Day 1 Slide Deck - API Foundation & Design

"Build the Blueprint"

Slide 1: Welcome & Day Overview

```
🎯 Day 1: API Foundation & Design
Building Production-Ready API Blueprints

Today's Journey:
├── 10:15 AM - API Reality Check & Types
├── 11:00 AM - Domain-Driven API Architecture
├── 1:00 PM - Professional API Design
├── 1:45 PM - Production Error Handling
├── 2:45 PM - gRPC vs REST Deep Dive
└── 3:30 PM - Legacy Integration Reality

Outcome: Complete TaskFlow API design ready for implementation
```

Slide 2: Ground Rules & Expectations

```
🎯 How We Learn APIs

80% Hands-On Activities
- No passive listening for more than 10 minutes
- Every concept = immediate practice
- Real code, real problems, real solutions

20% Strategic Discussion
- Senior engineer perspectives
- Production war stories
- Business impact thinking

Your Role: Active participant, not student
My Role: Guide, not lecturer
```

Slide 3: API Treasure Hunt Instructions

🎯 API Treasure Hunt - 10 Minutes

Your Mission: Explore 5 APIs, document findings

APIs to Explore:
1. JSONPlaceholder - Educational API
2. GitHub API - Enterprise API
3. OpenWeather - Commercial API
4. REST Countries - Simple API
5. Legacy SOAP Service - Problem API

Questions for Each:
- What makes this easy/hard to use?
- How is authentication handled?
- What's the response structure like?
- Would you want to integrate with this?

Tools: Postman, browser, your judgment

Slide 4: API Type Decision Matrix

When to Choose Each API Type

Scenario	REST	GraphQL	gRPC	WebSocket
Public API	✅ First	⚡ Mobile	❌ Complex	❌ Overkill
Internal Services	✅ CRUD	⚠️ Consider	✅ Fast	❌ Usually
Real-time	❌ Polling	⚠️ Subs	✅ Stream	✅ Perfect
Mobile Apps	✅ Standard	✅ Efficient	⚠️ Size	✅ Real-time
Microservices	✅ HTTP	❌ Complex	✅ Perf	❌ Wrong

Key Insight: Choose based on constraints, not preferences

Slide 5: Domain-Driven Design Introduction

🎯 Domain-Driven Design in APIs

The Problem:
❌ GET /api/v1/user-project-assignments?filter=active
❌ POST /api/v1/task-status-updates
❌ GET /api/v1/team-member-relationships

The Solution:
✅ GET /teams/{id}/projects?status=active
✅ POST /tasks/{id}/complete
✅ GET /teams/{id}/members

Principle: Your API should speak business language, not database language

---

Slide 6: TaskFlow Business Overview

🎯 TaskFlow: Our Domain Example

Business Context:
"A project management system for software teams"

Core Business Capabilities:
├── Team Management - organizing people
├── Project Planning - defining work
├── Task Execution - getting things done
├── User Identity - who can do what
└── Communication - keeping everyone informed

Question: How do these capabilities become API domains?

Slide 7: Domain Modeling Activity Setup

🎯 Domain Modeling Exercise - 20 Minutes

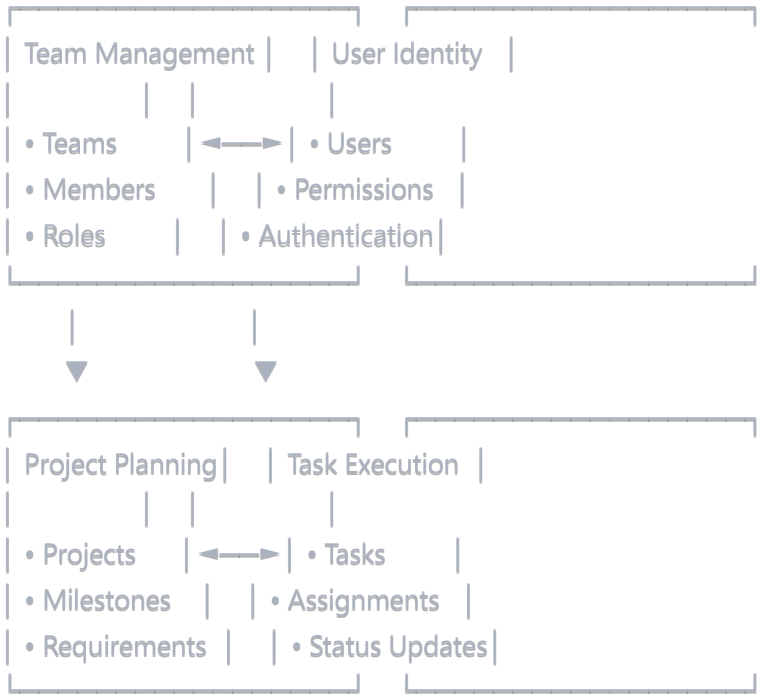
Materials:
- Excalidraw shared board
- Your domain knowledge
- Business thinking hat

Process:
1. Individual Brainstorm (5 min) - add entities in your color
2. Pair Discussion (8 min) - group and organize
3. Full Group Integration (5 min) - draw boundaries
4. Present & Defend (2 min each pair)

Goal: Map TaskFlow business domains to API structure

Slide 8: Domain Boundaries Visual

Domain-Driven TaskFlow Architecture



Each domain = potential microservice = API boundary

Slide 9: API Design Principles

🎯 REST API Design Principles

1. Resource-Based URLs
✅ /teams/123/projects
❌ /getProjectsByTeam?id=123

2. HTTP Verbs Express Actions
✅ POST /teams/123/projects (create)
❌ POST /teams/123/createProject

3. Nested Resources Show Relationships
✅ /projects/456/tasks/789
❌ /tasks/789?project=456

4. Query Parameters for Filtering
✅ /tasks?status=active&assignee=john
❌ /getActiveTasks/john

Slide 10: Domain-to-REST Mapping Challenge

🎯 Design Challenge - 15 Minutes

Given TaskFlow Domains:
- Teams (with members)
- Projects (belong to teams)
- Tasks (belong to projects)

Your Mission: Design REST endpoints

Consider:
- How do you get all projects for a team?
- How do you assign a task to someone?
- How do you handle bulk operations?
- What about cross-domain operations?

Constraint: Use business language in URLs

Slide 11: HTTP Status Codes Reality

🎯 HTTP Status Codes in Production

The Basics Everyone Knows:
- 200 OK
- 404 Not Found
- 500 Internal Server Error

The Ones That Matter:
- 400 Bad Request (client error)
- 401 Unauthorized (missing auth)
- 403 Forbidden (insufficient permissions)
- 409 Conflict (business rule violation)
- 422 Unprocessable Entity (validation failed)
- 429 Too Many Requests (rate limited)

The Real Question: Which one do you use when...?

Slide 12: Error Response Design Pattern

🎯 Professional Error Responses

❌ Technical Error (What developers write):

```
{
  "error": "ValidationError",
  "message": "Foreign key constraint failed",
  "code": 500
}
```

✅ Business Error (What users need):

```
{
  "error": {
    "type": "invalid_request",
    "message": "Cannot assign task to user outside the team",
    "details": {
      "task_id": "task_123",
      "user_id": "user_456",
      "team_id": "team_789"
    },
    "suggestion": "Add user to team first, then assign task"
  }
}
```

Slide 13: gRPC vs REST Comparison

🎯 gRPC vs REST: When to Use Each

Performance Comparison:

Scenario	REST	gRPC
Single Request	~100ms	~50ms
Streaming Data	Polling	Native
Binary Efficiency	JSON	Protobuf
Browser Support	Native	Proxy
Human Readable	Yes	No

Decision Framework:
- Public API → REST
- Internal Services → Consider gRPC
- Real-time Features → gRPC Streaming
- Simple CRUD → REST
- High Performance → gRPC

Slide 14: gRPC Demo Preview

🎯 gRPC Service Example

TaskFlow Analytics Service:

```
service TaskAnalytics {
  // Single request/response
  rpc GetTeamPerformance(TeamRequest) returns (TeamPerformance);

  // Server streaming - real-time updates
  rpc StreamTaskUpdates(TaskStreamRequest) returns (stream TaskUpdate);

  // Bidirectional streaming - collaboration
  rpc CollaborateOnTask(stream TaskCollaboration) returns (stream TaskCollaboration);
}
```

Key Insight: gRPC excels at internal service communication

Slide 15: Legacy Integration Reality

🎯 The Integration Challenge

Reality Check: 80% of API work is integrating with existing systems

Common Problems:
- XML responses in a JSON world
- Inconsistent field naming
- Poor error handling
- No documentation
- Unreliable uptime

Your Job: Make ugly APIs beautiful for your consumers

Strategy: Design clean facades that hide the complexity

Slide 16: Legacy API Challenge Setup

🎯 Legacy Integration Challenge - 15 Minutes

The Scenario:
You must integrate with a legacy weather service and user management system

Your Mission:
Design modern, clean API responses that hide the legacy ugliness

Requirements:
- Consistent response format
- Modern field naming (camelCase)
- Proper HTTP status codes
- Graceful error handling

Question: How do you make bad APIs good?

Slide 17: Day 1 Wrap-up

🎯 Day 1 Achievements

What You Built Today:
✅ Complete TaskFlow domain model
✅ REST endpoint design with business language
✅ Professional error handling patterns
✅ gRPC vs REST decision framework
✅ Legacy integration strategies

Tomorrow's Focus:
🔨 Turn your blueprint into working FastAPI code
🔨 Domain-driven implementation patterns
🔨 Production-grade testing strategies

Take-Home: Set up FastAPI project with domain structure

Slide 18: Questions & Discussion

🎯 Strategic Questions for Tonight

Domain Modeling:
- How will DDD influence your current API projects?
- What domain boundaries do you see in your work systems?

API Design:
- Which status code scenarios surprised you?
- Where would you use gRPC in your architecture?

Integration:
- What's the worst legacy system you've worked with?
- How do you handle unreliable external APIs?

Tomorrow: We turn these designs into production code

📋 Slide Usage Notes

Slide Timing Guide:

- **1-2 slides per major section** - not slide-heavy
- **Activity instruction slides** stay up during hands-on work
- **Reference slides** (like status codes) stay visible during discussions
- **Remove slides** when doing pure hands-on work

Presentation Tips:

- **Don't read slides** - use them as visual anchors
- **Point to slides** during activities for reference
- **Skip slides if running behind** - prioritize hands-on time
- **Return to reference slides** during Q&A

4-Student Modifications:

- **More discussion time** per slide since group is small
- **Ask direct questions** to each student
- **Use slides as debate starters** - "Who disagrees with this?"
- **Pause longer** on decision matrix slides for deep discussion

Backup Slides (if time allows):

- Additional status code scenarios
- More gRPC examples
- Advanced domain modeling patterns
- Real-world API case studies

This gives you the visual structure you need while keeping the focus on hands-on activities. The slides support the activities rather than replace them!