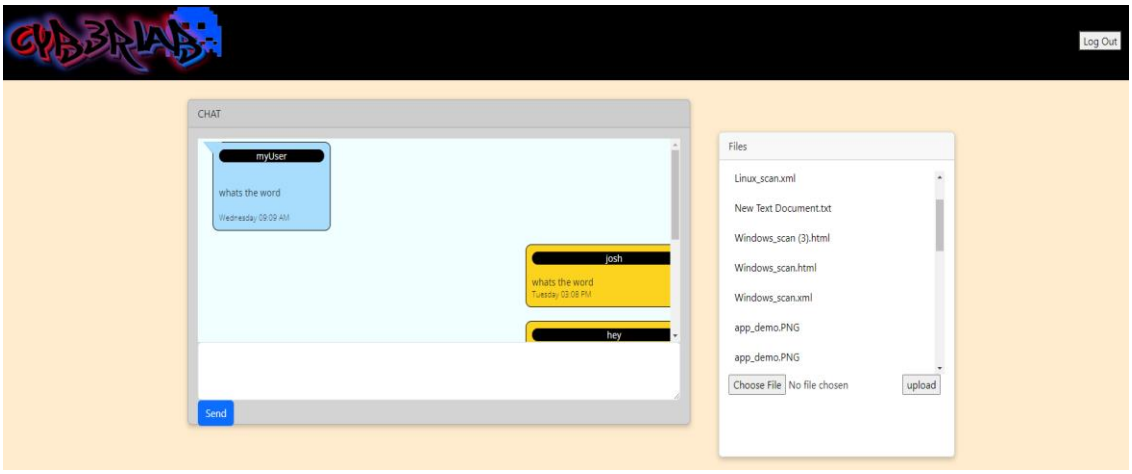
File Share

This file sharing application was built to providing an easy to deploy web app that can run on local devices in a network. This application will be used to quickly allow users to discuss and share files about a specific topic in a local environment. All data stored and API calls made are stored locally and thus information remains private to users with only the host and admin able to tear down the server and mount the shared files for backups. Upcoming additional features such as user login, admin administration, docker deployment paths will be added for future functionality.



Front End

The Application consist of two main components, a lightweight angular frontend built using the angular/CLI NPM modules, and the backend code used for storing and retrieving messages, as well as hosting the links to download files from the server. Figure 1 below describes the overall approach and system components used to build the front-end application.

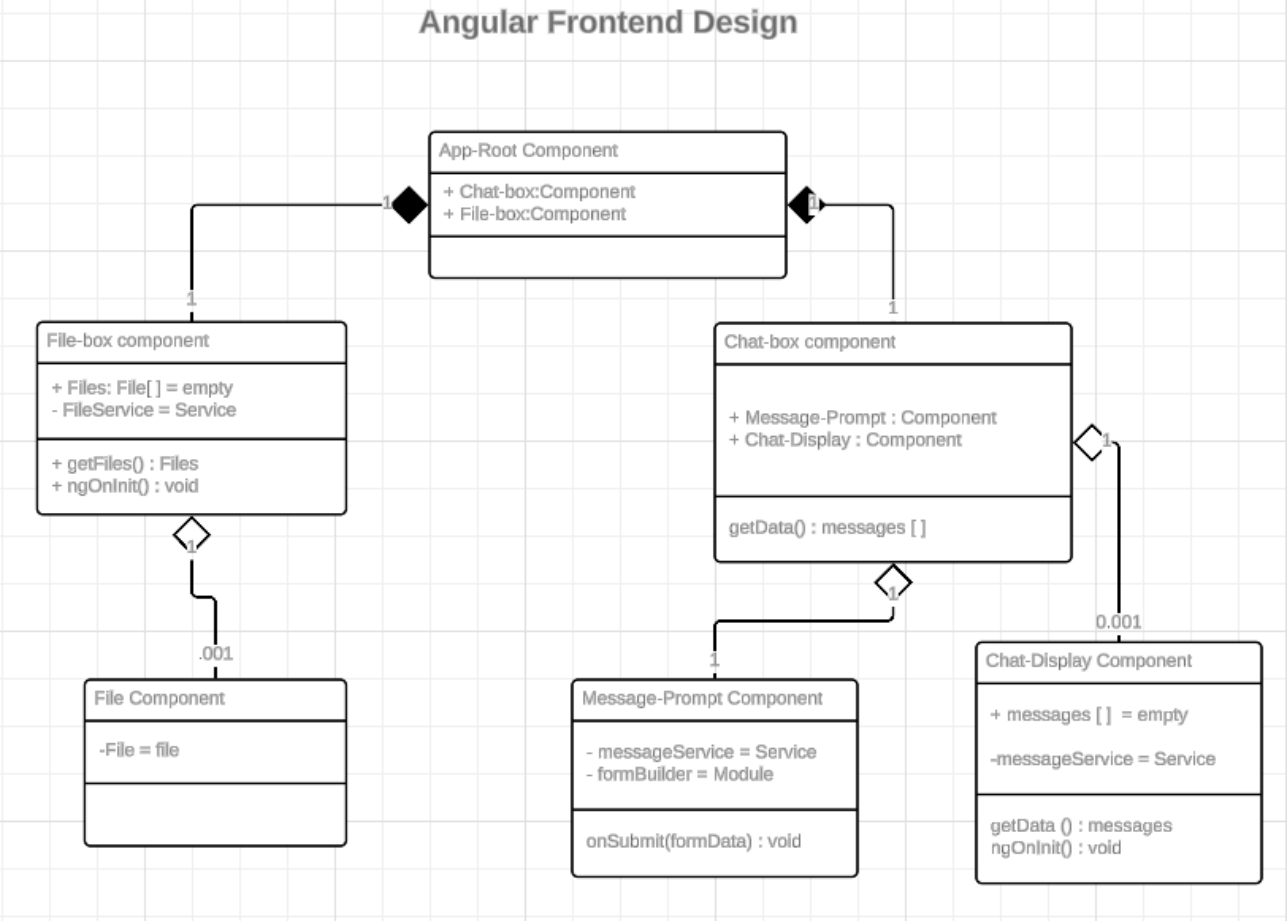


Figure 1: UML Diagram

Data is retrieved from the local database via the message service and file service module which are custom built api endpoints that interact with the NodeJS backend server running on local host via RESTful APIS.

Backend API Structure

The backend server is running an express backend using CORS (Cross-Origin Resource Sharing) to listen to the front-end applications Api request for message data and file request. File uploads are handles via the Post: '/upload/' API where the file is salted, saved, and a database document is created to reference the location users can download the file from as shown in figure 2.

```
app.post('/upload', async (req, res, next) => {
  try {
    //Retrieve File data from request body
    const file = req.files.mFile
    console.log(file)

    //create unique file identifier based on time
    const fileName = new Date().getTime().toString() + path.extname(file.name)

    //define where and how to save the file
    const savePath = path.join(__dirname, 'public', 'ftp', fileName)
    if (file.truncated) {
      throw new Error('File size is too big...')
    }

    //save file
    await file.mv(savePath)

    //create new file database object saving file location and url download link to retrieve files
    const newFile = await Model.create({
                name: file.name,
                url: fileName,
                link: "http://localhost:3000/files/download?name=" + fileName
      })
      .then((data) => {
      res.redirect("http://localhost:4200")
      })
  } catch (error) {
    console.log(error)
    res.send('Error uploading file')
  }
})
```

Figure 2: upload API

Users are able to download specific files from the application via the file object referenced in the database, Links and file names are custom made during runtime and there are mitigations against hash collisions and with unique file identification. Downloading files are as simple as making an http get request with the unique identifier of the file to the '/download' Api endpoint which is handled by the frontend application with its file data references.

```
const downloadFile = async (req, res) => {
    //get File name from request body
    let file_name = req.query.name

    //Define path to file using the file name and the public folder
    const downloadPath = path.join(__dirname, '..', '..','public', 'ftp',file_name);

    //serve the file
    res.download(downloadPath, function (error) {
        console.log("Error : ", error)
    });
}
```

Figure 3: file download Api

DATABASE

The File Share application uses the mongoose MongoDB driver for express to handle two collections for user messages and file data being used. Upon user submission user messages are built in the NodeJS backend and sent to the database for persistent data storage along with additional information such as the author and time stamp of each message shown in figure 4. The file collection stores the name of each file uploaded to the server, the URL link for users to download the file, and the original name of the file before salting for hash collisions which is shown in figure 5.

```
_id: ObjectId('66993c915ca0c54dccd88d49')
author : "No Author"
time : 2024-07-18T16:02:25.579+00:00
message : "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmo…"
__v : 0
```

Figure 4: User-message data

```
_id: ObjectId('66982ded16ea4a073c00bdc7')
name : "Linux_scan (1).html"
url : "1721249261371.html"
link : "http://localhost:3000/files/download?name=1721249261371.html"
__v : 0
```

Figure 5: File-storage data

Additional Information

Backend API endpoints:

Method	API	Description	Response
GET	“/api/messages”	Get messages from DB	Array of Json messages
POST	“/api/messages”	Add message to DB	None
GET	“/api/oldMessages”	Get messages from DB	Array of Json messages
GET	“/files”	Get Files from DB	Array of Json file references
GET	“/files/download”	Download file from server	File
POST	“/Upload”	Upload file to server	Home redirect