SciCon: The Ultimate Scientific Conference Management Ecosystem

Comprehensive Technical & Functional Specification

DAW2 Module – L3-TI – Academic Year 2025/2026

Team Development Report

January 11, 2026

Abstract

SciCon is an enterprise-grade full-stack platform engineered to revolutionize scientific conference administration. This document provides an exhaustive technical specification covering the React.js + Django REST Framework architecture, four distinct user dashboards, comprehensive API routing, database schema design, and the security mechanisms protecting academic integrity. The platform automates high-friction administrative tasks including paper submission workflows, double-blind peer reviews, and automated certificate generation while maintaining a professional user experience through Shadcn/UI components and Tailwind CSS styling.

Contents

1 Executive Summary **3**

2 Full-Stack Architecture **3**
2.1 Technology Stack . . . 3
2.2 System Actors and Role-Based Access Control . . . 3

3 Frontend Architecture **3**
3.1 React Application Structure . . . 3
3.2 Shadcn/UI Component System . . . 4
3.3 Role-Specific Sidebars . . . 4

4 Backend Implementation **4**
4.1 Django Data Models . . . 4
4.2 Review Scoring and Consensus Logic . . . 5
4.3 REST API Endpoints . . . 6
4.4 Permission Classes . . . 6

5 Four Primary User Dashboards **6**
5.1 A. Organizer Dashboard . . . 6
5.2 B. Author Dashboard . . . 7
5.3 C. Reviewer Dashboard . . . 7
5.4 D. Participant Dashboard . . . 7

6 Registration Workflow **7**

7 Certificate Generation System **8**

8 Event Statistics and Analytics **9**

9 Security Architecture **10**
9.1 Authentication Flow . . . 10
9.2 Authorization Strategy . . . 10

10 File Organization **10**
10.1 Backend Structure . . . 10
10.2 Frontend Structure . . . 10

11 API Routing Map **11**

12 Development Workflow **11**
12.1 Team Organization . . . 11
12.2 Tools and Practices . . . 11

13 Deployment Strategy **11**
13.1 Local Development . . . 11
13.2 Production Deployment . . . 12

14 Key Technical Achievements **12**

15 Deliverables Checklist **12**

16 Conclusion **12**

1 Executive Summary

SciCon bridges the critical gap between event organizers and the academic community by providing a unified platform for managing the complete lifecycle of scientific conferences, seminars, and workshops in the health domain. The platform addresses:

- **Administrative Overhead:** Eliminates manual paper tracking, reviewer assignment, and certificate generation
- **Peer Review Complexity:** Implements structured double-blind review with automatic consensus scoring
- **Participant Engagement:** Provides real-time Q&A, interactive surveys, and workshop management
- **Data Consistency:** Enforces Django ORM constraints and JWT-based authentication

2 Full-Stack Architecture

2.1 Technology Stack

Layer	Technology	Purpose
Frontend SPA	React 18+ with Vite	UI rendering, routing, state management
UI Framework	Shadcn/UI + Tailwind CSS	Accessible component system
Backend API	Django 4.2+ & DRF	REST endpoints, business logic
Authentication	JWT (SimpleJWT)	Stateless token-based security
Database	MySQL/PostgreSQL	Persistent data storage
Build Tool	Vite with HMR	Fast development & hot reloading
Icons	Lucide React	Consistent vector iconography
HTTP Client	Axios	API communication with interceptors

2.2 System Actors and Role-Based Access Control

The platform implements RBAC with four primary user domains:

Role	Responsibilities	Dashboard
Organizer	Event creation, reviewer assignment, participant management	OrganizerDashboard
Author	Paper submission, review tracking, certificate access	AuthorDashboard
Reviewer	Paper evaluation, scoring, decision-making	ReviewerDashboard
Participant	Event discovery, registration, attendance tracking	ParticipantDashboard

3 Frontend Architecture

3.1 React Application Structure

The frontend is organized using feature-based architecture with clear separation of concerns:

```
src/
          features/
                auth/              # Authentication & Access Control
                dashboard/         # Role-specific dashboards
                events/           # Event management & discovery
                submissions/       # Paper submission workflows
                users/            # Participant management
          components/
                layout/           # Layout components (sidebar, navbar)
                ui/               # Shadcn/UI atomic components
          services/
                api.js            # Centralized API client
          hooks/
              use-toast.js      # Toast notification system
```

3.2 Shadcn/UI Component System

The design system provides accessible, composable components:

Component	**Application**
Card	Dashboard panels, event listings, submission cards
Button	Actions (Create, Submit, Approve, Delete) with variants
Badge	Status indicators (Open Call, Draft, Ongoing, Completed)
Tabs	Dashboard navigation (Overview, Registrations, Certificates)
Select/Dropdown	Role selection, event filtering, reviewer assignment
Table	Submission listings, participant management
Toast	Non-blocking notifications (success, error, info)
Avatar	User profiles with initials or images

3.3 Role-Specific Sidebars

Each user role has a customized sidebar for navigation:

```
// OrganizerSidebar.jsx
<SidebarProvider>
  <Sidebar>
    <SidebarContent>
      <Link to="/organizer/events">Create Event</Link>
      <Link to="/organizer/assign-reviewers">
        Manage Reviewers
      </Link>
      <Link to="/organizer/participants">
        View Participants
      </Link>
    </SidebarContent>
  </Sidebar>
  <SidebarInset>
    {/* Main content */}
  </SidebarInset>
</SidebarProvider>
```

4 Backend Implementation

4.1 Django Data Models

The database schema implements 14 interconnected models representing the conference domain:

```
class Event(models.Model):
    STATUS_CHOICES = [
        ('draft', 'Draft'),
        ('open_call', 'Call for Papers Open'),
        ('reviewing', 'Under Review'),
        ('program_ready', 'Program Ready'),
        ('ongoing', 'Ongoing'),
        ('completed', 'Completed'),
    ]

    organizer = ForeignKey(User, on_delete=models.CASCADE)
    title = CharField(max_length=300)
    start_date = DateField()
    end_date = DateField()
    submission_deadline = DateTimeField()
    scientific_committee = ManyToManyField(User)
    status = CharField(max_length=20, choices=STATUS_CHOICES)
```

```
class Submission(models.Model):
    STATUS_CHOICES = [
        ('pending', 'Pending'),
        ('under_review', 'Under Review'),
        ('accepted', 'Accepted'),
        ('rejected', 'Rejected'),
        ('revision_requested', 'Revision Requested'),
    ]

    event = ForeignKey(Event, related_name='submissions')
    author = ForeignKey(User, related_name='submissions')
    title = CharField(max_length=300)
    abstract = TextField()
    assigned_reviewers = ManyToManyField(User)
    status = CharField(max_length=20, choices=STATUS_CHOICES)
```

4.2 Review Scoring and Consensus Logic

The peer review system implements automatic decision-making:

```
class Review(models.Model):
    submission = ForeignKey(Submission, related_name='reviews')
    reviewer = ForeignKey(User, related_name='reviews')

    # Three-dimensional scoring (1-5 scale)
    relevance_score = IntegerField(
        validators=[MinValueValidator(1), MaxValueValidator(5)]
    )
    quality_score = IntegerField(
        validators=[MinValueValidator(1), MaxValueValidator(5)]
    )
    originality_score = IntegerField(
        validators=[MinValueValidator(1), MaxValueValidator(5)]
    )

    decision = CharField(
        choices=[('accept', 'Accept'), ('reject', 'Reject'),
                 ('revision', 'Revision Required')]
    )
```

Automated Consensus Logic:

```
if reviews_count >= 2:
    avg_score = (
```

```python
        Avg('relevance_score') +
        Avg('quality_score') +
        Avg('originality_score')
    ) / 3

    if avg_score >= 4.0:
        submission.status = 'accepted'
    elif avg_score < 2.5:
        submission.status = 'rejected'
    else:
        submission.status = 'revision_requested'

    submission.save()
    notify_author(submission)
```

4.3 REST API Endpoints

Endpoint	Method	Purpose
/api/auth/register/	POST	User registration
/api/auth/login/	POST	JWT token acquisition
/api/events/	GET/POST	List/create events
/api/events/<id>/	GET/PUT/DELETE	Event operations
/api/submissions/	GET/POST	List/submit papers
/api/assign-reviewers/	POST	Assign reviewers to submission
/api/reviews/	GET/POST	Submit reviews
/api/registrations/	GET/POST	Manage registrations
/api/certificates/generate/	POST	Auto-generate certificates
/api/surveys/	GET/POST	Create/respond to surveys

4.4 Permission Classes

Custom permission logic enforces access control:

```python
class IsEventOrganizer(permissions.BasePermission):
    """Only the event organizer can modify"""
    def has_object_permission(self, request, view, obj):
        if request.method in permissions.SAFE_METHODS:
            return True
        if hasattr(obj, 'organizer'):
            return obj.organizer == request.user
        if hasattr(obj, 'event'):
            return obj.event.organizer == request.user
        return False

class IsReviewerOrOrganizer(permissions.BasePermission):
    """Reviewers and organizers can access reviews"""
    def has_permission(self, request, view):
        return (request.user.is_authenticated and
                request.user.role in ['reviewer', 'organizer'])
```

5 Four Primary User Dashboards

5.1 A. Organizer Dashboard

The command center for event administration:

Section	Features
Overview Stats	Total events, active registrations, pending submissions
Event Portfolio	Tabular view with inline Create/Edit/Publish/Delete actions
Reviewer Management	Assign scientific committee to submissions by domain
Participant Management	Searchable delegate list with payment status filtering

5.2 B. Author Dashboard

The contribution portal for researchers:

Section	Features
Submission Tracking	Visual timeline (Pending → Under Review → Accepted/Rejected)
Review Feedback	Direct access to reviewer comments and scoring metrics
New Submission	Multi-step form for abstract, keywords, PDF upload
Certificate Center	Auto-access to "Presentation" certificates upon acceptance

5.3 C. Reviewer Dashboard

The evaluation hub for scientific committee:

Section	Features
Assigned Reviews	Prioritized list of papers to evaluate
Scoring Interface	Evaluate on Relevance, Quality, Originality (1-5 scale)
Expert Rating	Profile-level metric reflecting contribution volume
Review History	Archive of all past evaluations

5.4 D. Participant Dashboard

The attendance hub for delegates:

6 Registration Workflow

The registration process demonstrates the RESTful architecture in action:

```
// Frontend: EventDetailsPage.jsx
const handleRegister = async () => {
  if (!canRegister) return;

  try {
    const response = await api.post(
      `/api/events/${eventId}/registrations/`,
      { registration_type: 'participant' }
    );

    toast({
      title: "Registration Successful",
```

Section	**Features**
Event Discovery	Unified catalog with advanced search and filtering
Registration History	Confirmed attendance and payment status records
Certificate Vault	Downloadable "Participation" certificates in PDF
Attended Events	Archive of completed conferences

```
      description: "You are now registered for this event"
    });

    setRegistered(true);
  } catch (error) {
    toast({
      title: "Registration Failed",
      description: error.response?.data?.detail,
      variant: "destructive"
    });
  }
};
```

Backend validation:

```
class RegistrationListCreateView(generics.ListCreateAPIView):
    permission_classes = [IsAuthenticated]

    def perform_create(self, serializer):
        event_id = self.kwargs.get('event_id')

        # Check for duplicate registration
        existing = Registration.objects.filter(
            event_id=event_id,
            user=self.request.user
        ).exists()

        if existing:
            raise ValidationError("Already registered")

        serializer.save(
            user=self.request.user,
            event_id=event_id
        )
```

7 Certificate Generation System

Automatic certificate creation for multiple categories:

```
@api_view(['POST'])
@permission_classes([IsOrganizer])
def generate_certificates(request, event_id):
    event = Event.objects.get(id=event_id)
    generated_count = 0

    # Participation certificates
    for registration in event.registrations.all():
        cert_type = 'participation'
        if registration.registration_type == 'speaker':
            cert_type = 'presentation'

        cert, created = Certificate.objects.get_or_create(
            event=event,
            user=registration.user,
            certificate_type=cert_type
        )
        if created:
            generated_count += 1

    # Committee certificates
    for member in event.scientific_committee.all():
        cert, created = Certificate.objects.get_or_create(
            event=event,
            user=member,
            certificate_type='committee'
        )
        if created:
            generated_count += 1

    return Response({
        'message': f'{generated_count} certificates created',
        'total': event.certificates.count()
    })
```

8 Event Statistics and Analytics

Real-time dashboard metrics computation:

```
@api_view(['GET'])
@permission_classes([IsOrganizer])
def event_statistics(request, event_id):
    event = Event.objects.get(id=event_id)
    submissions = event.submissions.all()
    registrations = event.registrations.all()

    stats = {
        'total_submissions': submissions.count(),
        'accepted': submissions.filter(
            status='accepted'
        ).count(),
        'rejection_rate': (
            submissions.filter(status='rejected').count() /
            submissions.count() * 100
            if submissions.count() > 0 else 0
        ),
        'registrations_by_country': list(
            registrations.values('user__country')
            .annotate(count=Count('id'))
            .order_by('-count')
        )
    }

    return Response(stats)
```

9 Security Architecture

9.1 Authentication Flow

1. User registers with email and password
2. Password hashed via Django's PBKDF2 algorithm
3. JWT token generated on successful login
4. Refresh token allows extended session management
5. Token embedded in request headers: `Authorization: Bearer <token>`

9.2 Authorization Strategy

- **Method-level:** DRF permission classes on API views
- **Object-level:** Custom `has_object_permission` checks
- **Field-level:** Read-only fields prevent unauthorized modification
- **Unique constraints:** Database-enforced uniqueness (e.g., one review per reviewer per submission)

10 File Organization

10.1 Backend Structure

```
backend/
          api/
                models.py           # 14 interconnected models
                serializers.py      # Data serialization logic
                views.py            # API endpoints & business logic
                permissions.py      # RBAC implementation
                urls.py             # REST routing (50+ endpoints)
          backend/
                settings.py         # Django configuration
                wsgi.py             # Production entry point
          media/                  # User uploads (papers, images)
```

10.2 Frontend Structure

```
frontend/src/
          features/               # Feature modules
                auth/              # Login, register, auth logic
                dashboard/         # 4 role-specific dashboards
                events/            # Event CRUD operations
                submissions/       # Paper workflow
                users/             # Participant management
          components/
                layout/            # Sidebars, navigation
                ui/                # Shadcn/UI atoms
          services/
                api.js             # Axios with JWT handling
          styles/
              global.css         # Tailwind configuration
```

11 API Routing Map

Endpoint	Feature	Auth
`POST /auth/register`	User onboarding	AllowAny
`POST /auth/login`	JWT token acquisition	AllowAny
`GET /auth/profile`	User profile hydration	Authenticated
`GET /events`	Event listing	Authenticated
`POST /events`	Create event	Organizer
`GET /events/:id/statistics`	Event analytics	Organizer
`POST /submissions/:id/assign-reviewers`	Reviewer assignment	Organizer
`POST /reviews`	Submit review	Reviewer
`POST /registrations`	Register for event	Participant
`POST /certificates/generate`	Auto-generate certificates	Organizer
`POST /surveys/:id/results`	Survey analytics	Authenticated

12 Development Workflow

12.1 Team Organization

- **Project Manager:** Coordinates sprints, manages Trello/Notion board
- **Frontend Lead:** Manages React components, Shadcn/UI integration
- **Backend Lead:** Oversees Django models, serializers, API logic
- **Database Administrator:** Designs schema, migration management

12.2 Tools and Practices

- **Version Control:** Git/GitHub with feature branch workflow
- **Project Management:** Trello or Notion for sprint tracking
- **Communication:** Discord/Slack for real-time team coordination
- **Code Quality:** Automated linting, pre-commit hooks

13 Deployment Strategy

13.1 Local Development

```bash
# Backend setup
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python manage.py migrate
python manage.py runserver

# Frontend setup
cd frontend
npm install
npm run dev  # Vite development server
```

13.2 Production Deployment

- **Containerization:** Docker for consistent environments
- **App Server:** Gunicorn for Django WSGI serving
- **Reverse Proxy:** Nginx for static asset delivery
- **Database:** MySQL/PostgreSQL with automated backups
- **Environment:** Separate production/staging configurations

14 Key Technical Achievements

- **Consensus Scoring:** Automatic paper acceptance based on multi-reviewer scores
- **Role-Based UI:** Four distinct dashboards adapting to user role
- **Toast Notifications:** Non-blocking feedback for all user actions
- **JWT Security:** Stateless authentication with refresh token rotation
- **Atomic Components:** Shadcn/UI system ensuring design consistency
- **Vite HMR:** Hot module replacement for rapid development iteration

15 Deliverables Checklist

Complete source code (frontend + backend + migrations)

Technical documentation (schema diagrams, API docs)

Architecture decision records and technology justification

Project presentation (concept, architecture, live demo)

Public GitHub repository with README

Deployment instructions (Docker, environment setup)

16 Conclusion

SciCon represents a masterclass in modern full-stack development, successfully combining the robustness of Django with the interactivity of React. By prioritizing user experience through Shadcn/UI components and maintaining data consistency through Django's ORM and JWT authentication, the platform delivers a professional, reliable solution for scientific conference management.

The architecture scales from small workshops to large international congresses, with built-in support for multi-reviewer consensus, real-time notifications, and comprehensive analytics. The role-based dashboard system ensures each stakeholder (organizer, author, reviewer, participant) enjoys a tailored experience optimized for their specific workflow.

Project Duration: January 2025 – June 2026
Team Size: 5–6 students with defined technical roles
Technology Stack: React + Vite + Django 4.2+ + PostgreSQL + JWT
Code Repository: GitHub (public/private with team access)