Learning Platform - Implementation Documentation Report

1. File Upload Implementation

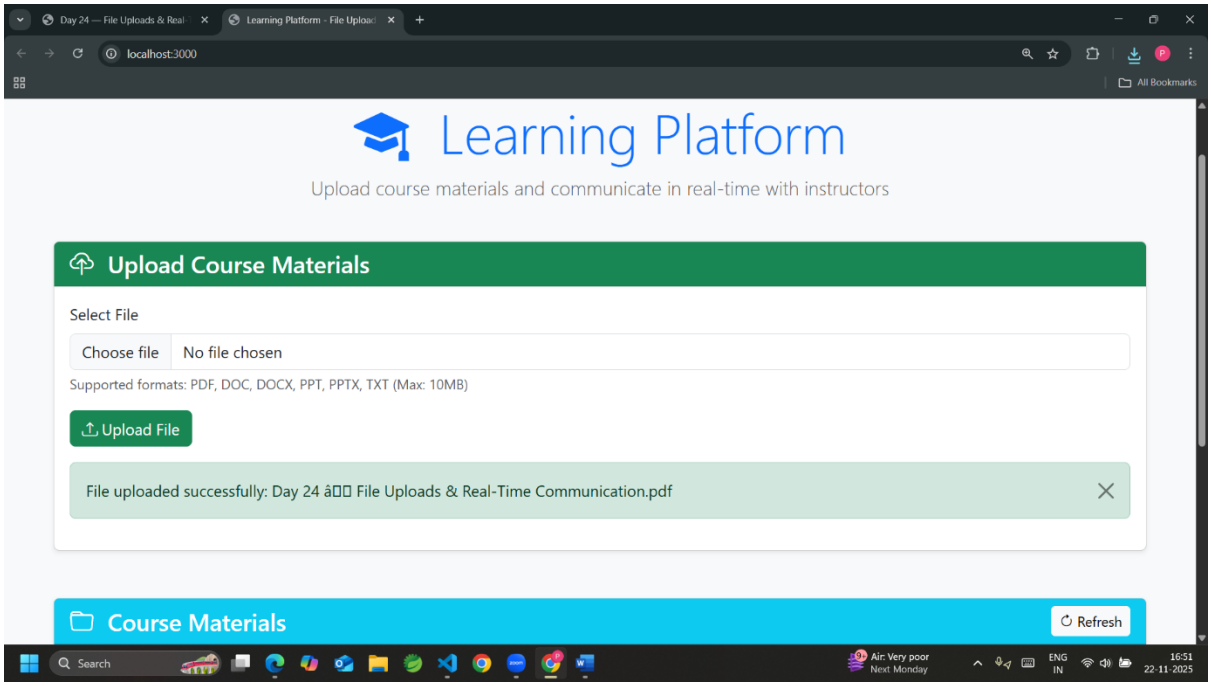
Implementation Approach

Used Multer middleware for handling multipart/form-data with security features including file type validation, size limits, and filename sanitization. Files are stored locally with timestamp prefixes to prevent naming conflicts.

Key Code Snippet

```
const upload = multer({
    storage: multer.diskStorage({
        destination: 'uploads/',
        filename: (req, file, cb) => {
            const sanitizedName = file.originalname.replace(/[^a-zA-Z0-9.\-
]/g, '_');
            cb(null, `${Date.now()}_${sanitizedName}`);
        }
    }),
    fileFilter: (req, file, cb) => {
        const allowedTypes = ['.pdf', '.doc', '.docx', '.txt'];
        const fileExtension = path.extname(file.originalname).toLowerCase();
        allowedTypes.includes(fileExtension) ? cb(null, true) : cb(new
Error('Invalid file type'));
    },
    limits: { fileSize: 10 * 1024 * 1024 } // 10MB
});
```

Test Output



2. Static File Serving Implementation

Implementation Approach

Implemented Express static middleware to serve uploaded files from the /uploads directory. Added secure headers for file downloads and created a RESTful API endpoint to list available materials with metadata.

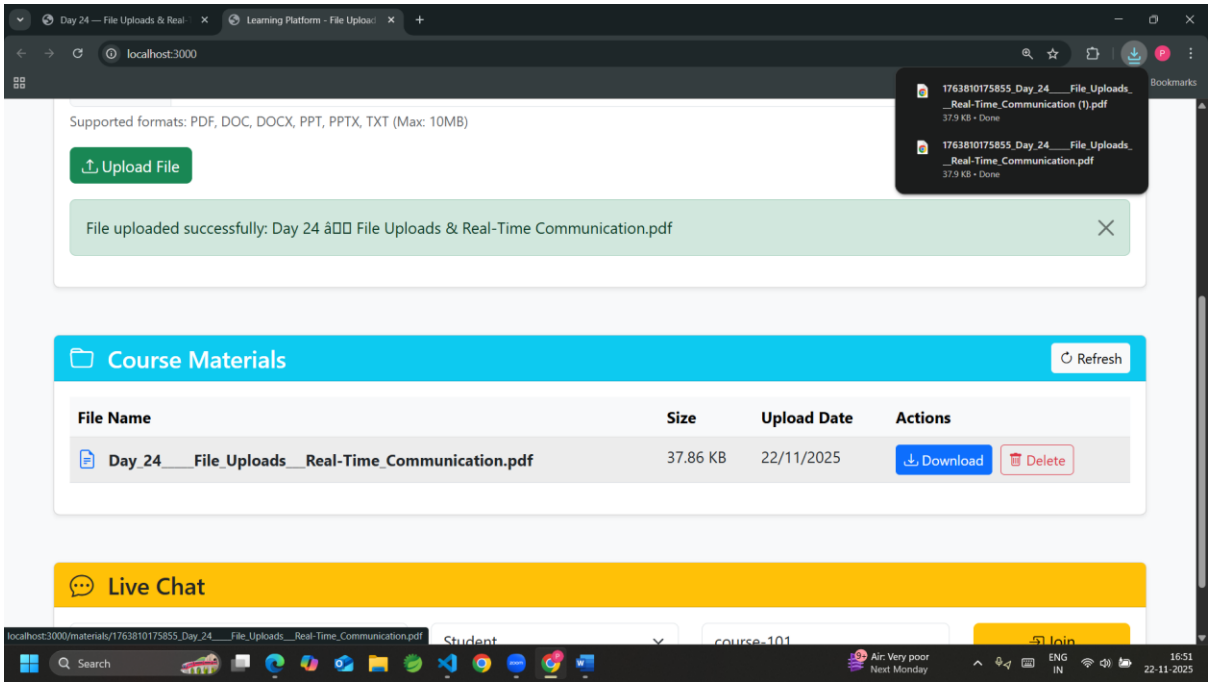
Key Code Snippet

```
// Serve static files with download headers
app.use('/materials', express.static('uploads', {
    setHeaders: (res, path) => {
        res.set('Content-Disposition', 'attachment');
    }
}));

// API to list materials
app.get('/api/materials', (req, res) => {
    fs.readdir(uploadsDir, (err, files) => {
        const materials = files.map(file => {
            const stats = fs.statSync(path.join(uploadsDir, file));
            return {
                filename: file,
                originalName: file.split('_').slice(1).join('_'),
                size: stats.size,
                downloadUrl: `/materials/${file}`
            };
        });
        res.json(materials);
```

```
    });
});
```

Test Output



3. Real-Time Chat Implementation

Implementation Approach

Integrated Socket.io for bidirectional communication using room-based architecture. Implemented typing indicators, user join/leave notifications, and role-based messaging with proper connection handling and error management.

Key Code Snippet

```
// Socket.io room management
io.on('connection', (socket) => {
    socket.on('join_course', (data) => {
        socket.join(data.courseId);
        socket.userData = data;
        socket.to(data.courseId).emit('user_joined', {
            userName: data.userName,
            message: `${data.userName} joined the chat`
        });
    });

    socket.on('send_message', (data) => {
        io.to(data.courseId).emit('new_message', {
            id: Date.now(),
            userName: data.userName,
            userType: data.userType,
```

```
            message: data.message,
            timestamp: new Date().toISOString()
        });
    });

    socket.on('typing_start', (data) => {
        socket.to(data.courseId).emit('user_typing', {
            userName: data.userName,
            isTyping: true
        });
    });
});
```

Test Output