SynapShare: Comprehensive Project Report

Executive Summary

SynapShare is a collaborative knowledge-sharing platform designed for developers and students to exchange programming knowledge, notes, and engage in technical discussions. The application follows a modern full-stack architecture with distinct frontend and backend components built on the MERN stack (MongoDB, Express, React, Node.js).

---

1. Project Overview

1.1 Purpose

SynapShare provides a dedicated platform for technical knowledge exchange with specialized features for developers and students:

- Share programming notes and resources
- Create and participate in technical discussions
- Exchange code snippets and examples
- Build a community-driven knowledge base

1.2 Target Audience

- Software developers
- Computer science students
- Technical instructors
- Programming enthusiasts

---

2. System Architecture

2.1 Project Structure

```
SynapShare/
├── synapshare-frontend/      # React-based frontend
│   ├── public/               # Static assets
│   ├── src/                  # React components and logic
│   └── package.json          # Frontend dependencies
├── synapshare-backend/       # Express server backend
│   ├── routes/               # API route definitions
│   ├── models/               # MongoDB schema models
│   ├── uploads/              # File storage for user uploads
│   ├── server.js             # Main server entry point
│   └── package.json          # Backend dependencies
```

2.2 Core Technologies

Frontend

- **React**: UI library for component-based architecture
- **React Router**: For navigation between different sections
- **Tailwind CSS**: For styling the application
- **Axios**: For making API requests to the backend
- **React TSParticles**: For dynamic background effects

Backend

- **Express.js**: Web server framework
- **MongoDB/Mongoose**: Database and ORM
- **Firebase Authentication**: User management and authentication
- **Multer**: File upload handling
- **Nodemailer**: Email functionality

---

3. Key Features & Implementation

3.1 Authentication System

SynapShare uses Firebase for authentication, with custom middleware that verifies tokens and synchronizes user data with MongoDB.

Key Authentication Features:

- JWT token verification

- Automatic user creation in MongoDB
- Special admin privileges detection
- Username management

javascript

```javascript
// Firebase Authentication (server.js)
firebaseAdmin.initializeApp({
  credential: firebaseAdmin.credential.cert(
    JSON.parse(process.env.FIREBASE_SERVICE_ACCOUNT)
  ),
});

// Middleware to verify Firebase token and fetch username
const verifyToken = async (req, res, next) => {
  const token = req.headers.authorization?.split("Bearer ")[1];
  if (!token) return res.status(401).json({ error: "No token provided" });

  try {
    const decoded = await firebaseAdmin.auth().verifyIdToken(token);
    req.user = decoded;

    let user = await User.findOne({ uid: decoded.uid });
    if (!user) {
      user = new User({
        uid: decoded.uid,
        email: decoded.email,
      });
      await user.save();
      console.log(`New user created: ${decoded.email}`);
    }

    // Admin override: Force username for admin if not set
    const isAdmin = decoded.email === "porwalkamlesh5@gmail.com";
    req.isAdmin = isAdmin;
    req.username = isAdmin ? user.username || "admin" : user.username || null;

    next();
  } catch (error) {
    console.error("Token verification error:", error.message, error.stack);
    res.status(401).json({ error: "Invalid token" });
  }
};
```

3.2 Content Management System

The platform offers three main content types, each with their own MongoDB schemas:

Notes

For sharing lecture notes, study guides, and educational resources

```javascript
const noteSchema = new mongoose.Schema({
  title: String,
  fileUrl: String,
  uploadedBy: String,
  subject: String,
  createdAt: { type: Date, default: Date.now },
  text: { type: String, index: "text" },
  upvotes: { type: Number, default: 0 },
  downvotes: { type: Number, default: 0 },
  voters: [
    {
      username: String,
      voteType: { type: String, enum: ["upvote", "downvote"] },
    },
  ],
  comments: [
    {
      content: String,
      postedBy: String,
      createdAt: { type: Date, default: Date.now },
    },
  ],
});
```

Discussions

For technical questions, debates, and collaborative problem-solving

```javascript
const discussionSchema = new mongoose.Schema({
  title: String,
  content: String,
  fileUrl: String,
  postedBy: String,
  createdAt: { type: Date, default: Date.now },
  text: { type: String, index: "text" },
  upvotes: { type: Number, default: 0 },
  downvotes: { type: Number, default: 0 },
  voters: [/* same as notes */],
  comments: [/* same as notes */],
});
```

Code Nodes

For sharing code snippets, algorithms, and programming examples

```javascript
const nodeSchema = new mongoose.Schema({
  title: String,
  description: String,
  codeSnippet: String,
  fileUrl: String,
  postedBy: String,
  createdAt: { type: Date, default: Date.now },
  text: { type: String, index: "text" },
  upvotes: { type: Number, default: 0 },
  downvotes: { type: Number, default: 0 },
  voters: [/* same as notes */],
  comments: [/* same as notes */],
});
```

3.3 File Upload System

SynapShare implements a robust file upload system using Multer:

Key Features:

- Secure file storage with uniquely generated filenames
- File type validation for security

- 50MB file size limit
- Support for images, PDFs, and videos

javascript

```javascript
// Multer for file uploads
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    const uploadDir = "uploads/";
    if (!fs.existsSync(uploadDir)) {
      fs.mkdirSync(uploadDir, { recursive: true });
      console.log("Created uploads directory");
    }
    cb(null, uploadDir);
  },
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + "-" + Math.round(Math.random() * 1e9);
    cb(null, uniqueSuffix + path.extname(file.originalname));
  },
});

const fileFilter = (req, file, cb) => {
  const allowedTypes = [
    "image/jpeg",
    "image/png",
    "image/gif",
    "application/pdf",
    "video/mp4",
    "video/webm",
  ];
  if (allowedTypes.includes(file.mimetype)) {
    cb(null, true);
  } else {
    cb(new Error("Only images, PDFs, and MP4/WebM videos are allowed"), false);
  }
};

const upload = multer({
  storage,
  fileFilter,
  limits: { fileSize: 50 * 1024 * 1024 }, // 50MB limit
});
```

3.4 Voting and Engagement System

All content types include a voting system to promote quality content:

Key Features:

- Toggle functionality (clicking the same vote button twice removes the vote)
- Vote type switching (from upvote to downvote or vice versa)
- User tracking to prevent multiple votes from the same user
- Vote count maintenance

```javascript
// Example of upvote implementation (for notes)
app.post("/api/notes/:id/upvote", verifyToken, async (req, res) => {
  try {
    if (!req.isAdmin && !req.username) {
      return res.status(403).json({ error: "Please set a username" });
    }
    const note = await Note.findById(req.params.id);
    if (!note) return res.status(404).json({ error: "Note not found" });

    const existingVote = note.voters.find((v) => v.username === req.username);
    if (existingVote) {
      if (existingVote.voteType === "upvote") {
        note.upvotes -= 1;
        note.voters = note.voters.filter((v) => v.username !== req.username);
      } else {
        note.downvotes -= 1;
        note.upvotes += 1;
        existingVote.voteType = "upvote";
      }
    } else {
      note.upvotes += 1;
      note.voters.push({ username: req.username, voteType: "upvote" });
    }

    await note.save();
    res.json(note);
  } catch (error) {
    console.error("Error upvoting note:", error.message);
    res.status(500).json({ error: "Failed to upvote note" });
  }
});
```

3.5 Search Functionality

The platform provides integrated search functionality across all content types:

javascript

```javascript
app.get("/api/search", async (req, res) => {
  try {
    const { q } = req.query;
    const notes = await Note.find({ $text: { $search: q } });
    const discussions = await Discussion.find({ $text: { $search: q } });
    const nodes = await Node.find({ $text: { $search: q } });
    res.json({ notes, discussions, nodes });
  } catch (error) {
    console.error("Error searching:", error.message);
    res.status(500).json({ error: "Failed to search" });
  }
});
```

The search functionality uses MongoDB's text indexing capability, made possible by the text field in each content schema with an explicit text index.

3.6 Save Posts Feature

Users can save content for later reference:

javascript

```javascript
app.post("/api/savedPosts", verifyToken, async (req, res) => {
  try {
    if (!req.isAdmin && !req.username) {
      return res.status(403).json({ error: "Please set a username" });
    }
    const { postType, postId } = req.body;
    const savedPost = new SavedPost({
      userEmail: req.user.email,
      postType,
      postId,
    });
    await savedPost.save();
    res.status(201).json(savedPost);
  } catch (error) {
    console.error("Error saving post:", error.message);
    res.status(500).json({ error: "Failed to save post" });
  }
});
```

3.7 Admin Controls

The platform includes special administrative controls:

```javascript
// Admin middleware
const isAdmin = (req, res, next) => {
  if (req.isAdmin) {
    next();
  } else {
    res.status(403).json({ error: "Admin access required" });
  }
};

// Example admin endpoint for content moderation
app.delete("/api/admin/notes/:id", verifyToken, isAdmin, async (req, res) => {
  try {
    const note = await Note.findById(req.params.id);
    if (!note) return res.status(404).json({ error: "Note not found" });
    if (note.fileUrl) {
      const filePath = path.join(
        __dirname,
        note.fileUrl.replace("http://localhost:5000", "")
      );
      if (fs.existsSync(filePath)) fs.unlinkSync(filePath);
    }
    await note.deleteOne();
    console.log(`Note deleted by admin: ${req.params.id}`);
    res.json({ message: "Note deleted successfully" });
  } catch (error) {
    console.error("Error deleting note (admin):", error.message);
    res.status(500).json({ error: "Failed to delete note" });
  }
});
```

---

4. Data Flow and User Experience

4.1 User Registration & Authentication Flow

1. Users register via Firebase authentication
2. Upon successful authentication, a JWT token is issued
3. The token is verified by the backend middleware
4. New users are automatically created in MongoDB

4.2 Content Creation Flow

1. User creates content (note, discussion, or code node)
2. Files can be attached during creation
3. Content is stored in MongoDB with references to file locations
4. Text fields are indexed for search functionality

4.3 Content Interaction Flow

1. Users can view, upvote/downvote, comment on content
2. Each interaction is tracked to prevent duplicate actions
3. Users can save content for later reference

4.4 Admin Moderation Flow

1. Admins have special privileges to delete any content
2. Admin status is determined by email address verification
3. Admin actions are logged for accountability

---

5. Technical Innovations

5.1 Integrated Authentication System

SynapShare combines Firebase Authentication with MongoDB user management, providing a secure yet flexible user system.

5.2 Unified Content Model

Despite having different content types, the platform maintains a consistent schema structure for voting, comments, and metadata.

5.3 Flexible File Storage

The application handles multiple file types and sizes, with appropriate security measures in place.

5.4 Cross-Content Search

The text indexing approach allows for efficient searching across all content types simultaneously.

---

6. Deployment Considerations

For a production deployment, the following adjustments would be needed:

6.1 Environment Variables

Replace hardcoded URLs with environment variables:

```javascript
const BASE_URL = process.env.NODE_ENV === 'production'
  ? process.env.API_URL
  : 'http://localhost:5000';
```

6.2 Backend Start Scripts

Add proper start script in backend package.json:

```json
"scripts": {
  "start": "node server.js",
  "dev": "nodemon server.js"
}
```

6.3 Secure Database Connection

Set up secure MongoDB connection:

```javascript
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  serverSelectionTimeoutMS: 5000
})
```

6.4 CORS Configuration

Configure production-ready CORS:

```javascript
const allowedOrigins = ['https://synapshare.com', 'https://admin.synapshare.com'];
app.use(cors({
  origin: function(origin, callback) {
    if (!origin || allowedOrigins.includes(origin)) {
      callback(null, true);
    } else {
      callback(new Error('Not allowed by CORS'));
    }
  }
}));
```

---

7. Conclusion

SynapShare represents a comprehensive knowledge-sharing platform with robust features for developers and students. Its architecture demonstrates a modern approach to full-stack development, with clear separation of concerns between frontend and backend components.

The integration of Firebase for authentication with a custom MongoDB-based user system provides both security and flexibility, while the content management system offers rich functionality for different types of knowledge sharing.

The platform's voting system, search capabilities, and file management create a powerful ecosystem for technical knowledge exchange that can benefit the developer and student communities.

---

8. Presentation Key Points

Architecture Highlights

- Full MERN stack implementation
- Clean separation of frontend/backend concerns
- Scalable database design with MongoDB

Feature Highlights

- Three distinct content types (Notes, Discussions, Code Nodes)
- Robust file upload system supporting multiple formats
- Integrated voting and commenting system
- Advanced search capabilities

- Admin moderation tools

Security Highlights

- Firebase authentication integration
- Token verification middleware
- File type validation
- Admin role management

User Experience Highlights

- Ability to save favorite content
- Content voting for quality control
- Rich commenting system
- Cross-content search functionality