SkillSync

SkillSync is a competitive programming platform that connects businesses with skilled developers through coding contests. Similar to platforms like TopCoder or HackerRank, but focused on real-world business problems, our platform allows companies to post development challenges while developers compete for prizes and recognition.

Technology Stack: MERN (MongoDB, Express.js, React.js/Next.js, Node.js)
Target Audience: Businesses seeking development solutions, freelance developers, and coding enthusiasts
Project Type: Full-stack web application capstone project

Problem Statement

Current freelancing platforms often suffer from:

- Race-to-the-bottom pricing
- Difficulty assessing developer skills
- Limited quality control
- Poor matching between client needs and developer capabilities

SkillSync addresses these issues by:

- Creating skill-based competitions
- Ensuring quality through competitive selection
- Offering fair compensation for winning solutions

Project Objectives

Primary Goals

1. **Create a functional contest platform** where businesses can post coding challenges
2. **Enable developer participation** through a competitive submission system
3. **Provide real-time communication** between clients and developers

Target Users

Primary Users

Clients (Challenge Creators)

- Startups needing MVP development
- Companies requiring specific features
- Businesses seeking innovative solutions
- Organizations with coding challenges

Developers (Participants)

- Freelance developers seeking opportunities
- Students building portfolios
- Professionals wanting to showcase skills
- Coding enthusiasts enjoy competitions

Super Users

- Platform administrators
- Community moderators

Core Features Specification

1. User Authentication & Management

Registration & Login

- Email/password authentication
- Email verification system
- Password reset functionality

User Profiles

- Separate profiles for clients and developers
- Portfolio showcase for developers
- Company information for clients
- Skill tags and experience levels
- Rating and reputation system

Access Control

- Role-based permissions (Client/Developer/Admin)
- Protected routes based on user type
- Session management and security

2. Challenge Management System

Challenge Creation (Client Interface)

- Technical requirements specification
- Prize amount
- Contest duration and deadlines
- Judging criteria definition

- Sample inputs/outputs
- Resource file attachments

Challenge Lifecycle

- Draft → Published → Active → Judging → Completed
- Automatic status transitions
- Deadline enforcement
- Extension capabilities

3. Submission System

Developer Submission Portal

- GitHub repository integration
- Live demo URL submission
- Documentation requirements
- Submission history tracking

File Management

- Secure file upload validation
- Support for various file types

4. Judging & Evaluation

Client Review Dashboard

- Submission gallery with previews
- Filtering and sorting options
- Rating system (1-5 stars)
- Detailed feedback forms

- Winner selection interface

Community Features

- Public voting options
- Community comments

Technical Architecture

Frontend Architecture (Next.js)

```
src/
├── app/
│   ├── layout.tsx
│   ├── page.tsx
│   ├── client/
│   │   └── dashboard/page.tsx
│   ├── developer/
│   │   └── dashboard/page.tsx
│   ├── admin/
│   │   └── panel/page.tsx
│   ├── challenges/
│   │   ├── [id]/page.tsx
│   │   └── create/page.tsx
│   └── submissions/
├── components/
│   ├── auth/
│   │   ├── LoginForm.tsx
│   │   ├── RegisterForm.tsx
│   ├── challenges/
│   │   ├── ChallengeCard.tsx
│   │   ├── ChallengeForm.tsx
│   │   └── ChallengeList.tsx
│   ├── submissions/
│   │   ├── SubmissionForm.tsx
```

```
│   │   ├── SubmissionList.tsx
│   ├── ui/              # Shared components
│   │   ├── Button.tsx
│   │   ├── Input.tsx
│   │   ├── LoadingSpinner.tsx
│   │   └── Modal.tsx
│   └── layout/
│       ├── Header.tsx
│       ├── Footer.tsx
│       └── Sidebar.tsx
├── hooks/
│   ├── useAuthStore.ts
│   └── useSubmission.ts
├── services/
│   ├── apiClient.ts
│   └── authService.ts
├── store/
│   ├── authStore.ts
│   └── challengeStore.ts
├── utils/
│   ├── validation.ts
│   └── helpers.ts
├── styles/
│   ├── globals.css
│   └── tailwind.config.ts
```

Key React Technologies:

- Zustand for state management
- Custom hooks for API integration
- Socket.io-client for real-time features
- React-hook-form + Zod for form validation
- React Query for data fetching
- Components library (e.g, DaisyUI)

Backend Architecture (Node.js + Express)

server/

├── models/

│ ├── User.ts

│ ├── Challenge.ts

│ ├── Submission.ts

│ └── Notification.ts

├── routes/

│ ├── auth.ts

│ ├── challenges.ts

│ ├── submissions.ts

│ └── users.ts

├── middleware/

│ ├── auth.ts

```
│   ├── validation.ts
│   ├── upload.ts
│   └── rateLimiting.ts
├── controllers/
│   ├── authController.ts
│   ├── challengeController.ts
│   ├── submissionController.ts
├── utils/
│   ├── emailService.ts
│   ├── fileUpload.ts
│   └── socketHandlers.ts
├── config/
│   ├── database.ts
│   ├── cloudinary.ts
└── tests/
    ├── unit/
    └── integration/
```

Key Backend Technologies:

- Express.js framework

- MongoDB with Mongoose ODM
- JWT for authentication
- Bcrypt for password hashing
- Multer for file uploads
- Socket.io for real-time features
- Nodemailer for email services
- Jest for testing

Database Schema Design

Users Collection

```
{
  _id: ObjectId,
  email: String (unique),
  password: String (hashed),
  role: String ['client', 'developer', 'admin'],
  profile: {
    firstName: String,
    lastName: String,
    avatar: String,
    bio: String,
    skills: [String],
    experience: String,
```

```
    portfolio: [Object],
    socialLinks: Object
  },
  reputation: {
    rating: Number,
    totalRatings: Number,
    completedChallenges: Number
  },
  createdAt: Date,
  updatedAt: Date,
  isVerified: Boolean,
  lastLogin: Date
}
```

Challenges Collection

```
{
  _id: ObjectId,
  title: String,
  description: String,
  requirements: String,
```

```
  category: String,
  difficulty: String ['beginner', 'intermediate',
'advanced'],
  prize: Number,
  deadline: Date,
  status: String ['draft', 'published', 'active',
'judging', 'completed'],
  client: ObjectId (ref: User),
  submissions: [ObjectId] (ref: Submission),
  judgingCriteria: {
    codeQuality: Number,
    functionality: Number,
    creativity: Number,
    documentation: Number
  },
  files: [Object],
  tags: [String],
  viewCount: Number,
  participantCount: Number,
```

```
  createdAt: Date,
  updatedAt: Date
}
```

Submissions Collection

```
{
  _id: ObjectId,
  challenge: ObjectId (ref: Challenge),
  developer: ObjectId (ref: User),
  files: [Object],
  githubRepo: String,
  liveDemo: String,
  description: String,
  documentation: String,
  submittedAt: Date,
  version: Number,
  status: String ['pending', 'reviewed', 'winner',
'rejected'],
  ratings: {
    codeQuality: Number,
```

```
    functionality: Number,
    creativity: Number,
    documentation: Number,
    overall: Number
  },
  feedback: String,
  isWinner: Boolean
}
```

API Design

Authentication Endpoints

- **POST /api/auth/register User registration**
- **POST /api/auth/login User login**
- POST /api/auth/logout User logout
- POST /api/auth/forgot-password Password reset
- POST /api/auth/verify-email Email verification

Challenge Endpoints

- **GET /api/challenges List all challenges**
- **POST /api/challenges Create a new challenge**
- **GET /api/challenges/:id Get challenge details**
- PUT /api/challenges/:id Update challenge
- DELETE /api/challenges/:id Delete challenge
- POST /api/challenges/:id/join Join challenge

Submission Endpoints

- **POST /api/submissions Submit solution**
- GET /api/submissions/:challengeId Get challenge submissions
- PUT /api/submissions/:id Update submission
- DELETE /api/submissions/:id Delete submission
- POST /api/submissions/:id/rate Rate submission

User Endpoints

- GET /api/users/profile Get user profile

- PUT /api/users/profile Update profile
- GET /api/users/:id Get public profile
- POST /api/users/upload-avatar Upload avatar

Security Implementation

Authentication Security

- JWT tokens with expiration
- Refresh token rotation
- Password strength requirements
- Rate limiting on auth endpoints
- Account lockout after failed attempts

Data Protection

- Input validation and sanitization
- Secure file upload validation

Testing Strategy [Optional]

Unit Testing

- Component testing with React Testing Library
- API endpoint testing with Jest, super test
- Database model testing
- Utility function testing

Deployment Strategy [Optional]

Development Environment

- Local MongoDB instance
- Node.js development server
- React development server
- Environment variable management

Production Environment

- **Frontend**: Vercel or Netlify
- **Backend**: Railway, Render, or AWS EC2
- **Database**: MongoDB Atlas
- **File Storage**: AWS S3 or Cloudinary or local file system
- **Email Service**: SendGrid or Mailgun

CI/CD Pipeline

- Automated deployment on merge and code push

Conclusion

SkillSync represents a comprehensive full-stack development project that demonstrates mastery of the MERN stack while addressing real market needs. The platform combines complex technical challenges with practical business applications, making it an ideal capstone project for showcasing advanced development skills.

The project provides extensive learning opportunities in areas such as real-time communications, file management, and scalable architecture design. Upon completion, students will have developed a production-ready application suitable for portfolio presentation and potential commercial deployment.