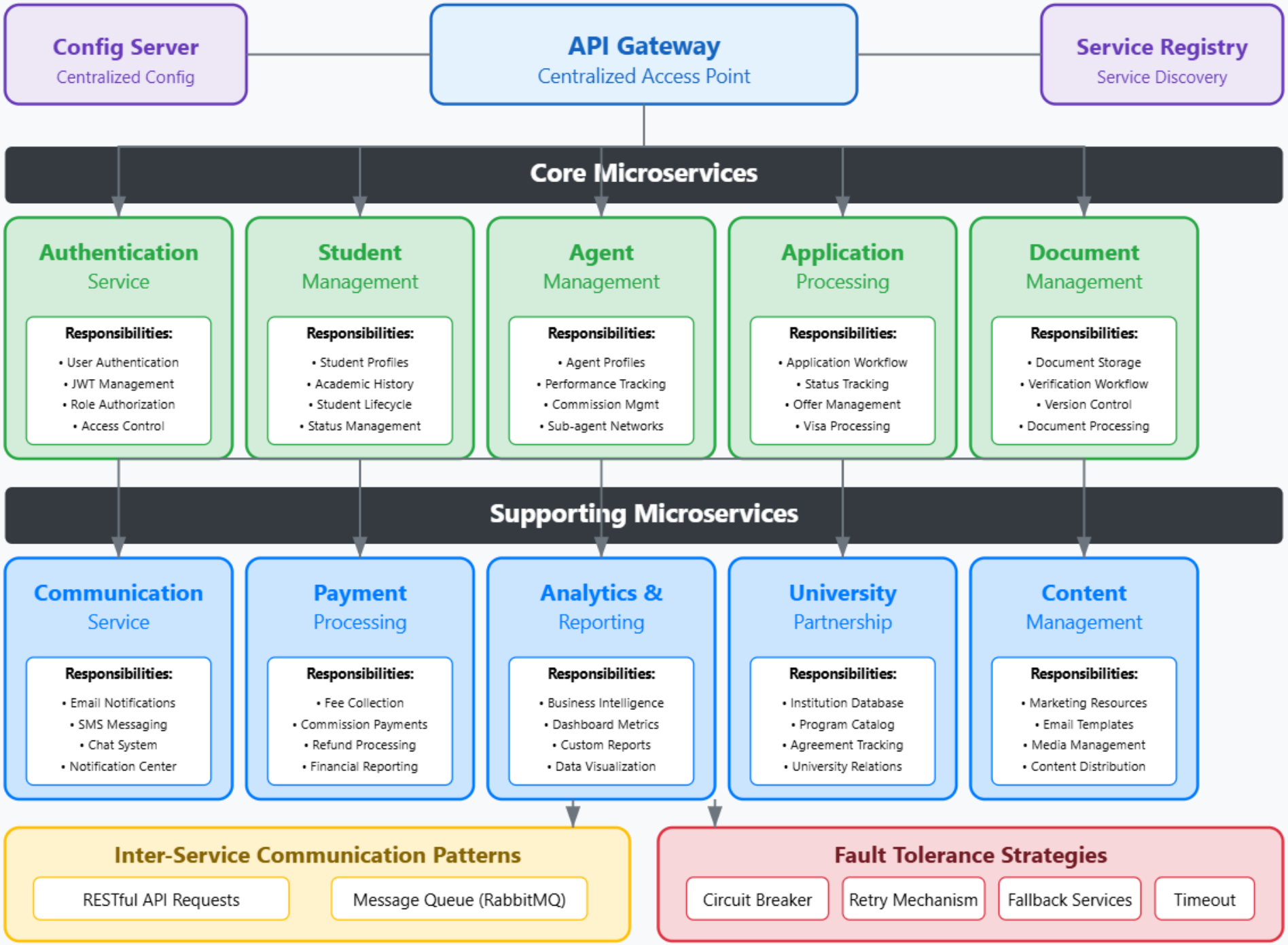
Deployment Strategy

CI/CD Tools
- GitHub Actions
- Docker Containerization
- Kubernetes Orchestration

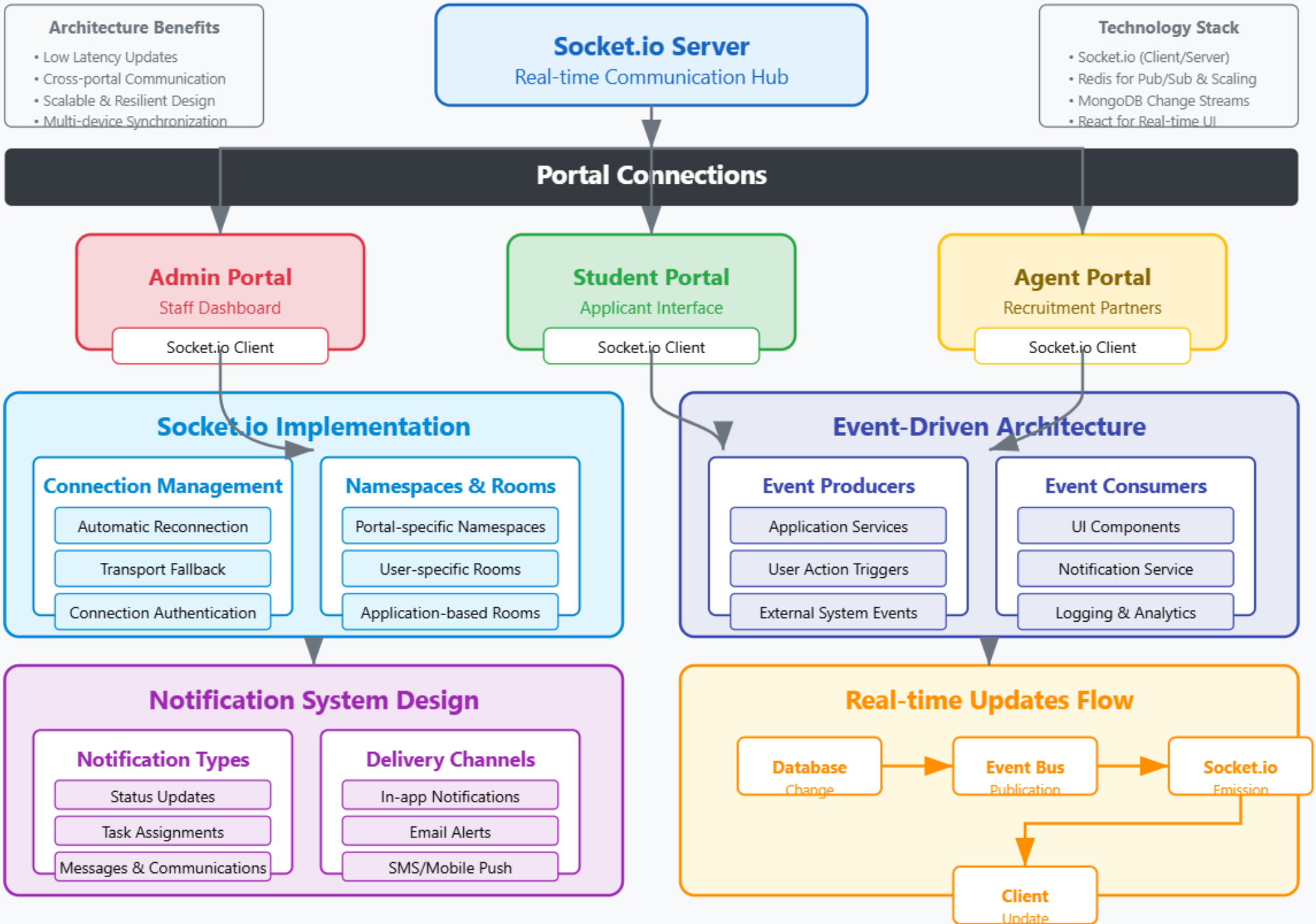
Deployment Techniques
- Blue-Green Deployment
- Canary Releases
- Automated Rollback Capability

Disaster Recovery

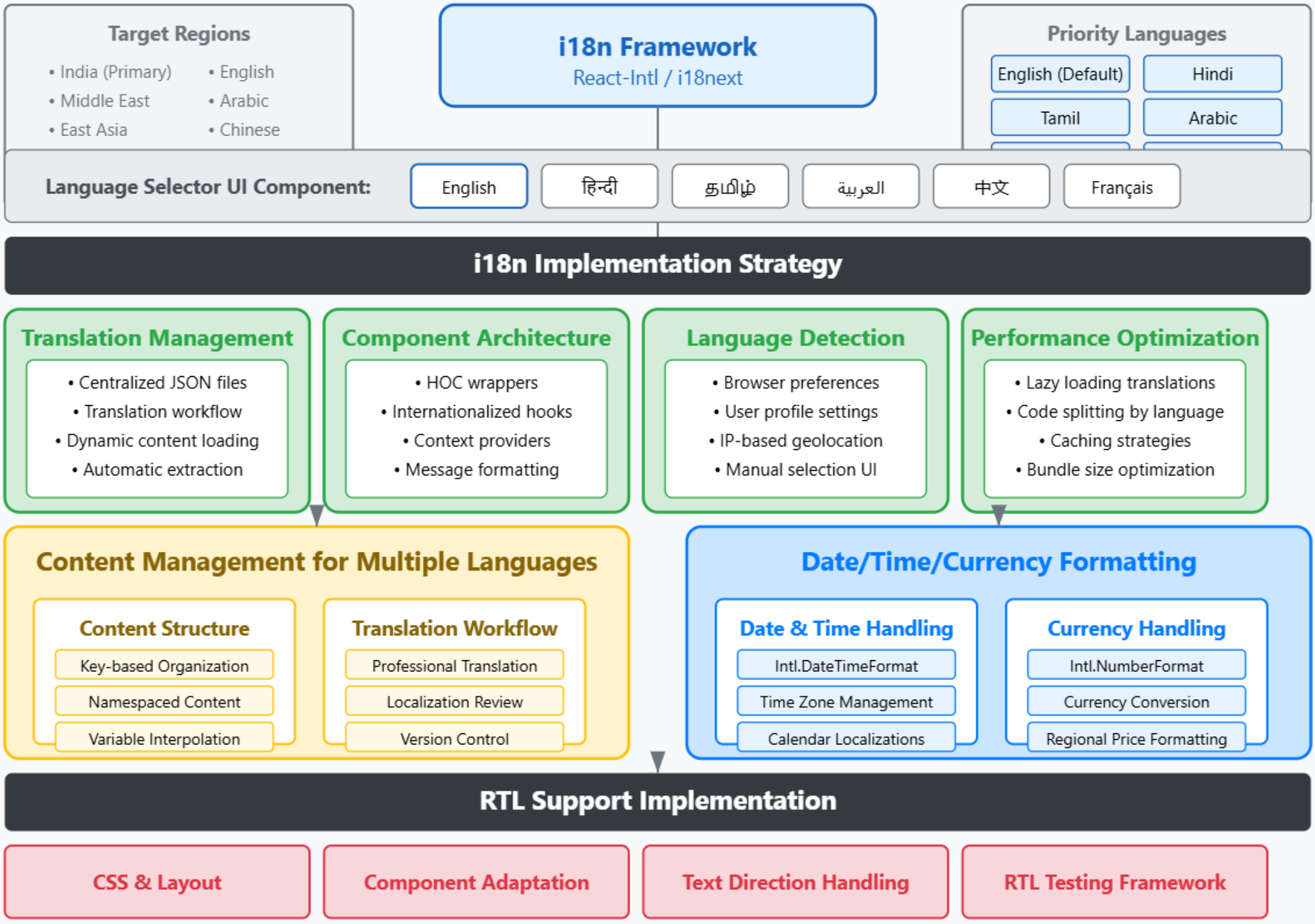
InterEd Admin Portal Microservices Architecture



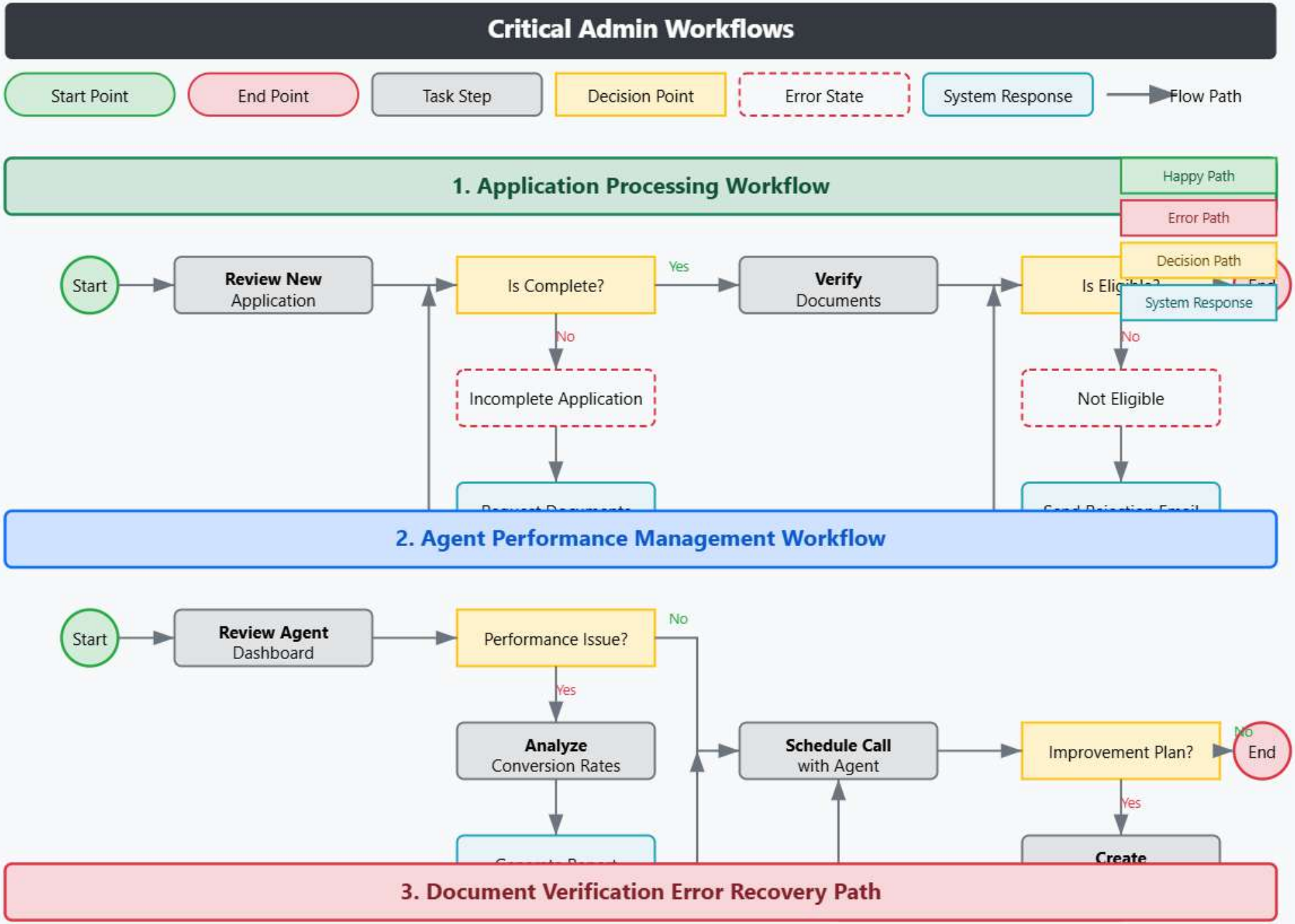
InterEd Admin Portal Real-time Communication Architecture



InterEd Admin Portal Internationalization Architecture

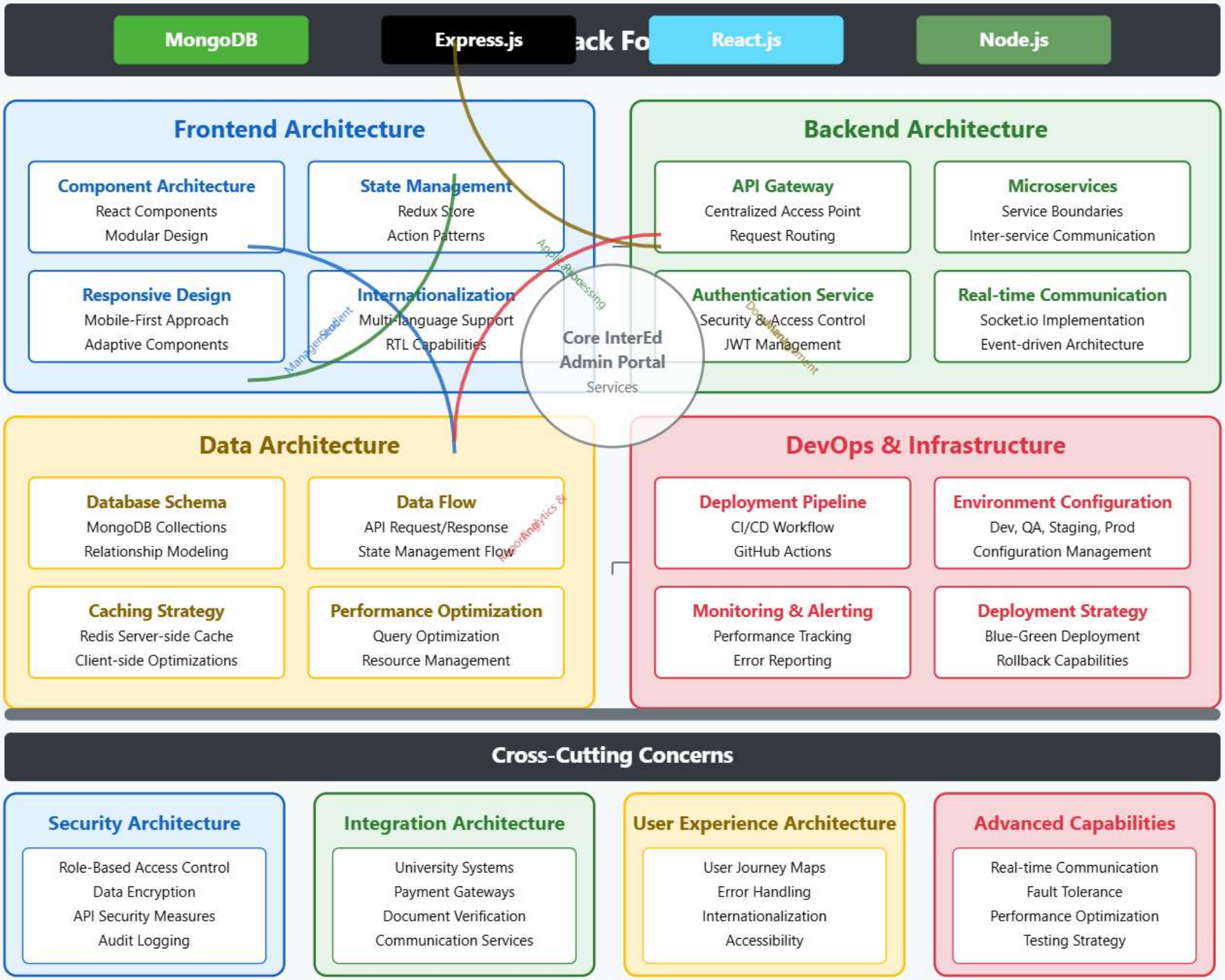


InterEd Admin Portal User Journey Maps

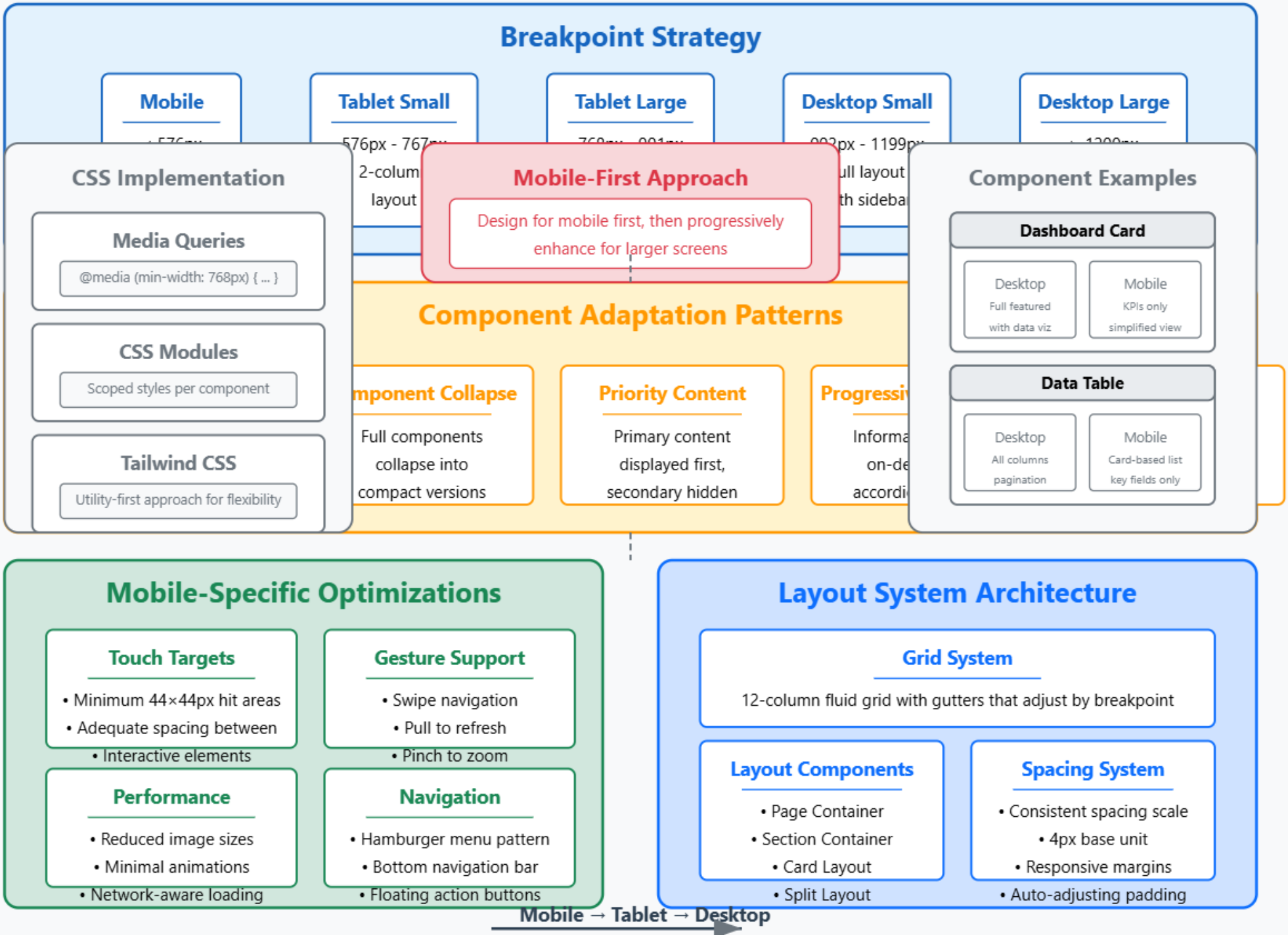


Additional workflow detailed here

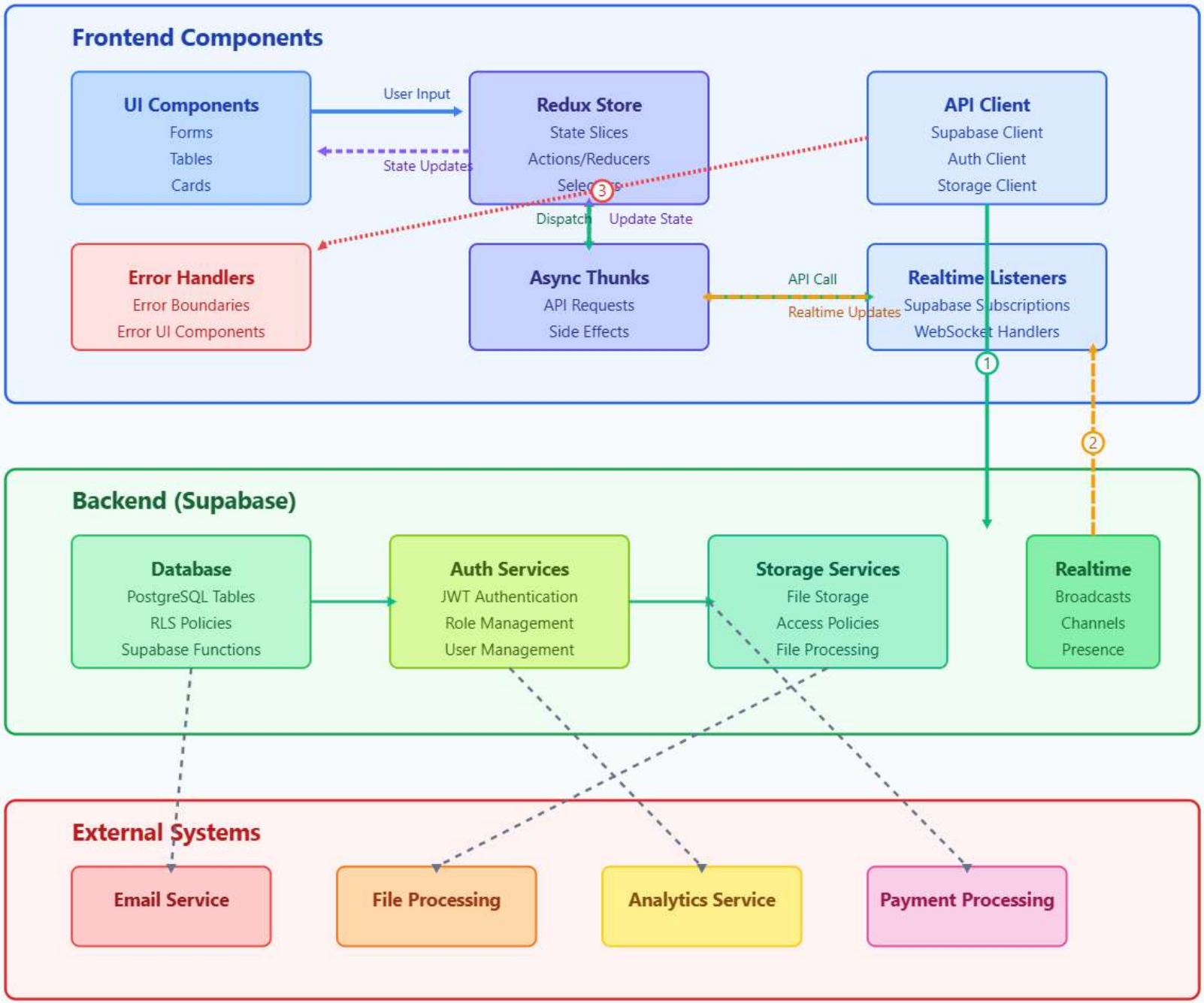
InterEd Admin Portal: Comprehensive Architecture



InterEd Admin Portal Responsive Design System



InterEd Admin Portal - Data Flow Architecture



Flow Legend

- User Input
- State Update
- API Request
- Realtime
- Error Flow

Flow Sequences

1. API Request Flow
2. Realtime Flow
3. Error Flow

Key Processes

Form Submission

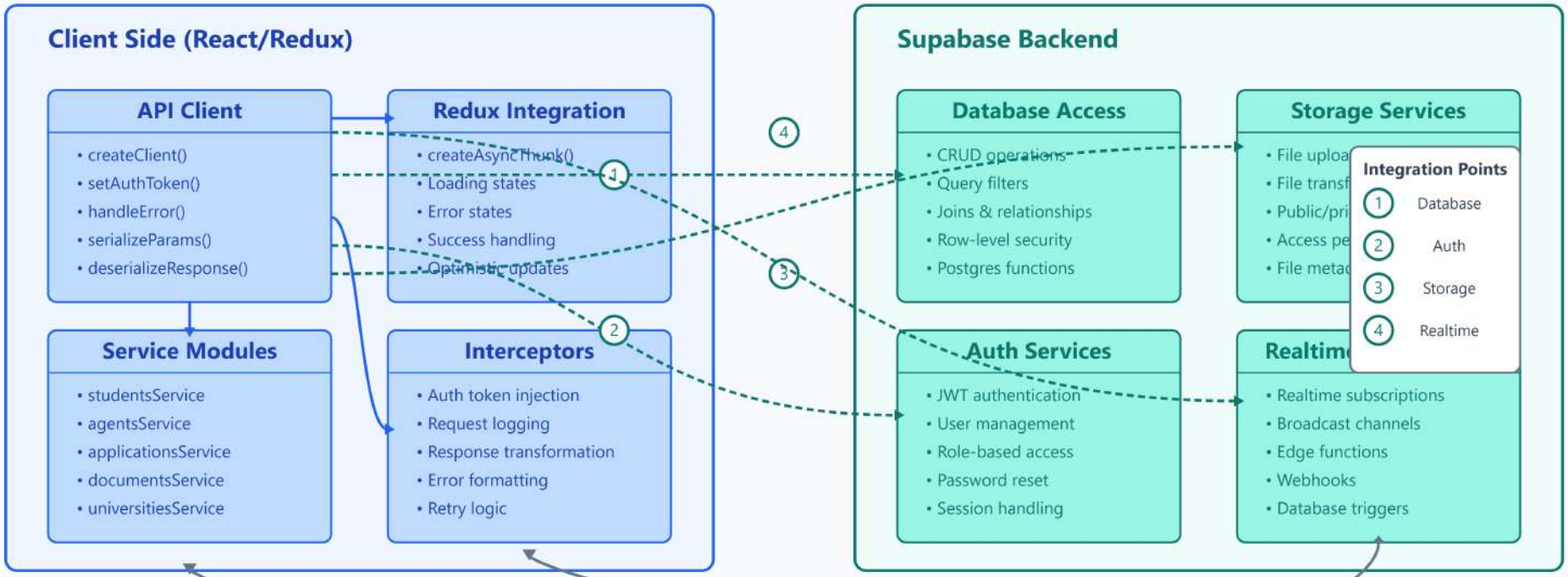
1. User input validation
2. Redux thunk dispatch
3. API client request
4. Success/Error handling
5. State update

Realtime Updates

1. Subscribe to changes
2. Receive websocket event
3. Parse payload
4. Update Redux store
5. UI rerender

Error Handling

InterEd Admin Portal - API Architecture



API Flow Patterns

Standard CRUD Pattern

```
// Service layer function
export const getStudents = async (filters = {}) => {
let query = supabase.from('students').select('*');

// Apply filters
if (filters.status) {
query = query.eq('status', filters.status);
}

const { data, error } = await query;

if (error) throw new APIError(error.message, error.code);
return data;
};
```

Optimistic Updates Pattern

```
// Redux thunk with optimistic update
export const updateStudent = createAsyncThunk(
'students/updateStudent',
async (student, { dispatch, rejectWithValue, getState }) => {
// Save original for rollback if needed
const originalStudent = getState().students
.entities[student.id];

try {
// Optimistically update UI
dispatch(studentsAdapter.upsertOne(student));

// Actual API call
const result = await studentsService.updateStudent(student);
```

Realtime Subscription Pattern

```
// Setup realtime subscriptions
export const setupRealtimeSubscriptions = (dispatch) => {
// Subscribe to application status changes
const subscription = supabase
.from('applications')
.on('UPDATE', (payload) => {
// Dispatch to Redux
dispatch(updateApplication(payload.new));

// Trigger notifications if status changed
if (payload.new.status !== payload.old.status) {
dispatch(addNotification({
message: `Application status updated to ${payload.new.status}`
}));
```

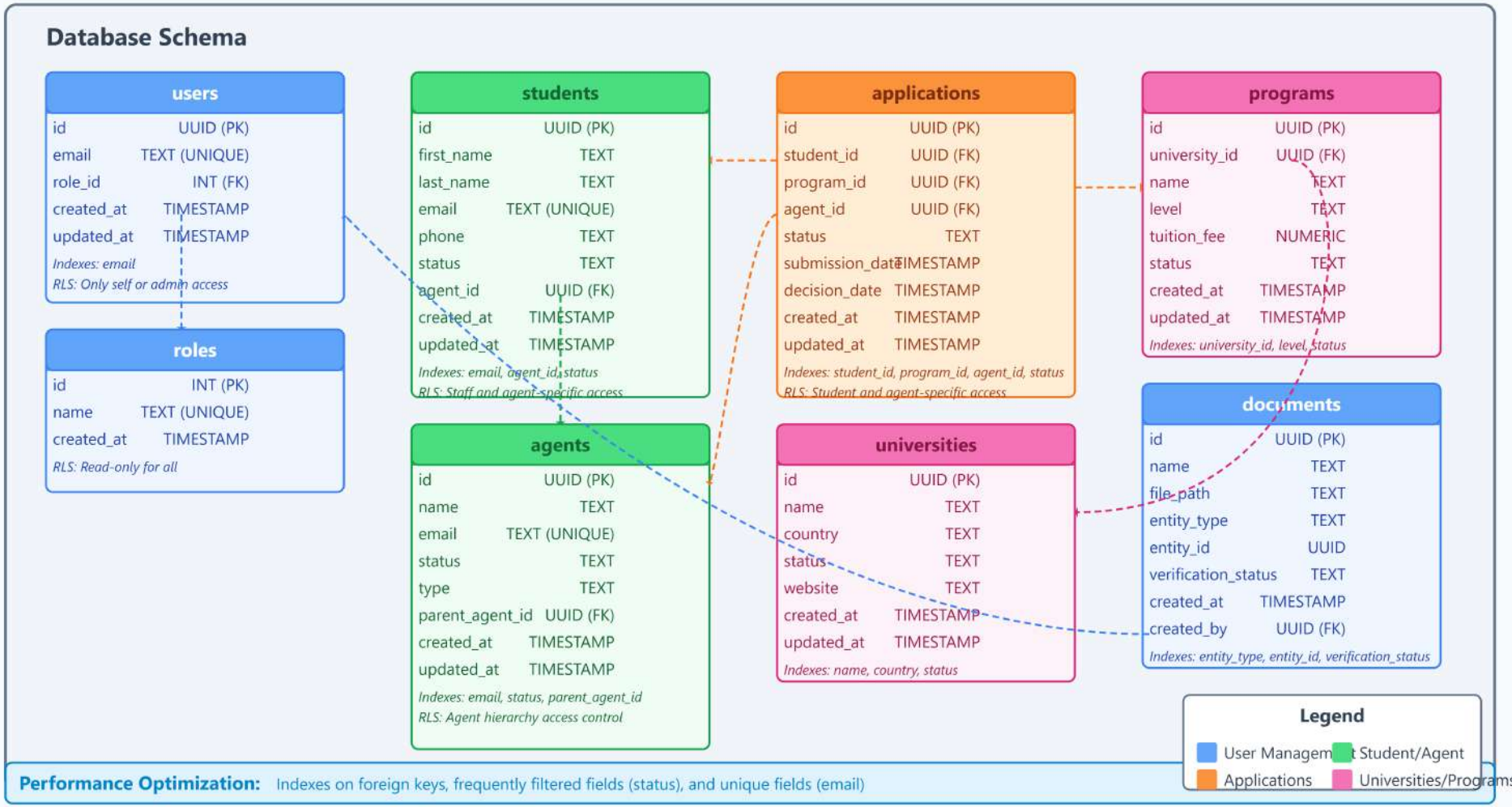
Error Handling Strategy

- Centralized Error Handling
- Error Classification & Retry Logic

Caching Strategy

- Query Result & Entity Caching

InterEd Admin Portal - Database Schema Architecture



Row Level Security Policies

- Users: Only self or admin can view/edit user data
- Students: Staff and associated agent access only
- Applications: Student, agent, and staff access
- *Documents: Entity-based access control with ownership policies*

Data Validation & Migration Strategy

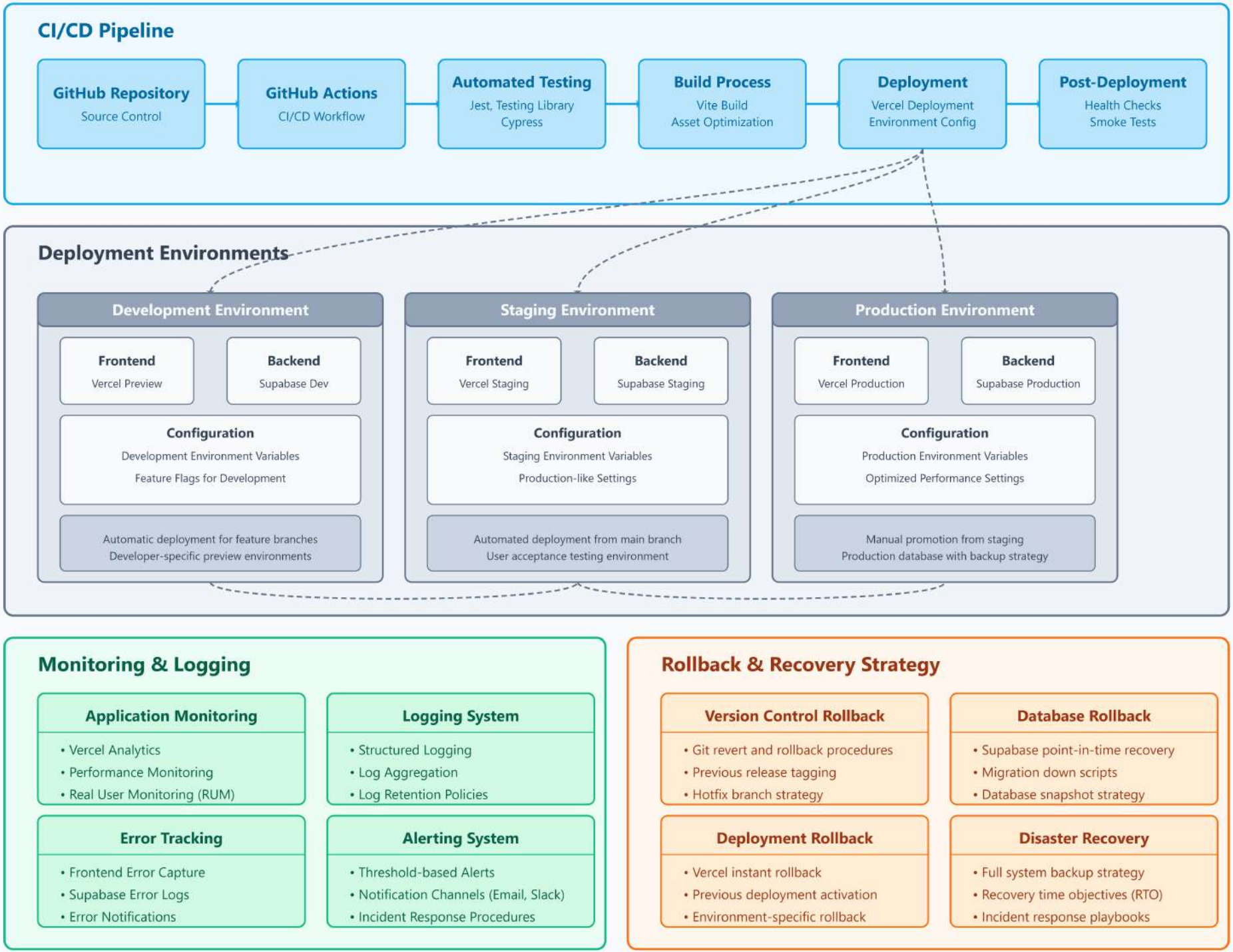
Validation Rules

- Email format validation
- Status field enums
- Required field constraints
- Foreign key constraints
- Numeric range validation
- Timestamp validations

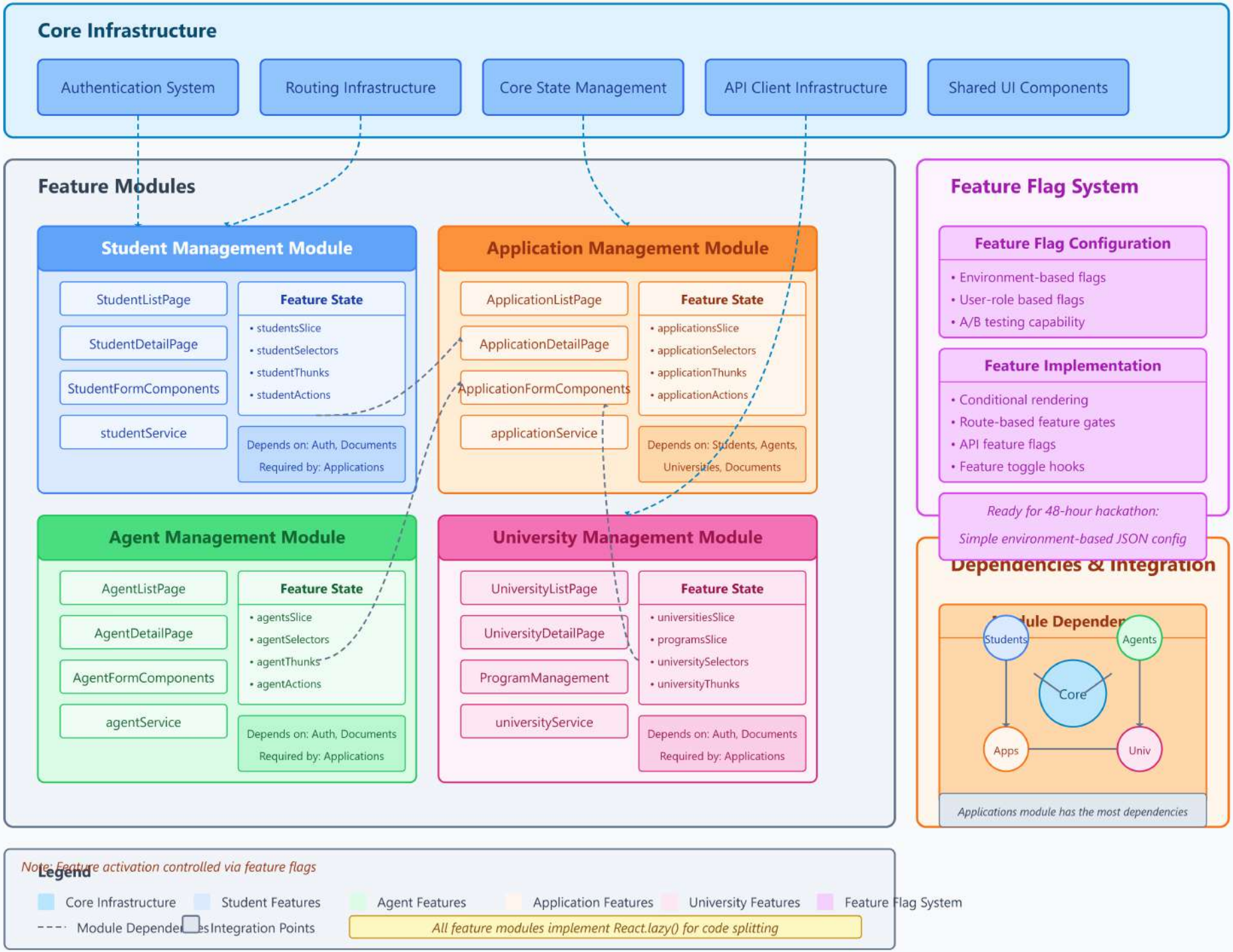
Migration Strategy

- Version-controlled migrations
- Up/down migration scripts
- Sequential numbering
- Transaction-based changes
- Seed data management
- CI/CD pipeline integration

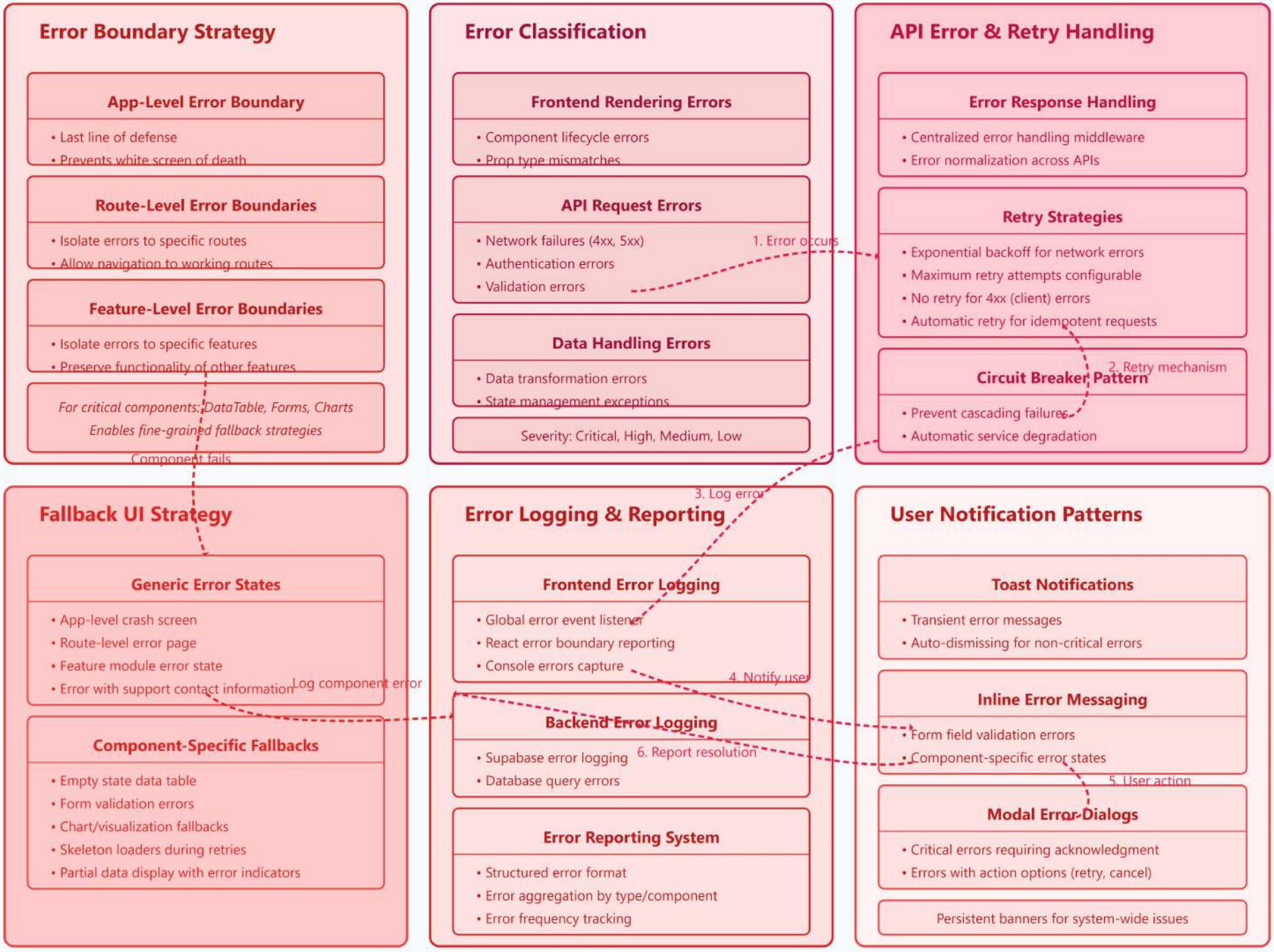
InterEd Admin Portal - Deployment Architecture



InterEd Admin Portal - Feature Module Architecture



InterEd Admin Portal - Error Handling Architecture



InterEd Admin Portal - Testing Architecture