AI-Based Automated STEM Evaluation System - Technical Implementation Plan

1. Implementation Overview

1.1 Development Approach

- **Methodology**: Agile with 2-week sprints
- **Team Size**: 6-8 engineers (2 frontend, 3 backend, 1 DevOps, 1 QA, 1 AI/ML specialist)
- **Timeline**: 6 months to MVP, 9 months to full v1.0
- **Release Strategy**: Phased rollout with pilot program

1.2 Implementation Phases

```
Phase 0: Foundation (Weeks 1-4)
    Infrastructure setup, repo structure, CI/CD

Phase 1: Core Platform (Weeks 5-12)
    Authentication & user management
    Basic frontend scaffolding
    Database foundation

Phase 2: Assessment System (Weeks 13-18)
    Question management
    Assessment delivery
    Basic AI integration

Phase 3: Evaluation Engine (Weeks 19-24)
    File upload and processing
    AI evaluation pipeline
    Feedback generation

Phase 4: Reporting & Analytics (Weeks 25-28)
    Individual reports
    Benchmarking system
    Coach analytics

Phase 5: Forum & Collaboration (Weeks 29-32)
    Discussion board
    Q&A functionality
    Notifications

Phase 6: Polish & Launch (Weeks 33-40)
    Performance optimization
    Security hardening
```

```
User testing & refinement
Production launch
```

2. Phase 0: Foundation (Weeks 1-4)

2.1 Week 1-2: Project Setup

Repository Structure

```
/monorepo
  /frontend          # Next.js application
  /services          # Go microservices
     /api-gateway
     /auth-service
     /user-service
     /assessment-service
     /evaluation-service
     /report-service
     /forum-service
     /notification-service
  /shared            # Shared libraries
     /proto         # Protocol buffers (if using gRPC)
     /types         # Shared type definitions
  /infrastructure    # Terraform configurations
  /docs              # Documentation
  /scripts           # Utility scripts
```

Development Environment Setup

- ☐ Create GitHub organization and repositories
- ☐ Set up branch protection rules (main, develop)
- ☐ Configure pre-commit hooks (linting, formatting)
- ☐ Set up development Docker Compose configuration
- ☐ Create `.env.example` files for each service
- ☐ Document local development setup in README

CI/CD Pipeline

- ☐ Configure GitHub Actions workflows
  - ☐ Lint workflow (ESLint, golangci-lint)
  - ☐ Test workflow (Jest, Go test)
  - ☐ Build workflow (Docker images)
  - ☐ Security scanning (Trivy, Snyk)
- ☐ Set up Artifact Registry in GCP
- ☐ Configure automated deployments to staging
- ☐ Set up deployment approval workflow for production

2.2 Week 2-3: Infrastructure as Code

GCP Resource Provisioning

```
# Terraform modules to create:
- VPC and networking
- Cloud SQL (PostgreSQL)
- Redis (Memorystore)
- Cloud Storage buckets
- Cloud Run services (placeholder)
- Load balancers
- IAM roles and service accounts
- Secret Manager for sensitive data
```

Tasks: - [ ] Initialize Terraform project structure - [ ] Create GCP project and enable APIs - [ ] Set up Terraform backend (GCS) - [ ] Write Terraform modules for each resource type - [ ] Create separate environments (dev, staging, prod) - [ ] Document infrastructure deployment process - [ ] Test infrastructure creation and destruction

Database Setup

- ☐ Design initial database schema (see Database Schema document)
- ☐ Create migration tool setup (golang-migrate)
- ☐ Write initial migration scripts
- ☐ Set up database seeding for development
- ☐ Configure connection pooling (PgBouncer)
- ☐ Set up read replica configuration (manual activation later)

2.3 Week 3-4: Development Tooling

Frontend Tooling

- ☐ Initialize Next.js project with TypeScript
- ☐ Configure Tailwind CSS and shadcn/ui
- ☐ Set up ESLint and Prettier
- ☐ Configure path aliases (@/ imports)
- ☐ Set up React Query and Zustand
- ☐ Create base UI component library
- ☐ Set up Storybook for component development
- ☐ Configure testing (Jest, React Testing Library)

Backend Tooling

- ☐ Initialize Go modules for each service
- ☐ Set up shared libraries and utilities
- ☐ Configure golangci-lint
- ☐ Set up API documentation (Swagger/Swag)
- ☐ Create common middleware (logging, auth, CORS)

☐ Set up testing framework and mocks
☐ Configure database connection management

AI Integration Setup

☐ Create GCP service account for Vertex AI
☐ Set up Vertex AI SDK in Go
☐ Create prompt template management system
☐ Design AI abstraction layer interfaces
☐ Set up prompt caching configuration
☐ Create cost tracking utilities
☐ Write initial prompt templates

3. Phase 1: Core Platform (Weeks 5-12)

3.1 Week 5-7: Authentication System

Authentication Service Implementation **Priority**: Critical

Tasks: - [ ] Design user and session database tables - [ ] Implement user registration endpoint - [ ] Email validation - [ ] Password hashing (bcrypt) - [ ] Email verification flow - [ ] Implement login endpoint - [ ] Credential validation - [ ] JWT generation (access + refresh tokens) - [ ] Session storage in Redis - [ ] Implement token refresh endpoint - [ ] Implement password reset flow - [ ] Reset token generation - [ ] Email sending integration - [ ] Password update endpoint - [ ] Implement Google OAuth integration - [ ] OAuth callback handler - [ ] Account linking logic - [ ] Write comprehensive tests (unit + integration)

API Endpoints:

```
POST   /api/v1/auth/register
POST   /api/v1/auth/login
POST   /api/v1/auth/logout
POST   /api/v1/auth/refresh
POST   /api/v1/auth/forgot-password
POST   /api/v1/auth/reset-password
GET    /api/v1/auth/google
GET    /api/v1/auth/google/callback
```

Database Tables: `- users - sessions - password_reset_tokens - oauth_connections`

API Gateway Implementation **Priority**: Critical

Tasks: - [ ] Set up Gin/Echo HTTP server - [ ] Implement JWT validation middleware - [ ] Implement rate limiting middleware - [ ] Per-user limits - [ ] Per-IP limits - [ ] Configure Redis for rate limit storage - [ ] Implement request logging middleware - [ ] Implement CORS middleware - [ ] Set up request routing

to services - [ ] Implement health check endpoints - [ ] Add request tracing (trace ID generation)

Middleware Chain:

```
Request → CORS → Rate Limit → Logging → Auth → Service Router → Response
```

Frontend Authentication **Priority**: Critical

Tasks: - [ ] Create authentication context/store - [ ] Build login page UI - [ ] Build registration page UI - [ ] Build password reset flow UI - [ ] Implement token management - [ ] Token storage (httpOnly cookies) - [ ] Automatic token refresh - [ ] Token expiry handling - [ ] Create protected route wrapper - [ ] Implement Google OAuth button - [ ] Handle OAuth callback

Pages: - `/login` - `/register` - `/forgot-password` - `/reset-password/:token` - `/auth/callback`

3.2 Week 7-9: User Management System

User Service Implementation **Priority**: Critical

Tasks: - [ ] Implement CRUD endpoints for users - [ ] Get user profile - [ ] Update user profile - [ ] Upload profile picture - [ ] Implement role management - [ ] Assign roles to users - [ ] Update user roles (admin only) - [ ] Implement organization management - [ ] Create organization - [ ] Manage organization settings - [ ] Implement class management - [ ] Create class - [ ] Add/remove students from class - [ ] Assign coaches to class - [ ] Implement team management - [ ] Create VEX team - [ ] Add/remove team members - [ ] Implement parent-student linking - [ ] Send link request - [ ] Approve/deny link request - [ ] Unlink accounts - [ ] Write comprehensive tests

API Endpoints:

```
# User management
GET    /api/v1/users/me
PUT    /api/v1/users/me
POST   /api/v1/users/me/avatar
GET    /api/v1/users/:id
PUT    /api/v1/users/:id (admin only)

# Organization management
POST   /api/v1/organizations
GET    /api/v1/organizations/:id
PUT    /api/v1/organizations/:id

# Class management
POST   /api/v1/classes
GET    /api/v1/classes/:id
```

```
PUT    /api/v1/classes/:id
POST   /api/v1/classes/:id/students
DELETE /api/v1/classes/:id/students/:studentId
GET    /api/v1/classes/:id/invite-code

# Team management
POST   /api/v1/teams
GET    /api/v1/teams/:id
PUT    /api/v1/teams/:id
POST   /api/v1/teams/:id/members

# Parent-student linking
POST   /api/v1/links/request
GET    /api/v1/links/pending
POST   /api/v1/links/:id/approve
DELETE /api/v1/links/:id
```

Database Tables: - `users` (extended) - `organizations` - `classes` - `class_students` - `class_coaches` - `teams` - `team_members` - `parent_student_links`

Frontend User Management **Priority**: Critical

Tasks: - [ ] Build user profile page - [ ] Build profile edit form - [ ] Build organization dashboard (admin) - [ ] Build class management interface (coach) - [ ] Build team management interface - [ ] Build parent linking interface - [ ] Implement RBAC UI components - [ ] Role-based navigation - [ ] Conditional rendering based on permissions

Pages: - `/dashboard/profile` - `/dashboard/admin/organization` - `/dashboard/coach/classes` - `/dashboard/coach/classes/:id` - `/dashboard/parent/children` - `/dashboard/student/team`

3.3 Week 9-12: Basic Frontend Structure

Layout and Navigation **Priority**: High

Tasks: - [ ] Create main layout component - [ ] Build navigation sidebar - [ ] Role-based menu items - [ ] Active state highlighting - [ ] Collapsible on mobile - [ ] Build top navigation bar - [ ] User menu dropdown - [ ] Notification bell (placeholder) - [ ] Organization/class selector - [ ] Create loading states and skeletons - [ ] Implement error boundary components - [ ] Create 404 and error pages

Dashboard Views **Priority**: High

Tasks: - [ ] Build student dashboard home - [ ] Recent submissions - [ ] Upcoming assessments - [ ] Progress overview - [ ] Build coach dashboard home - [ ] Class summary cards - [ ] Recent student activity - [ ] Pending evaluations - [ ] Build

parent dashboard home - [ ] Children overview - [ ] Recent notifications - [ ] Build admin dashboard home - [ ] System statistics - [ ] User activity metrics

4. Phase 2: Assessment System (Weeks 13-18)

4.1 Week 13-15: Question Management

Assessment Service Implementation **Priority**: Critical

Tasks: - [ ] Design question schema with flexibility for types - [ ] Implement question CRUD endpoints (admin/coach) - [ ] Implement question pool management - [ ] Implement question tagging and categorization - [ ] Build question difficulty rating system - [ ] Implement question versioning - [ ] Support multiple question types: - [ ] Multiple choice - [ ] Short answer - [ ] Code submission - [ ] Image upload - [ ] Write question bank seeding script - [ ] Write comprehensive tests

API Endpoints:

```
POST   /api/v1/questions
GET    /api/v1/questions/:id
PUT    /api/v1/questions/:id
DELETE /api/v1/questions/:id
GET    /api/v1/questions (with filters)
POST   /api/v1/question-pools
GET    /api/v1/question-pools/:id
```

Database Tables: - `questions` - `question_options` (for multiple choice) - `question_tags` - `question_pools` - `question_pool_items`

Question Management UI **Priority**: Medium (coaches can use API directly initially)

Tasks: - [ ] Build question library page (coach/admin) - [ ] Build question creation form - [ ] Dynamic form based on question type - [ ] Rich text editor for question content - [ ] Option management for multiple choice - [ ] Build question edit interface - [ ] Build question preview component - [ ] Implement question search and filtering

4.2 Week 15-17: Assessment Delivery

Assessment Session Management **Priority**: Critical

Tasks: - [ ] Implement assessment creation endpoint - [ ] Implement assessment assignment to students/classes - [ ] Build adaptive question selection algorithm - [ ] Start with medium difficulty - [ ] Adjust based on correctness - [ ] Avoid question repetition - [ ] Implement assessment session management - [ ] Start assessment - [ ] Submit answers - [ ] Track progress - [ ] Time limits (optional) - [ ] Implement assessment scoring - [ ] Immediate scoring for objective questions

- [ ] Queue subjective questions for AI evaluation - [ ] Store assessment results - [ ] Write comprehensive tests

API Endpoints:

```
POST   /api/v1/assessments
GET    /api/v1/assessments/:id
PUT    /api/v1/assessments/:id
POST   /api/v1/assessments/:id/assign
GET    /api/v1/assessments/assigned (student view)

POST   /api/v1/assessment-sessions
GET    /api/v1/assessment-sessions/:id
POST   /api/v1/assessment-sessions/:id/submit-answer
POST   /api/v1/assessment-sessions/:id/complete
GET    /api/v1/assessment-sessions/:id/results
```

Database Tables: - `assessments` - `assessment_assignments` - `assessment_sessions` - `assessment_responses` - `assessment_results`

Assessment UI **Priority**: Critical

Tasks: - [ ] Build assessment list page (student) - [ ] Build assessment taking interface - [ ] Question navigation - [ ] Progress indicator - [ ] Answer submission - [ ] Timer (if applicable) - [ ] Build assessment results page - [ ] Score display - [ ] Question review - [ ] Correct answers (if enabled) - [ ] Build assessment management (coach) - [ ] Create assessment - [ ] Assign to students/classes - [ ] View submissions

4.3 Week 17-18: Basic AI Integration

AI Evaluation for Short Answer Questions **Priority**: High

Tasks: - [ ] Create prompt template for short answer evaluation - [ ] Implement AI evaluation worker - [ ] Fetch pending evaluations - [ ] Call Vertex AI API - [ ] Parse and store results - [ ] Implement retry logic for API failures - [ ] Implement cost tracking per evaluation - [ ] Set up prompt caching for common rubrics - [ ] Write evaluation quality tests (against human labels)

Prompt Template Example:

```
You are an expert STEM educator evaluating student responses.

Question: {question_text}
Expected Answer: {expected_answer}
Student Answer: {student_answer}

Evaluate the student's answer on a scale of 0-100 and provide brief feedback.
Return your response in JSON format:
```

```
{
  "score": <0-100>,
  "feedback": "<constructive feedback>",
  "strengths": ["<strength1>", "<strength2>"],
  "improvements": ["<area1>", "<area2>"]
}
```

Background Worker: - [ ] Set up worker process or Cloud Run job - [ ] Poll for pending evaluations - [ ] Process in batches - [ ] Update database with results - [ ] Send notifications on completion

5. Phase 3: Evaluation Engine (Weeks 19-24)

5.1 Week 19-20: File Upload System

Evaluation Service - Upload Module **Priority**: Critical

Tasks: - [ ] Implement file upload endpoint - [ ] Multipart form handling - [ ] File type validation - [ ] File size limits - [ ] Virus scanning integration (ClamAV or Cloud Security Scanner) - [ ] Implement GCS integration - [ ] Upload files to appropriate bucket - [ ] Generate signed URLs for access - [ ] Set proper access controls - [ ] Implement thumbnail generation for images - [ ] Implement file metadata storage - [ ] Support batch uploads (up to 20 files) - [ ] Write comprehensive tests

API Endpoints:

```
POST   /api/v1/submissions
GET    /api/v1/submissions/:id
GET    /api/v1/submissions/:id/files/:fileId
DELETE /api/v1/submissions/:id (soft delete)
```

Database Tables: - `submissions` - `submission_files` - `file_metadata`

Frontend Upload Interface **Priority**: Critical

Tasks: - [ ] Build submission creation page - [ ] Implement drag-and-drop file upload - [ ] Preview uploaded files - [ ] Remove files before submission - [ ] Progress indicators - [ ] Implement file type restrictions - [ ] Show upload progress for large files - [ ] Build submission detail view - [ ] Display all submitted files - [ ] Show evaluation status - [ ] Display feedback when ready

5.2 Week 20-22: Image Analysis (VEX Robotics)

AI Image Evaluation **Priority**: Critical

Tasks: - [ ] Create prompt templates for robot image analysis - [ ] Component identification - [ ] Design quality assessment - [ ] VEX-specific criteria (gear ratios, stability, etc.) - [ ] Implement Vertex AI Vision integration - [ ] Process robot images: - [ ] Detect VEX components (motors, sensors, structural pieces)

- [ ] Evaluate mechanical design - [ ] Identify potential issues - [ ] Generate improvement suggestions - [ ] Implement batch image processing - [ ] Handle multiple images per submission - [ ] Store structured evaluation results - [ ] Write evaluation quality tests

Prompt Template Example:

```
You are a VEX robotics expert evaluating a robot design from an image.

Competition: {competition_type} (VEX IQ or VEX V5)
Image Description: Analyze the robot shown in this image.

Evaluate the following aspects:
1. Design Quality (0-100)
2. Component Usage
3. Structural Integrity
4. Potential Issues
5. Improvement Suggestions

Return your response in JSON format:
{
  "overall_score": <0-100>,
  "components_identified": ["<component1>", "<component2>"],
  "design_strengths": ["<strength1>", "<strength2>"],
  "potential_issues": ["<issue1>", "<issue2>"],
  "improvement_suggestions": ["<suggestion1>", "<suggestion2>"],
  "detailed_feedback": "<comprehensive feedback>"
}
```

5.3 Week 22-23: Code Evaluation

Code Analysis Module **Priority**: Critical

Tasks: - [ ] Implement static code analysis - [ ] Python: pylint, mypy - [ ] C++: clang-tidy - [ ] VEXcode Blocks: custom parser - [ ] Integrate AI for code review - [ ] Code quality assessment - [ ] Best practices validation - [ ] Algorithm efficiency analysis - [ ] Implement optimization suggestions - [ ] Performance improvements - [ ] Code simplification - [ ] Better algorithmic approaches - [ ] Create VEX-specific code patterns - [ ] Sensor usage validation - [ ] Motor control best practices - [ ] Competition-ready code checks - [ ] Store code metrics and feedback

Code Evaluation Pipeline:

```
1. Parse code → 2. Static analysis → 3. AI analysis → 4. Generate feedback
```

Prompt Template Example:

```
You are a programming mentor evaluating VEX robotics code.
```

```
Language: {language}
Competition: {competition_type}
Code:
```{language}
{code_content}
```

Evaluate the code for: 1. Correctness and logic 2. Code quality and style 3. VEX best practices 4. Performance optimization opportunities 5. Suggest specific improvements with examples

Return your response in JSON format: { "overall_score": <0-100>, "correctness_score": <0-100>, "quality_score": <0-100>, "performance_score": <0-100>, "positive_aspects": ["", ""], "issues": [{ "line": , "severity": "<info|warning|error>", "message": "", "suggestion": "" }], "optimizations": [{ "description": "", "current_code": "", "improved_code": "", "reasoning": "" }] }
```

```
### 5.4 Week 23-24: Engineering Notebook Analysis

#### Document Processing
**Priority**: High

**Tasks**:
- [ ] Implement PDF text extraction
  - [ ] Use pdf.js or pdfplumber
  - [ ] Preserve structure and formatting
- [ ] Implement DOCX parsing
- [ ] Extract images from documents
- [ ] Implement AI-based notebook evaluation
  - [ ] Check for required sections
  - [ ] Evaluate documentation quality
  - [ ] Assess design process documentation
  - [ ] Check for reflection and iteration
- [ ] Generate rubric-based scores
- [ ] Provide section-by-section feedback
- [ ] Store structured evaluation results

**VEX Engineering Notebook Rubric**:
- Team identification and game description
- Design process and iterations
- Meeting notes and progress logs
- Testing and troubleshooting
- Programming documentation
- Reflection and learning
```

Prompt Template Example:

You are a VEX judge evaluating an engineering notebook.

Extracted Notebook Content: {notebook_text}

Evaluate the notebook using the official VEX rubric: 1. Completeness: Are all required sections present? 2. Design Process: Is the design process clearly documented? 3. Iteration: Are design iterations shown with reasoning? 4. Testing: Are tests documented with results? 5. Reflection: Is there evidence of learning and reflection?

Return your response in JSON format: { “overall_score”: <0-100>, “rubric_scores”: { “completeness”: <0-25>, “design_process”: <0-25>, “iteration”: <0-25>, “testing_reflection”: <0-25> }, “missing_sections”: [“”, “”], “strengths”: [“”, “”], “improvement_areas”: [“”, “”], “detailed_feedback”: “” }

6. Phase 4: Reporting & Analytics (Weeks 25-28)

6.1 Week 25-26: Individual Reports

Report Service Implementation
Priority: High

Tasks:
- [ ] Implement report generation endpoints
- [ ] Calculate student STEM capability scores
  - [ ] Programming proficiency
  - [ ] Engineering design skills
  - [ ] Problem-solving ability
  - [ ] Documentation quality
- [ ] Implement progress tracking over time
- [ ] Generate trend charts data
- [ ] Implement PDF report generation
  - [ ] Use library like wkhtmltopdf or Puppeteer
  - [ ] Design professional report templates
- [ ] Cache generated reports
- [ ] Write comprehensive tests

API Endpoints:

GET /api/v1/reports/student/:studentId GET /api/v1/reports/student/:studentId/progress GET /api/v1/reports/student/:studentId/pdf

Report Sections:
- Executive summary

```
- STEM capability scores with radar chart
- Progress over time (line charts)
- Recent submissions and feedback
- Strengths and growth areas
- Recommended next steps

#### Report UI
**Priority**: High

**Tasks**:
- [ ] Build student report page
- [ ] Implement interactive charts (Recharts)
- [ ] Build PDF export functionality
- [ ] Build print-friendly view
- [ ] Create parent-friendly report version

### 6.2 Week 26-27: Benchmarking System

#### Benchmarking Implementation
**Priority**: High

**Tasks**:
- [ ] Define cohort groupings (class, grade, organization)
- [ ] Implement percentile calculations
- [ ] Calculate standard deviation from mean
- [ ] Define performance levels (Beginner, Intermediate, Advanced, Expert)
- [ ] Ensure anonymization of comparison data
- [ ] Implement FERPA compliance checks
- [ ] Create benchmark visualization data
- [ ] Write comprehensive tests

**API Endpoints**:
```

GET /api/v1/benchmarks/student/:studentId GET /api/v1/benchmarks/class/:classId

```
**Benchmark Metrics**:
- Overall STEM score percentile
- Programming score percentile
- Engineering score percentile
- Submission frequency percentile
- Performance level designation

#### Benchmarking UI
**Priority**: Medium

**Tasks**:
```

```
- [ ] Build benchmark comparison page
- [ ] Create anonymized comparison charts
- [ ] Show percentile indicators
- [ ] Display performance level badges
- [ ] Add explanatory tooltips

### 6.3 Week 27-28: Coach Analytics Dashboard

#### Coach Analytics Implementation
**Priority**: High

**Tasks**:
- [ ] Implement class-level aggregation queries
- [ ] Calculate class performance metrics
- [ ] Identify students needing intervention
  - [ ] Low submission rate
  - [ ] Declining performance
  - [ ] Incomplete assessments
- [ ] Implement engagement metrics
- [ ] Build export functionality (CSV, PDF)
- [ ] Write comprehensive tests

**API Endpoints**:
```

GET /api/v1/analytics/class/:classId GET /api/v1/analytics/class/:classId/students GET /api/v1/analytics/class/:classId/export

```
**Analytics Metrics**:
- Average class STEM score
- Submission rate
- Assessment completion rate
- Forum participation
- Performance distribution
- At-risk student list

#### Coach Analytics UI
**Priority**: High

**Tasks**:
- [ ] Build coach analytics dashboard
- [ ] Create class overview cards
- [ ] Build student performance table
  - [ ] Sortable columns
  - [ ] Filterable data
  - [ ] Highlight at-risk students
- [ ] Add export buttons
```

```
- [ ] Create drill-down views for individual students

## 7. Phase 5: Forum & Collaboration (Weeks 29-32)

### 7.1 Week 29-30: Discussion Board

#### Forum Service Implementation
**Priority**: Medium

**Tasks**:
- [ ] Implement post CRUD endpoints
- [ ] Implement comment system with threading
- [ ] Implement voting system (upvote/downvote)
- [ ] Implement post tagging
- [ ] Implement search functionality
  - [ ] Full-text search on posts and comments
  - [ ] Filter by tags
  - [ ] Filter by answered status
- [ ] Implement sorting algorithms
  - [ ] Hot (trending)
  - [ ] New (chronological)
  - [ ] Top (most upvotes)
- [ ] Implement content moderation queue
- [ ] Write comprehensive tests

**API Endpoints**:
```

POST /api/v1/forum/posts GET /api/v1/forum/posts GET /api/v1/forum/posts/:id
PUT /api/v1/forum/posts/:id DELETE /api/v1/forum/posts/:id

POST /api/v1/forum/posts/:postId/comments GET /api/v1/forum/posts/:postId/comments
PUT /api/v1/forum/comments/:id DELETE /api/v1/forum/comments/:id

POST /api/v1/forum/posts/:postId/vote POST /api/v1/forum/comments/:id/vote

GET /api/v1/forum/search

```
**Database Tables**:
- `forum_posts`
- `forum_comments`
- `forum_votes`
- `forum_tags`
- `forum_post_tags`
- `moderation_queue`

#### Forum UI
**Priority**: Medium
```

```
**Tasks**:
- [ ] Build forum home page
  - [ ] Post list with sorting
  - [ ] Tag filter sidebar
  - [ ] Search bar
- [ ] Build post creation form
  - [ ] Rich text editor
  - [ ] Media upload support
  - [ ] Tag selection
- [ ] Build post detail page
  - [ ] Post content display
  - [ ] Comment threading (up to 5 levels)
  - [ ] Vote buttons
  - [ ] Share functionality
- [ ] Build search results page

### 7.2 Week 30-31: Q&A Functionality

#### Q&A Features
**Priority**: Medium

**Tasks**:
- [ ] Implement question marking logic
- [ ] Implement "best answer" designation
- [ ] Create coach notification system for unanswered questions
- [ ] Implement answer acceptance workflow
- [ ] Add reputation/points system (optional)
- [ ] Write comprehensive tests

**Additional API Endpoints**:
```

POST /api/v1/forum/posts/:postId/mark-answered POST /api/v1/forum/comments/:id/accept-answer GET /api/v1/forum/unanswered

```
#### Q&A UI
**Priority**: Medium

**Tasks**:
- [ ] Add "Mark as Question" toggle to post creation
- [ ] Show question status indicator
- [ ] Build "Accept Answer" button (for post author)
- [ ] Highlight accepted answer
- [ ] Create unanswered questions view (coach)

### 7.3 Week 31-32: Notifications
```

```
#### Notification Service Implementation
**Priority**: High

**Tasks**:
- [ ] Implement notification creation endpoints
- [ ] Implement notification delivery
  - [ ] In-app notifications
  - [ ] Email notifications
- [ ] Implement notification preferences
- [ ] Implement digest scheduling (daily/weekly)
- [ ] Integrate with SendGrid for emails
- [ ] Implement WebSocket server for real-time push
- [ ] Write email templates
- [ ] Write comprehensive tests

**API Endpoints**:
```

GET /api/v1/notifications GET /api/v1/notifications/unread-count POST /api/v1/notifications/:id/read POST /api/v1/notifications/mark-all-read DELETE /api/v1/notifications/:id

GET /api/v1/notifications/preferences PUT /api/v1/notifications/preferences

```
**Notification Types**:
- Submission feedback ready
- Assessment assigned
- Forum reply
- Forum mention
- Best answer selected
- Parent link request
- Class invitation

**Database Tables**:
- `notifications`
- `notification_preferences`
- `email_queue`

#### Notification UI
**Priority**: High

**Tasks**:
- [ ] Build notification dropdown in navbar
  - [ ] Show unread count badge
  - [ ] List recent notifications
  - [ ] Mark as read on click
```

```
- [ ] Build notification center page
  - [ ] All notifications list
  - [ ] Filter by type
  - [ ] Mark all as read button
- [ ] Build notification preferences page
- [ ] Implement WebSocket connection for real-time updates

## 8. Phase 6: Polish & Launch (Weeks 33-40)

### 8.1 Week 33-34: Performance Optimization

**Tasks**:
- [ ] Frontend performance audit
  - [ ] Lighthouse CI integration
  - [ ] Code splitting optimization
  - [ ] Image optimization (next/image)
  - [ ] Reduce bundle size
  - [ ] Lazy loading implementation
- [ ] Backend performance optimization
  - [ ] Query optimization (EXPLAIN ANALYZE)
  - [ ] Add database indexes for slow queries
  - [ ] Implement query result caching
  - [ ] Optimize AI API calls (batching, caching)
- [ ] Load testing
  - [ ] Use k6 or Apache JMeter
  - [ ] Test with 1000 concurrent users
  - [ ] Identify bottlenecks
  - [ ] Optimize slow endpoints
- [ ] Set up APM (Application Performance Monitoring)
  - [ ] Integrate New Relic or Datadog
  - [ ] Set up alerts for slow queries
  - [ ] Monitor AI API costs

### 8.2 Week 34-36: Security Hardening

**Tasks**:
- [ ] Security audit
  - [ ] OWASP Top 10 checklist
  - [ ] SQL injection testing
  - [ ] XSS testing
  - [ ] CSRF protection verification
- [ ] Penetration testing (external service)
- [ ] Implement security headers
  - [ ] Content-Security-Policy
  - [ ] X-Content-Type-Options
  - [ ] X-Frame-Options
```

```
  - [ ] Strict-Transport-Security
- [ ] Set up Web Application Firewall (Cloud Armor)
- [ ] Implement rate limiting enhancements
- [ ] Review and rotate secrets
- [ ] Set up security monitoring
  - [ ] Intrusion detection
  - [ ] Anomaly detection
  - [ ] Failed login attempt monitoring
- [ ] Complete FERPA compliance checklist
- [ ] Document security measures

### 8.3 Week 36-38: User Testing & Refinement

**Tasks**:
- [ ] Recruit pilot users
  - [ ] 2-3 coaches
  - [ ] 1 class of students (~20 students)
  - [ ] 5-10 parents
  - [ ] 1 admin
- [ ] Conduct usability testing sessions
  - [ ] Task-based testing
  - [ ] Think-aloud protocol
  - [ ] Record sessions
- [ ] Gather feedback
  - [ ] User surveys (SUS, NPS)
  - [ ] Feature requests
  - [ ] Bug reports
- [ ] Prioritize and fix issues
- [ ] Iterate on UI/UX based on feedback
- [ ] Conduct follow-up testing
- [ ] Document user feedback and changes

### 8.4 Week 38-39: Documentation & Training

**Tasks**:
- [ ] Write user documentation
  - [ ] Getting started guide
  - [ ] Student user guide
  - [ ] Coach user guide
  - [ ] Parent user guide
  - [ ] Admin user guide
  - [ ] FAQ
- [ ] Create video tutorials
  - [ ] Platform overview
  - [ ] How to submit work
  - [ ] How to create assessments
```

```
  - [ ] How to read reports
- [ ] Write technical documentation
  - [ ] API documentation (complete OpenAPI spec)
  - [ ] Architecture overview
  - [ ] Deployment guide
  - [ ] Troubleshooting guide
- [ ] Create onboarding materials
- [ ] Prepare training sessions for pilot coaches

### 8.5 Week 39-40: Production Launch

**Pre-Launch Checklist**:
- [ ] All tests passing (unit, integration, e2e)
- [ ] Security audit complete
- [ ] Performance benchmarks met
- [ ] Documentation complete
- [ ] Monitoring and alerting configured
- [ ] Backup and recovery tested
- [ ] Disaster recovery plan documented
- [ ] Support process defined
- [ ] Terms of Service and Privacy Policy published
- [ ] FERPA compliance verified

**Launch Steps**:
- [ ] Deploy to production environment
- [ ] Smoke test in production
- [ ] Enable monitoring and alerts
- [ ] Communicate launch to pilot users
- [ ] Monitor closely for first 48 hours
- [ ] Address any critical issues immediately
- [ ] Collect initial feedback
- [ ] Celebrate launch!

**Post-Launch (Week 40+)**:
- [ ] Daily monitoring of system health
- [ ] Weekly feedback review meetings
- [ ] Monthly feature planning
- [ ] Continuous improvement based on user feedback
- [ ] Plan Phase 2 features

## 9. Development Best Practices

### 9.1 Code Quality Standards

**Frontend**:
- TypeScript strict mode enabled
```

```
- ESLint and Prettier configured
- PropTypes or TypeScript interfaces for all components
- Minimum 70% test coverage (unit + integration)
- Accessibility testing (axe-core)

**Backend**:
- golangci-lint with strict rules
- 80% test coverage (unit + integration)
- Code review required for all PRs
- No direct database queries in handlers (use repository pattern)
- Error handling and logging in all functions

### 9.2 Git Workflow

**Branch Strategy**: Git Flow
- `main`: Production-ready code
- `develop`: Integration branch for features
- `feature/*`: Individual feature branches
- `hotfix/*`: Emergency production fixes

**Commit Messages**: Conventional Commits
```

feat: add submission file upload fix: resolve authentication bug docs: update API documentation test: add unit tests for user service refactor: improve code readability in assessment service “‘

Pull Request Process: 1. Create feature branch from `develop` 2. Implement feature with tests 3. Open PR with description and screenshots 4. Code review by at least one team member 5. Automated tests must pass 6. Merge to `develop` after approval

9.3 Testing Strategy

Unit Tests: - Test individual functions and methods - Mock external dependencies - Aim for 80% coverage

Integration Tests: - Test API endpoints with real database (test DB) - Test service interactions - Test authentication flows

End-to-End Tests: - Use Playwright or Cypress - Test critical user journeys - Student submission flow - Coach evaluation review flow - Assessment taking flow - Run in CI before deployment

AI Evaluation Tests: - Maintain labeled dataset of sample submissions - Regularly evaluate AI output against human expert labels - Track precision, recall, F1 score - Alert if quality metrics decline

9.4 Deployment Process

Continuous Integration: - Run on every commit to any branch - Lint, test, build - Report results on PR

Continuous Deployment: - Auto-deploy to staging on merge to `develop` - Manual approval required for production - Blue-green deployment strategy - Automatic rollback on errors

Deployment Checklist: - [ ] All tests passing - [ ] Database migrations ready - [ ] Environment variables configured - [ ] Monitoring enabled - [ ] Rollback plan prepared

10. Team Roles and Responsibilities

10.1 Frontend Engineers (2)

- Implement Next.js application
- Build reusable UI components
- Integrate with backend APIs
- Ensure responsive design and accessibility
- Write frontend tests

10.2 Backend Engineers (3)

- Implement Go microservices
- Design and optimize database schema
- Integrate with external APIs (Vertex AI, OAuth)
- Write backend tests
- Monitor performance and optimize

10.3 DevOps Engineer (1)

- Set up and maintain infrastructure (Terraform)
- Configure CI/CD pipelines
- Set up monitoring and alerting
- Manage deployments
- Ensure system security and reliability

10.4 QA Engineer (1)

- Write and execute test plans
- Perform manual testing
- Write automated e2e tests
- Report and track bugs
- Verify fixes

10.5 AI/ML Specialist (1)

- Design and optimize AI prompts
- Integrate Vertex AI API
- Monitor AI quality metrics
- Optimize costs (caching, batching)
- Research and evaluate new models

11. Risk Management

11.1 Technical Risks

Risk	Likelihood	Impact	Mitigation
Vertex AI API reliability issues	Medium	High	Implement retry logic, fallback to cached responses
Database performance bottlenecks	Medium	Medium	Optimize queries, add indexes, implement caching
Security vulnerabilities	Low	High	Regular security audits, penetration testing
AI evaluation quality issues	Medium	High	Maintain test dataset, human review sampling
Vendor lock-in (GCP)	Low	Medium	Abstract cloud services, use standard APIs

11.2 Schedule Risks

Risk	Likelihood	Impact	Mitigation
Underestimated complexity	Medium	Medium	Build buffer time, prioritize ruthlessly
Team member unavailability	Medium	Medium	Cross-train team members, document thoroughly

Risk	Likelihood	Impact	Mitigation
Scope creep	High	High	Strict change control process, defer to Phase 2
Third-party API changes	Low	Medium	Monitor provider announcements, version lock

11.3 Adoption Risks

Risk	Likelihood	Impact	Mitigation
Poor user adoption	Medium	High	Extensive user testing, training materials
Resistance from coaches	Medium	Medium	Early involvement, demonstrate value
Student privacy concerns	Low	High	FERPA compliance, transparent privacy policy
Competition from existing tools	Medium	Medium	Focus on unique VEX robotics features

12. Success Metrics (KPIs)

12.1 Technical KPIs (Post-Launch)

- **System Uptime**: 99.5%
- **API Response Time (P95)**: $<$ 200ms
- **Error Rate**: $<$ 0.1%
- **Test Coverage**: 80%
- **AI Evaluation Completion**: $<$ 5 minutes

12.2 User Engagement KPIs

- **Weekly Active Users**: 80% of enrolled students
- **Submissions per Student**: 2 per week
- **Assessment Completion Rate**: 90%
- **Forum Participation**: 50% of students post at least once per month
- **User Satisfaction (NPS)**: 50

12.3 Educational Impact KPIs

- **Student Improvement**: 20% average score increase over semester

- **Coach Time Savings**: 50% reduction in evaluation time
- **Parent Engagement**: 70% of parents log in monthly
- **Students Showing Improvement**: 80%

13. Post-MVP Roadmap (Phase 2)

Months 10-12: Enhancements - AI tutoring chatbot - Advanced analytics and insights - Mobile app (React Native) - Integration with LMS (Canvas, Schoology)

Months 13-18: Expansion - Support for additional robotics platforms (FIRST, FTC) - Peer code review platform - Virtual robotics simulation integration - Tournament and competition management

Months 19-24: Enterprise Features - White-label solution for schools/districts - Advanced reporting and BI tools - Skills certification system - Industry partner integrations

---

Document Version: 1.0 **Last Updated**: 2025-10-18 **Status**: Ready for Team Review