SynapShare Backend — Complete Technical Walkthrough

Table of Contents

Introduction

SynapShare is a full-stack web application that allows users to share notes, engage in discussions, and collaborate on educational content. This document provides a comprehensive technical walkthrough of the backend implementation, including code examples and explanations.

Technologies Used

- **Node.js**: JavaScript runtime for building scalable server-side applications
- **Express.js**: Web framework for Node.js, used to handle HTTP requests and routing
- **MongoDB**: NoSQL database for storing user data, notes, discussions, etc.
- **Mongoose**: ODM (Object Data Modeling) library for MongoDB, allows defining schemas and models
- **Firebase Admin SDK**: Handles authentication (JWT verification, password resets)
- **Multer**: Middleware for handling file uploads (notes, discussions, code files)
- **dotenv**: Loads environment variables from a .env file
- **CORS**: Allows cross-origin requests (frontend-backend communication)
- **Axios**: For making HTTP requests to external APIs (e.g., NewsAPI)
- **fs, path**: Node.js modules for file system operations
- **Other**: Tailwind CSS, Framer Motion, etc. are used on the frontend

Project Structure

```
synapshare-backend/
├── server.js               # Main server file (core logic and routes)
├── routes/                 # Modular route handlers (notes, discussions, nodes, users)
├── models/                 # Mongoose schemas for MongoDB collections
├── uploads/                # Directory for uploaded files
├── .env                    # Environment variables (DB URI, API keys, etc.)
└── package.json            # Project dependencies and scripts
```

Setting Up the Server

Loading Environment Variables and Initializing Services

```javascript
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const dotenv = require("dotenv");
const firebaseAdmin = require("firebase-admin");

dotenv.config(); // Loads variables from .env (like DB URI, Firebase credentials)

firebaseAdmin.initializeApp({
  credential: firebaseAdmin.credential.cert(
    JSON.parse(process.env.FIREBASE_SERVICE_ACCOUNT)
  ),
}); // Initializes Firebase for authentication

mongoose.connect(process.env.MONGO_URI)
  .then(() => console.log("Connected to MongoDB"))
  .catch((err) => console.error("MongoDB connection error:", err));
// Connects to the MongoDB database
```

Explanation:

- `dotenv` loads sensitive information from `.env` so you don't hardcode secrets
- Firebase Admin SDK is set up using credentials from `.env` for secure user authentication
- Mongoose connects to MongoDB for persistent data storage

Express App Setup and Middleware

```javascript
const app = express();
app.use(cors()); // Allows frontend to make requests to backend
app.use(express.json()); // Parses JSON request bodies
app.use("/uploads", express.static("uploads")); // Serves uploaded files
```

Explanation:

- `cors()` lets your React frontend access the backend API
- `express.json()` parses incoming JSON data
- Static middleware serves files (like uploaded PDFs or images) from the `uploads/` directory

File Uploads with Multer

Configuring Multer for File Uploads

```javascript
const multer = require("multer");
const fs = require("fs");
const path = require("path");

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    if (!fs.existsSync("uploads")) fs.mkdirSync("uploads");
    cb(null, "uploads/");
  },
  filename: (req, file, cb) => {
    const ext = path.extname(file.originalname);
    cb(null, Date.now() + "-" + Math.round(Math.random() * 1e9) + ext);
  },
});

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("Invalid file type"), false);
};

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

Explanation:

- `multer.diskStorage` saves files in `/uploads` with unique names
- Only certain file types (images, PDFs, videos) are allowed
- Files larger than 50MB are rejected

Defining Data Models with Mongoose

User Model

```javascript
const mongoose = require("mongoose");
const UserSchema = new mongoose.Schema({
  uid: { type: String, required: true, unique: true },
  username: { type: String, unique: true },
  email: { type: String, required: true },
  createdAt: { type: Date, default: Date.now },
});
module.exports = mongoose.model("User", UserSchema);
```

Explanation:

- Each user has a unique Firebase UID, a username, and an email
- `createdAt` is set automatically

Note Model

```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" }, // For search
  upvotes: { type: Number, default: 0 },
  downvotes: { type: Number, default: 0 },
  voters: [{ username: String, voteType: String }],
  comments: [{ content: String, postedBy: String, createdAt: Date }],
});
module.exports = mongoose.model("Note", NoteSchema);
```

Explanation:

- Stores all details about a note, including file URL, uploader, votes, and comments
- `text` is indexed for efficient searching

Authentication and Authorization

JWT Verification Middleware

```javascript
async function verifyToken(req, res, next) {
  const authHeader = req.headers.authorization;
  if (!authHeader) return res.status(401).json({ message: "No token provided" });
  const token = authHeader.split(" ")[1];
  try {
    const decoded = await firebaseAdmin.auth().verifyIdToken(token);
    req.uid = decoded.uid;
    const user = await User.findOne({ uid: req.uid });
    if (!user) return res.status(403).json({ message: "User not found" });
    req.username = user.username;
    req.isAdmin = user.email.endsWith("@admin.com"); // Example admin logic
    req.user = user;
    next();
  } catch (err) {
    res.status(403).json({ message: "Invalid token" });
  }
}
```

Explanation:

- Extracts JWT from the Authorization header
- Verifies it using Firebase
- Looks up the user in MongoDB and attaches their info to the request
- Blocks requests if authentication fails

API Endpoints and Their Logic

User Management

```javascript
app.get("/api/user/:uid", verifyToken, async (req, res) => {
  const user = await User.findOne({ uid: req.params.uid });
  if (!user) return res.status(404).json({ message: "User not found" });
  res.json(user);
});
```

Purpose: Returns the profile of the authenticated user.

```javascript
app.post("/api/check-username", async (req, res) => {
  const { username } = req.body;
  const exists = await User.findOne({ username });
  res.json({ exists: !!exists });
});
```

Purpose: Checks if a username is already taken.

```javascript
app.post("/api/save-username", verifyToken, async (req, res) => {
  const { username } = req.body;
  if (await User.findOne({ username })) {
    return res.status(400).json({ message: "Username taken" });
  }
  req.user.username = username;
  await req.user.save();
  res.json({ message: "Username saved" });
});
```

Purpose: Allows a user to set their username after authentication.

Notes CRUD Operations

```javascript
app.get("/api/notes", async (req, res) => {
  const notes = await Note.find();
  res.json(notes);
});
```

Purpose: Fetches all notes.

javascript

```
app.post("/api/notes", verifyToken, upload.single("file"), async (req, res) => {
  const { title, subject } = req.body;
  const fileUrl = req.file ? `${req.protocol}://${req.get("host")}/uploads/${req.file.filename}
  const note = new Note({
    title,
    subject,
    fileUrl,
    uploadedBy: req.username,
    text: `${title} ${subject}`,
  });
  await note.save();
  res.status(201).json(note);
});
```

Purpose: Creates a new note, optionally with a file.

javascript

```
app.put("/api/notes/:id", verifyToken, upload.single("file"), async (req, res) => {
  const note = await Note.findById(req.params.id);
  if (!note) return res.status(404).json({ message: "Not found" });
  if (note.uploadedBy !== req.username && !req.isAdmin) {
    return res.status(403).json({ message: "Not authorized" });
  }
  if (req.file && note.fileUrl) {
    const filePath = path.join(__dirname, note.fileUrl.replace(`${req.protocol}://${req.get("ho
    if (fs.existsSync(filePath)) fs.unlinkSync(filePath);
    note.fileUrl = `${req.protocol}://${req.get("host")}/uploads/${req.file.filename}`;
  }
  note.title = req.body.title || note.title;
  note.subject = req.body.subject || note.subject;
  await note.save();
  res.json(note);
});
```

Purpose: Updates a note, replaces file if a new one is uploaded.

javascript

```javascript
app.delete("/api/notes/:id", verifyToken, async (req, res) => {
  const note = await Note.findById(req.params.id);
  if (!note) return res.status(404).json({ message: "Not found" });
  if (note.uploadedBy !== req.username && !req.isAdmin) {
    return res.status(403).json({ message: "Not authorized" });
  }
  if (note.fileUrl) {
    const filePath = path.join(__dirname, note.fileUrl.replace(`${req.protocol}://${req.get("ho
    if (fs.existsSync(filePath)) fs.unlinkSync(filePath);
  }
  await note.deleteOne();
  res.json({ message: "Note deleted" });
});
```

Purpose: Deletes a note and its file (if present).

Voting System

javascript

```javascript
app.post("/api/notes/:id/upvote", verifyToken, async (req, res) => {
  const note = await Note.findById(req.params.id);
  if (!note) return res.status(404).json({ message: "Not found" });
  const existingVote = note.voters.find(v => v.username === req.username);
  if (existingVote) {
    if (existingVote.voteType === "upvote") {
      note.upvotes--;
      note.voters = note.voters.filter(v => v.username !== req.username);
    } else {
      note.upvotes++;
      note.downvotes--;
      existingVote.voteType = "upvote";
    }
  } else {
    note.upvotes++;
    note.voters.push({ username: req.username, voteType: "upvote" });
  }
  await note.save();
  res.json(note);
});
```

Purpose: Handles upvoting logic, prevents duplicate votes, and allows toggling.

Comments

javascript

```javascript
app.post("/api/notes/:id/comment", verifyToken, async (req, res) => {
  const note = await Note.findById(req.params.id);
  if (!note) return res.status(404).json({ message: "Not found" });
  note.comments.push({
    content: req.body.content,
    postedBy: req.username,
    createdAt: new Date(),
  });
  await note.save();
  res.json(note.comments);
});
```

Purpose: Adds a comment to a note.

Saved Posts (Bookmarks)

javascript

```javascript
app.post("/api/savedPosts", verifyToken, async (req, res) => {
  const { postType, postId } = req.body;
  const saved = new SavedPost({
    userEmail: req.user.email,
    postType,
    postId,
    createdAt: new Date(),
  });
  await saved.save();
  res.json(saved);
});
```

Purpose: Saves a post for quick access.

javascript

```javascript
app.get("/api/savedPosts", verifyToken, async (req, res) => {
  const saved = await SavedPost.find({ userEmail: req.user.email });
  res.json(saved);
});
```

Purpose: Retrieves all saved posts for the user.

News API Integration

```javascript
const axios = require("axios");
app.get("/api/news", async (req, res) => {
  const url = `https://newsapi.org/v2/top-headlines?category=technology&apiKey=${process.env.NE
  const response = await axios.get(url);
  res.json(response.data.articles);
});
```

Purpose: Fetches and returns the latest technology news.

Password Reset

```javascript
app.post("/api/request-password-reset", async (req, res) => {
  const { email } = req.body;
  try {
    const link = await firebaseAdmin.auth().generatePasswordResetLink(email);
    // Send link via email (implementation depends on your mail setup)
    res.json({ message: "Password reset email sent" });
  } catch (err) {
    res.status(400).json({ message: "Failed to send reset email" });
  }
});
```

Purpose: Sends a password reset email to the user.

Search Endpoint

```javascript
app.get("/api/search", async (req, res) => {
  const q = req.query.q;
  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 });
});
```

Purpose: Searches all main resources for a given query.

Security and Error Handling

- All sensitive operations require authentication
- Only owners or admins can edit/delete resources
- File uploads are validated for type and size
- All DB/file operations are wrapped in try/catch blocks
- Errors are logged and clear messages are sent to the client

Summary

The SynapShare backend is a secure, modular REST API built with Node.js, Express, MongoDB, and Firebase. It supports:

- Authentication using Firebase
- File uploads with Multer
- CRUD operations for notes, discussions, and nodes
- Voting and commenting systems
- Bookmark functionality
- Admin controls
- News integration
- Search functionality

The code is organized for clarity, security, and ease of maintenance, making it a robust foundation for a collaborative learning platform.