Creating a boilerplate code or folder structure for an application like OLobby involves setting up a project that can handle various features such as task management, document management, chat, video calls, and more. Below is a suggested folder structure and some initial boilerplate code for a full-stack application that could be used as a starting point for OLobby.  
  
The structure assumes a web application using a JavaScript/TypeScript stack with React for the frontend and Node.js with Express for the backend. It also assumes the use of a database such as MongoDB for storing data.  
  
```  
OLobby/  
│  
├── client/ (frontend)  
│ ├── public/  
│ │ └── index.html  
│ ├── src/  
│ │ ├── components/  
│ │ │ ├── Dashboard/  
│ │ │ │ ├── GlobalDashboard.js  
│ │ │ │ └── ...  
│ │ │ ├── DocumentManagement/  
│ │ │ │ ├── FileUploader.js  
│ │ │ │ └── ...  
│ │ │ ├── Workflow/  
│ │ │ │ ├── WorkflowEditor.js  
│ │ │ │ └── ...  
│ │ │ ├── TaskManagement/  
│ │ │ │ ├── TaskList.js  
│ │ │ │ └── ...  
│ │ │ ├── Notes/  
│ │ │ │ ├── QuickNotes.js  
│ │ │ │ └── ...  
│ │ │ ├── Chat/  
│ │ │ │ ├── ChatWindow.js  
│ │ │ │ └── ...  
│ │ │ ├── VideoConference/  
│ │ │ │ ├── VideoCall.js  
│ │ │ │ └── ...  
│ │ │ └── SharedFiles/  
│ │ │ ├── FileShare.js  
│ │ │ └── ...  
│ │ ├── App.js  
│ │ ├── index.js  
│ │ └── ...  
│ ├── package.json  
│ └── ...  
│  
├── server/ (backend)  
│ ├── config/  
│ │ └── db.js  
│ ├── models/  
│ │ ├── User.js  
│ │ ├── Task.js  
│ │ ├── Document.js  
│ │ └── ...  
│ ├── routes/  
│ │ ├── auth.js  
│ │ ├── tasks.js  
│ │ ├── documents.js  
│ │ └── ...  
│ ├── controllers/  
│ │ ├── authController.js  
│ │ ├── taskController.js  
│ │ ├── documentController.js  
│ │ └── ...  
│ ├── middleware/  
│ │ └── authMiddleware.js  
│ ├── utils/  
│ │ └── ...  
│ ├── app.js  
│ ├── server.js  
│ └── ...  
│ ├── package.json  
│ └── ...  
│  
├── .gitignore  
├── README.md  
└── package.json (for managing scripts and workspaces)  
```  
  
Here's a basic example of what some of the files might contain:  
  
`client/src/App.js` (React frontend entry point):  
```javascript  
import React from 'react';  
import GlobalDashboard from './components/Dashboard/GlobalDashboard';  
// Import other components as needed  
  
function App() {  
 return (  
 <div className="App">  
 <GlobalDashboard />  
 {/\* Render other components based on routing \*/}  
 </div>  
 );  
}  
  
export default App;  
```  
  
`server/app.js` (Express backend setup):  
```javascript  
const express = require('express');  
const mongoose = require('mongoose');  
const cors = require('cors');  
const taskRoutes = require('./routes/tasks');  
const authRoutes = require('./routes/auth');  
// Import other routes as needed  
  
const app = express();  
  
// Middleware  
app.use(cors());  
app.use(express.json());  
  
// Connect to MongoDB  
mongoose.connect('mongodb://localhost/OLobby', { useNewUrlParser: true, useUnifiedTopology: true });  
  
// Use routes  
app.use('/api/tasks', taskRoutes);  
app.use('/api/auth', authRoutes);  
// Use other routes as needed  
  
const PORT = process.env.PORT || 5000;  
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));  
  
module.exports = app;  
```  
  
`server/routes/tasks.js` (Express route for tasks):  
```javascript  
const express = require('express');  
const router = express.Router();  
const { getTasks, createTask, updateTask, deleteTask } = require('../controllers/taskController');  
  
router.get('/', getTasks);  
router.post('/', createTask);  
router.put('/:id', updateTask);  
router.delete('/:id', deleteTask);  
  
module.exports = router;  
```  
  
This is a very high-level overview and starting point. Each file and folder would need to be fleshed out with the appropriate code to handle the application's logic, UI, and data management. Additionally, you would need to set up authentication, authorization, database schemas, API endpoints, and integrate third-party services for features like video conferencing (e.g., using WebRTC or a service like Twilio).