Ernst-Mach-Gymnasium Haar

Ausbildungsabschnitt 11/2

Fileplay

Dateiübertragungswebseite mit Kontaktfunktion

Quentin Frey, Leonhard Masche, Dezhong Zhuang

Inhaltsverzeichnis

1 Einleitung **2**

2 Dokumentation **2**
2.1 Implementierung 2
2.2 Architektur 3
2.3 Authentifizierung 3
2.4 Geräte, Nutzer, Kontakte 4
2.5 Filesharing 4
2.6 User-Story 4

3 Aufteilung der Aufgaben **6**
3.1 Leonhard Masche (@leonhma) 6
3.2 Quentin Frey (@limosine) 8
3.3 Dezhong Zhuang (@didi1150) 9

1 Einleitung

Unser Projekt ist eine Filesharing App (Webseite), mit der Dateien Peer-to-Peer, also von Nutzer zu Nutzer, ohne zwischengeschalteten Server, übertragen werden können. Die App basiert auf modernen Web-Technologien, wie z.B. Web Push, Service Workers und Progressive Web Apps. Durch die Design-Philosophie des Progressive Enhancement werden aber auch ältere Browser unterstützt. Eine Besonderheit der App ist es, dass sie dauerhafte Verbindungen durch Kontakte möglich macht. Kontakte können zwischen mehreren Nutzern geknüpft werden, wobei jeder Nutzer mehrere Geräte haben kann. Zum Schutz der Privatsphäre und Benutzersicherheit werden keine identifizierbaren Informationen auf dem Server (in der Datenbank) gespeichert und die Dateien im Transit Ende-zu-Ende verschlüsselt. Die Motivation für dieses Projekt war das Problem, dass es keine einfache Möglichkeit gibt, zwischen iOS / Android / Windows / Linux Geräten Dateien zu teilen. Bestehende Webseiten nutzen hierfür einen Link, den der Empfänger öffnen muss, um die Datei zu empfangen. Hierbei stellt sich dann wieder die Frage, wie man diesen Link teilt. Um dieses Problem zu behandeln, implementiert die App ein Kontakt-System, in dem sich Nutzer einmalig über einen 6-stelligen Code verbinden und von nun an einfach miteinander Dateien teilen können. Dieses System ist vor allem in einer Umgebung nützlich, in der man oft Dokumente mit denselben Personen teilt, wie es z.B. in der Schule der Fall ist.

2 Dokumentation

2.1 Implementierung

Die App wurde in Form einer Webseite realisiert und ist somit auf allen Geräten zugänglich, die einen Browser haben. Für die Entwicklung haben wir uns für das Framework SvelteKit im NodeJS Ökosystem entschieden. Dadurch muss die App nicht manuell in den Browser-kompatiblen Sprachen geschrieben werden, sondern kann in einer modernen Sprache (Svelte in Verbindung mit Typescript) geschrieben werden, wodurch die Entwicklung durch vorgefertigte Implementierungsmuster und Tools stark vereinfacht wird. Die App ist nach dem Muster einer Single-Page-Application aufgebaut, d.h. die Seite wird nur einmal geladen, und alle weiteren Interaktionen finden über JavaScript statt. Dadurch wird die App sehr schnell, da nur die Daten, die benötigt werden, vom Server geladen werden müssen. Zusätzlich wird die App dadurch sehr benutzerfreundlich, da der Nutzer nicht durch das Laden von verschiedenen Seiten unterbrochen wird. Die Dateiübertragung selbst findet über die Bibliothek PeerJS statt, welche wiederum WbRTC benutzt. Durch die optionale Installation als Progressive Web App (PWA) kann die App auch auf dem Homescreen installiert werden, und verhält sich dann wie eine native App.
Die Entwicklung der App findet in einer öffentlichen GitHub Repository statt. Dieses Repository ist unter `https://github.com/leonhma/fileplay` zu finden, und wird automatisch durch Integration mit Cloudflare auf den Adressen `https://app.fileplay.me` (für den main-branch) und `https://dev.fileplay.pages.dev` (für den dev-branch) zur Verfügung gestellt.
Der Programmcode für einen Microservice der die Kommunikation von Server

zu Client verwaltet ist unter
`https://github.com/leonhma/fileplay-worker` zu finden. Dieser Mikroservice nutzt Cloudflare APIs die nicht in SvelteKit verfügbar sind, und wird deshalb separat gehosted.

2.2 Architektur

Die App ist nach dem MVC-Modell in drei Teile aufgeteilt: Frontend, Backend und Datenbank. Das Frontend (View) ist die Webseite, die der Nutzer sieht und mit der er interagiert. Das Backend (Controller) ist der Server, der die Webseite ausliefert und die Anwendungslogik verwaltet, und die Datenbank ist das Modell, in dem die Daten gespeichert werden. Sowohl das Frontend, als auch große Teile des Backends werden vom Framework SvelteKit verwaltet. Dieses Framework erlaubt es, die Webseite mit einer angepassten Syntax zu schreiben und Entwickler-freundlich in Komponenten zu zerteilen. Zusätzlich bietet es eine einfache Möglichkeit, Backend und Frontend Seite an Seite zu entwickeln. Zusätzlich zu dem Backend-Code, der in SvelteKit geschrieben ist, gibt es noch einen Microservice, der als Cloudflare Worker in Typescript geschrieben ist. Dieser Verwaltet die WebSockets, die für die Kommunikation von Server zu Client verwendet werden. Die Datenbank ist eine SQLite Datenbank, welche mit dem Backend verbunden ist und Daten wie Geräte, Nutzer oder Kontakte speichert. Die einzelnen Teile der App (Backend, Datenbank und Microservice) werden alle auf Cloudflare gehosted. Cloudflare bietet verschieden Web-Hosting-Services an, darunter auch Pages, welches über eine GitHub Integration einfach mit dem Build-Output von SvelteKit verbunden werden kann. Zusätzlich wird eine SQLite Datenbank namens D1 verwendet, welche im Backend einfach durch Umgebungsvariablen eingebunden werden kann. Der Microservice ist ein Cloudflare Worker, welcher mithilfe der neuen WebSocket Hibernation API ressourcenschonend die Verbindungen von Server zu Client verwaltet.
Die clientseitige Logik der App findet vor allem in Form von JavaScript clientseitig statt. Wenn Daten vom Server abgefragt werden müssen, wie zum Beispiel die Liste der Kontakte, wird dies über eine REST API gemacht. Dagegen findet die Kommunikation von Server zu Client über WebSockets statt. Alternativ wird hier auch eine Implementierung mit Web Push verwendet, jedoch wurde diese aufgrund von Kompatibilitätsproblemen mit Edge und Firefox kurzfristig deaktiviert. Die Dateiübertragung zwischen zwei Clients findet über WebRTC statt. So kann die Datei (bis auf die Ausnahme einer Firewall) direkt zwischen zwei Clients übertragen werden. Zwischen dem Front- und Backend steht noch ein Service Worker, der im Browser der Nutzers läuft. Dieser ist für das Empfangen von Nachrichten des Servers, und das cachen von Dateien zuständig.

2.3 Authentifizierung

Beim ersten Besuch der Webseite wird der Nutzer aufgefordert, bei der Einrichtung einen Name für das Gerät und den verbundenen Nutzer einzugeben. Diese Informationen werden dann an den Server gesendet, welcher dem Gerät eine einzigartige Device ID zuweist. Diese ID wird dann in einem Cookie gespeichert, und bei weiteren Besuchen der Webseite an den Server gesendet. Der Server kann dann durch diese ID das Gerät identifizieren, und die Daten des Geräts aus der Datenbank laden.

Um zu verhindern, dass ein Nutzer sich als ein anderes Gerät ausgibt, wird zusätzlich zur Device ID ein Token generiert, welches auch einem Cookie gespeichert wird. Dieses Token wird bei jeder Anfrage an den Server gesendet, und der Server kann dann überprüfen, ob das Token, das mit der Signatur des Servers unterzeichnet wurde, zu der Device ID passt.

2.4 Geräte, Nutzer, Kontakte

Um eine einfache Organisation und Nutzung der App zu ermöglichen, wird zwischen Geräten, Nutzern und Kontakten unterschieden. Ein Gerät ist ein Endgerät, auf dem die App installiert ist. Ein Nutzer ist eine Person, die die App nutzt. Ein Nutzer kann mehrere Geräte besitzen, ein Gerät dagegen kann nur einem Nutzer zugeordnet werden. Ein Kontakt repräsentiert eine Verbindung zwischen zwei Nutzern, welche wiederum unbegrenzt viele Kontakte haben können.
Diese Verknüpfungen von Nutzern und Geräten (one-to-many) und Nutzern untereinander, als Kontakte (many-to-many), können durch das Generieren und Einlösen von sechsstelligen Verbindungscodes realisiert werden. Wenn ein Nutzer einen Code generiert, wird dieser Code in der Datenbank gespeichert und kann diesen Code an einen anderen Nutzer weitergeben. Wenn der andere Nutzer diesen Code eingibt, wird eine Verbindung zwischen den beiden Nutzern hergestellt und der Code wird aus der Datenbank gelöscht.

2.5 Filesharing

Wenn ein Nutzer eine Datei teilen möchte, wählt er einen verfügbaren Kontakt aus, dessen ID dann in einer Anfrage an den Server gesendet wird. Dieser schickt diese Anfrage anschließend per WebSocket (oder Web Push) an die einzelnen Geräte des Kontakts. Wenn ein Gerät die Anfrage akzeptiert, schickt dieses Gerät eine Antwort mit der PeerJS-Adresse des Geräts, an das die Datei übertragen werden soll, an den Server. Diese Antwort enthält auch den Public Key des Geräts, der für die Verschlüsselung der Datei verwendet wird. Der Server leitet diese Antwort dann an den ursprünglichen Nutzer weiter, der die Datei dann an die PeerJS-Adresse senden kann.

2.6 User-Story

2.6.1 Einrichtung

Wenn ein Nutzer die App zum ersten Mal öffnet, wird er aufgefordert, einen Namen für das Gerät anzugeben, und den Typ des Geräts auszuwählen. Dies dient lediglich der erleichterten späteren Identifikation der Geräte durch den Nutzer selbst.
Zusätzlich zu den Daten des Geräts muss der Nutzer entweder einen neuen Nutzer erstellen, oder das Gerät zu einem Bereits existierenden hinzufügen. Wenn der Nutzer einen neuen erstellt, Kann er diesem einen Namen geben. Dieser Name wird auch für Kontakte sichtbar sein, wenn mit ihnen eine Datei geteilt wird. Dementsprechend wird der Name mithilfe der kostenlosen API `https://www.purgomalum.com/` auf unangemessene Inhalte überprüft.
Ein weiteres identifizierendes Merkmal des Nutzers ist ein Avatar. Um auch hier

ungewollte Überraschungen zu vermeiden, wird lediglich ein Seed generiert, der dann durch die Dicebear API in einen Avatar umgewandelt wird. Wenn der Nutzer das Gerät zu einem bereits existierenden Nutzer hinzufügt, muss er den Verbindungscode eingeben, den der Nutzer generiert hat. Dieser Code wird dann an den Server gesendet, der dann überprüft, ob der Code existiert. Wenn der Code existiert, wird das Gerät dem Nutzer hinzugefügt, und der Code wird aus der Datenbank gelöscht. Wenn der Code nicht existiert, wird eine Fehlermeldung angezeigt.
Nach der erfolgreichen Einrichtung wird sich die Statusanzeige (in der oberen rechten Ecke) schnell auf grün ändern. Das bedeutet, dass das Gerät mit dem Server verbunden ist, und die App vollständig einsatzbereit ist.

2.6.2 Kontakte verwalten

Zum hinzufügen und entfernen von Kontakte muss sich der Nutzer zuerst die Einrichtung durchlaufen. Dann kann er in der Top-Bar (der orangen Leiste oben im Fenster) auf das Icon mit dem Kontaktbuch klicken. Daraufhin öffnet sich ein Drawer (eine Leiste, die von der Seite aus aufgezogen wird), in dem alle Kontakte aufgelistet sind. Wenn der Nutzer einen Kontakt hinzufügen möchte, kann er auf 'add contact' klicken. Daraufhin öffnet sich ein Dialog, in dem der Nutzer entweder einen bestehenden Verbindungscode eingeben, oder einen neuen generieren kann. Wenn der Nutzer einen neuen Code generiert, wird dieser Code in der Datenbank gespeichert, und der Nutzer kann diesen Code an einen anderen Nutzer weitergeben.
Dieser Andere Nutzer kann dann in demselben Dialog auf seinem Gerät den Code eingeben, und verbindet sich somit als Kontakt der Person, die den Code generiert hat. Diese Verbindung ist beidseitiger Natur, d.h. beide Nutzer sind Kontakte des jeweils anderen.
Möchte der Nutzer einen Kontakt entfernen, kann er in der Liste der Kontakte auf das Trash-Icon neben dem Kontakt klicken. Daraufhin wird eine Anfrage an den Server gesendet, der dann den Kontakt aus der Datenbank löscht. Diese Anfrage bedarf natürlich - so wie die meisten - Authentifizierung damit nicht jeder Nutzer die Kontakte anderer Nutzer löschen kann.

2.6.3 Dateien teilen

Um eine Datei zu teilen, muss der Nutzer zuerst die Einrichtung durchlaufen. Dann kann er mitten auf der Seite das Icon mit dem Upload-Symbol klicken. Daraufhin wird vom Browser ein Datei-Dialog geöffnet, in dem der Nutzer eine Datei auswählen kann. Wenn der Nutzer eine Datei ausgewählt hat, wird der Name der Datei neben dem Icon angezeigt. Nun kann der Nutzer durch einen Klick auf das 'Senden' Icon die Datei an einen Kontakt senden. Es öffnet sich ein Dialog, in dem der Nutzer einen Kontakt auswählen kann. Hier werden nur Kontakte angezeigt, die in der letzten Minute ein Lebenszeichen an den Server gesendet haben. Wenn der Nutzer einen Kontakt auswählt, wird eine Anfrage an den Server gesendet, der dann die Anfrage an die einzelnen Geräte des Kontakts weiterleitet und auf eine Antwort wartet. Während diesem Prozess steht der Status beim Sender auf 'requesting'. Wenn ein Gerät die Anfrage akzeptiert, wird ein weiterer Request an den Server gesendet, der auch die Details benötigt, um die Datei zu senden. Wenn der Server diese Details erhalten hat,

wird eine Benachrichtigung mit den Details an das Gerät gesendet, welches die ursprüngliche Anfrage gestartet hat. Dieses Gerät kann dann die Datei an das Gerät des Kontakts senden. Während der Übertragung der Datei wird der Status beim Sender auf 'sending' gesetzt. Passiert ein Fehler, wird der Status auf 'failed' gesetzt.

2.6.4 Nutzer und Geräte verwalten

Wenn der Nutzer die Details (Name und Avatar) seines Nutzers ändern möchte, kann er in der Top-Bar auf das Icon mit dem Zahnrad klicken, und zum Tab 'Account' navigieren. Daraufhin öffnet sich ein Dialog, in dem der Nutzer seinen Namen und Avatar ändern kann. Wenn der Nutzer seinen Namen oder Avatar ändert, wird dieser auch für alle Kontakte geändert. Wenn der Nutzer die Namen oder Typen seiner Geräte ändern möchte, so kann er das im Tab 'Devices' tun. Hier werden alle Geräte des Nutzers aufgelistet, und durch einen Klick auf das Dreipunkt-Menü wird ein weiterer Dialog geöffnet, in dem die Details des Geräts angepasst, sowie die Verbindung des Gerätes zum Account gelöst werden können.

2.6.5 App-Updates

Während der Entwicklung, aber auch während der Veröffentlichung der App, ist es nützlich, den Nutzer über Updates zu informieren. Das ist besonders wichtig, da ein Service Worker benutzt wird, der die Webseite offline verfügbar mach, und somit nicht immer die aktuellste Version der Webseite lädt. Um den Nutzer über Updates zu informieren, wird periodisch (30 Minuten) überprüft, ob eine neue Version der App verfügbar ist. Ist das der Fall, wird eine Benachrichtigung angezeigt, die den Nutzer informiert, dass ein Update verfügbar ist. Wenn der Nutzer auf diese Benachrichtigung klickt, wird die Seite neu geladen, und die neue Version wird installiert.
Durch die Integration mit GitHub und Cloudflare wird die App automatisch bei jedem Commit auf GitHub neu gebaut und auf Cloudflare Pages veröffentlicht. Die Entwicklung findet auf dem 'dev'-branch statt, und finalen Versionen werden über einen Pull-Request auf den 'main'-branch übertragen. Wenn ein Pull-Request auf den 'main'-branch gemerged wird, wird die App automatisch auf der Haupt-Domain 'app.fileplay.me' veröffentlicht.

3 Aufteilung der Aufgaben

Inhaltsverzeichnis

3.1 Leonhard Masche (@leonhma) 6
3.2 Quentin Frey (@limosine) 8
3.3 Dezhong Zhuang (@didi1150) 9

3.1 Leonhard Masche (@leonhma)

- Infrastruktur und DevOps: Ich habe die Struktur des Projektes eingerichtet, und die Infrastruktur auf Cloudflare konfiguriert. Dazu gehört die

Konfiguration des Domainnamen und DNS, sowie das Einrichten der Datenbank, deren Integration mit dem Backend, sowie die CI/CD Integration mit GitHub. Implementierungen sind: '/src/lib/db.ts', '/src/lib/server/db.ts', '/wrangler.toml', '/migrations/*'

- API: Ich habe die REST API entwickelt, die z.B. das Auflisten von Kontakten und Geräten, Anfragen von Datei-Sharing und das Verbinden von Nutzern untereinander und mit Geräten ermöglicht. Im Projekt sind all diese Dateien unter '/src/routes/api/**' zu finden.

- Authentifizierung: Ich habe die Authentifizierung implementiert, die dafür sorgt, dass nur authentifizierte Geräte mit dem Server kommunizieren können. Dazu gehört die Generierung von Tokens und Device-IDs, sowie die Überprüfung dieser. Die Implementation ist unter '/src/lib/server/crypto.ts' zu finden.

- Server Side Notifications: Schon beim Thema Backend angekommen, habe ich auch das System für die Benachrichtigungen konzeptioniert und implementiert. Dieses System besteht aus einer Funktion auf dem Server, die eine Nachricht adaptiv über einen der beiden Kanäle (WebSockets oder Web Push) an den Client überliefert, sowie der clientseitige Logik, um die Verbindung für die Nachrichten zu initialisieren und aktiv zu halten. Die Implementation befindet sich unter '/src/lib/server/notifications.ts' udn '/src/sw.ts'

- Web Push: Als einer der beiden Kanäle für die Benachrichtigungen habe ich Web Push implementiert. Dazu gehört auch die Authentifizierung mit dem VAPID-Keypair des Servers. (siehe '/scripts/generate-keys.js'). Dieser Kanal ist vor allem für die Benachrichtigungen auf mobilen Geräten interessant, da er stromsparend Benachrichtigungen über einen Zentralisierten Push-Service liefert, auch wenn das Gerät sich nicht auf der Webseite befindet. Die Implementation findet sich zum Teil in '/src/lib/server/notifications.ts' und zum Teil im Service Worker ('/src/sw.ts'). Aufgrund von unergründbaren Problemen bei der Kompatibilität mit anderen Browsern (nicht Google Chrome), musste dieses Feature (zumindest für die Vorführung) aber deaktiviert werden.

- Service Worker: Zwischen Client und Server habe ich auch den Service Worker geschrieben, der das Caching der Webseite für eine schnellere Ladezeit mithilfe WorkBox verwaltet, und eine wichtige Rolle bei den Push Notifications spielt. Der Service Worker befindet sich unter '/src/sw.ts'.

- WebSockets: Als zweiter Kanal für die Benachrichtigungen habe ich die finale Version der WebSockets implementiert. Zu dieser Benachrichtigungsmethode gehört auch der in 'leonhma/fileplay-worker' aufzufindende Cloudflare Worker, der ein Durable Object verwaltet, welches die WebSocket Verbindung dem Server gegenüber zugänglich macht.

- Als ein weiterer Teil der Notifications, habe ich mich mit der Übertragung von Nachrichten zwischen Service Worker und Client auseinandergesetzt. Dazu gehört auch, die per Web Push empfangenen Nachrichten an den Client (die Webseite) weiterzuleiten, um auf sie reagieren zu können. Die relevanten Stücke Code befinden sich in '/src/sw.ts' und /src/lib/messages.ts'.

- Notifications: Als letzter Teil der Notifications habe ich mich um die Anzeige von interagierbaren Benachrichtigungen gekümmert. Das sind zum einen die Benachrichtigungen die über den Service Worker und Web Push im OS-Tray angezeigt werden, aber auch die Nachrichten, die in der App im Notification Drawer sichtbar sind. Bei den In-App Benachrichtigungen habe ich mich um die klickbaren Aktionen und deren Ausführung bemüht, welche stark die Notification API im Service Worker nachahmen. Die relevanten Dateien sind '/src/sw.ts', '/src/lib/stores/Dialog.ts' und '/src/lib/components/NotificationDrawer.svelte'.

- Ein Teil des UIs den Ich gestaltet habe, ist der Einrichtungs-Dialog. Dieser ist eng Verbunden mit der Authentifizierung und Der REST API. Die relevante Datei ist '/src/lib/dialogs/SetupDialog.svelte'.

- Progressive Web App: Die App kann durch das hinzufügen eines Webmanifest und des Service Workers als Progressive Web App installiert werden und als native App verwendet werden. Ich habe sowohl den Service Worker, als auch das Webmanifest geschrieben ('/src/static/manifest.json').

3.2 Quentin Frey (@limosine)

- UI (Material Design):
  Viele Webseiten, Progressive Web Apps oder auch native Apps verwenden Material Design. Um die Eingewöhnung so einfach wie möglich zu machen, hielt ich mich streng an die Material Design Guidelines. Daher verwendete ich ausschließlich Komponenten aus SMUI (SvelteMaterialUI), zudem blieb der Code dadurch übersichtlich. Die UI der App besteht aus der Startseite, dem Benachrichtigungsdrawer, dem Kontaktdrawer und einer Leiste, welche sich am oberen Fensterrand befindet. Außerdem können einige Dialoge, wie der Einstellungsdialog, der Dialog für das Senden von Dateien oder der Dialog, welcher es ermöglicht die Eigenschaften von Geräten zu verändern, geöffnet werden. In der oberen App-Leiste befinden sich Knöpfe, mit welchen die Drawer geöffnet werden können, der aktuelle Status angezeigt wird und der Einstellungsdialog geöffnet werden kann.

- State Management:
  Da die angezeigten Daten immer auf dem aktuellsten Stand sein müssen, wurde ein zentraler Timer implementiert, welcher die Daten in einem bestimmten Intervall abruft, wenn diese benötigt werden. Damit die Daten nicht verloren gehen und die Seite dynamisch erneuert wird, werden Svelte Stores verwendet, in denen die oben genannten Daten gespeichert werden.

- Senden & Aufteilen von Dateien (PeerJS):
  Die Dateien werden in 1 Megabyte große Stücke (Chunks) aufgeteilt und mithilfe von PeerJS, welches die Nutzung von WebRTC-Protokollen vereinfacht, versendet. Dies ist im Ordner *src/lib/peerjs* zu finden.

- Verschlüsselung & Entschlüsselung von Dateien (OpenPGP):
  Da die OpenPGP-Verschlüsselung schon im Alltag verwendet wird, ist diese vielfach überprüft worden und daher sehr sicher. Beispielsweise beim Versenden von E-Mails wird oftmals die Verschlüsselung durch GPG / OpenPGP durchgeführt. Bei diesem Verfahren wird ein öffentlicher und

ein privater Schlüssel erstellt. Um eine Nachricht zu verschlüsseln, wird der öffentliche Schüssel des Empfängers benötigt, sodass der Empfänger die Datei mit seinem persönlichen privaten Schlüssel entschlüsseln kann. Diese Nachrichten werden durch eine Armor verstärkt, indem diese ins ASCII-Encoding konvertiert werden, anschließend werden diese an den Empfänger gesendet, welcher die Stücke dann wieder zusammensetzt und entschlüsselt.

- WebSockets (ohne Durable Objects):
  Durable Objects sind Objekte, welche im Cloudflare-Netzwerk in jedem Rechenzentrum existieren und gegenseitig miteinander vernetzt sind. Durch diese Objekte ist es möglich, mehrere WebSocket-Verbindungen zu koppeln, sodass Daten ohne Verzögerung übertragen werden können. Da Durable Objects Premium-Funktionen von Cloudflare sind, implementierte ich eine WebSocket-Verbindung, welche nicht auf Durable Objects oder ähnliches angewiesen ist. Diese wurde aber von dem Fileplay-Worker ersetzt, sodass die Webseite, nachdem das Abonnement gekündigt wurde, nicht mehr funktioniert und voraussichtlich durch die ältere Methode ersetzt werden muss. Diese ältere Methode verwendete Einträge in der Datenbank, um die PeerJS-ID sowie den öffentlichen Schlüssel zu übertragen und funktioniert daher auch, wenn die WebSockets keine gegenseitige Kenntnis voneinander haben.

- Gast-Modus (ohne Erstellung eines Accounts):
  Da viele Personen lediglich ein einziges Mal eine Datei empfangen möchten, ist es für diese umständlich einen Account erstellen zu müssen. Daher fügte ich den Gast-Modus hinzu, welcher es ermöglicht einen Link zu generieren, welchen jeder aufrufen kann, ohne einen Account zu erstellen. Auch bei diesem Modus wird die Datei verschlüsselt, statt einem PublicKey wird hier jedoch einen Zeichenfolge verwendet, welche im Link enthalten ist.

3.3 Dezhong Zhuang (@didi1150)

- BETA UI:
  Für das Design einer Website/App muss zuerst immer ein Design entworfen werden, welches als Vorlage für die zukünftige Entwicklung dient. Dazu wurde explizit pures HTML, CSS und Javascript verwendet, um den Designprozess so schnell und effizient wie möglich zu gestalten. Zudem führt das Fehlen von jeglichen Abhängigkeiten dazu, dass später Features schneller implementiert werden können.

- Filesharing Progress UI:
  Der Zweck des Aufteilens der Dateien in Chunks war, dass somit der Empfänger Rückmeldung bekommt, wie weit der Dateiempfang schon vorangeschritten ist. Hierbei werden die angekommenen Chunks innerhalb eines Svelte Stores gespeichert (mit jeweils unterschiedlichen IDs zur Zuordnung) dessen Daten dann mit der Seite geteilt werden.

- State Management:
  Für das Suchen oder auch Speichern von ausgewählten Elementen werden Svelte Stores verwendet, um automatische interne Benachrichtigungen beim Ändern eines Wertes zu verschicken. Für den Fall, dass keine

- Senden & Aufteilen von Dateien (PeerJS):
  Der Sender und Empfänger verschicken jeweils Metadaten mithilfe Peerjs aneinander, wodurch ein Transferprozess initialisiert wird. Während dieses Prozess wird die angefragte Datei in 1 Megabyte große Stücke (Chunks) aufgeteilt und an den Empfänger gesendet, Während dieser nach jedem Chunk eine Bestätigung sendet.

- Recherche und Vorstrukturierung:
  Für die Verwendung von neuen Ressourcen muss oft vorrecherchiert werden, welche APIs oder SDKs nutzbar sind oder den Anforderungen des Projekts entsprechen. Im Anschluss daran werden diese vorübergehend in ihrer gewünschten Funktion zum Testen implementiert, damit diese später optimiert werden können.

- Generierung eines QR-Codes:
  Während des Testens mit mehr oder weniger freiwilligen Probanden wurde festgestellt, dass es sehr umständlich ist, einen Link an andere Personen ohne Verwendung von externen Plattformen zu verschicken. Aus diesem Grund wurde die Anzeige eines generierten QR-Codes implementiert, welcher einfach und leicht aufzurufen ist.