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 high-level folder structure and some initial boilerplate code for a full-stack application that could be used as a starting point for OLobby.  
  
Assuming we are using a JavaScript stack with Node.js for the backend, React for the frontend, and a MongoDB database, the folder structure might look something like this:  
  
```  
OLobby/  
|-- backend/  
| |-- src/  
| | |-- controllers/  
| | | |-- taskController.js  
| | | |-- userController.js  
| | | |-- chatController.js  
| | | |-- videoConferenceController.js  
| | | `-- documentController.js  
| | |-- models/  
| | | |-- taskModel.js  
| | | |-- userModel.js  
| | | |-- chatModel.js  
| | | |-- videoConferenceModel.js  
| | | `-- documentModel.js  
| | |-- routes/  
| | | |-- taskRoutes.js  
| | | |-- userRoutes.js  
| | | |-- chatRoutes.js  
| | | |-- videoConferenceRoutes.js  
| | | `-- documentRoutes.js  
| | |-- middleware/  
| | | `-- authMiddleware.js  
| | |-- utils/  
| | `-- app.js  
| `-- package.json  
|-- frontend/  
| |-- public/  
| | `-- index.html  
| |-- src/  
| | |-- components/  
| | | |-- Dashboard/  
| | | | `-- GlobalDashboard.js  
| | | |-- TaskManagement/  
| | | | `-- TaskList.js  
| | | |-- DocumentManagement/  
| | | | `-- DocumentList.js  
| | | |-- Workflow/  
| | | | `-- WorkflowBoard.js  
| | | |-- Chat/  
| | | | `-- ChatWindow.js  
| | | |-- VideoConference/  
| | | | `-- VideoConferenceRoom.js  
| | | `-- Shared/  
| | | `-- FileUploader.js  
| | |-- App.js  
| | |-- index.js  
| | `-- styles/  
| | `-- main.css  
| `-- package.json  
|-- .gitignore  
|-- README.md  
`-- package.json  
```  
  
Here's a brief explanation of the folder structure:  
  
- `backend/`: Contains all the server-side code.  
 - `src/`: The source directory for the backend.  
 - `controllers/`: Functions that handle requests and responses.  
 - `models/`: Schemas and models for database entities.  
 - `routes/`: Route definitions for the API endpoints.  
 - `middleware/`: Middleware functions for things like authentication.  
 - `utils/`: Helper functions and utilities.  
 - `app.js`: The main entry point for the backend server.  
 - `package.json`: Defines backend dependencies and scripts.  
  
- `frontend/`: Contains all the client-side code.  
 - `public/`: Public assets like the main HTML file.  
 - `src/`: The source directory for the frontend.  
 - `components/`: React components for each feature of the application.  
 - `App.js`: The main React component that assembles the application.  
 - `index.js`: The entry point for the React application.  
 - `styles/`: CSS files for styling the application.  
 - `package.json`: Defines frontend dependencies and scripts.  
  
- `.gitignore`: Lists files and folders to be ignored by git.  
- `README.md`: A markdown file containing information about the project.  
- `package.json`: (At the root) May define scripts to run both frontend and backend together, for example using `concurrently`.  
  
This is a very high-level structure and doesn't include all the files you would need, but it should give you a good starting point. Each file would contain boilerplate code to get started with the development. For example, `app.js` in the backend might set up an Express server and connect to MongoDB, while `App.js` in the frontend would set up the main React component with routing.  
  
Remember that this is just a starting point, and as the project grows, you might need to refactor the structure, add more components, services, and utilities, and possibly introduce state management libraries (like Redux), testing frameworks, and other tools as needed.