SilentTalk FYP
Partnership Collaboration Guide

Team Success Roadmap

Yasser - Backend & Infrastructure Lead
Zainab - Frontend & UI/UX Lead

Final Year Project
January - July 2026

November 24, 2025

Contents

1 Welcome & Introduction

1.1 Purpose of This Guide

This comprehensive collaboration guide is designed to ensure seamless teamwork between Yasser (backend/infrastructure) and Zainab (frontend/UI) throughout the SilentTalk FYP project. The guide provides:

- **Complete setup instructions** for Zainab to get started
- **Accurate project status** based on current codebase analysis
- **Clear task assignments** with realistic time estimates
- **Week-by-week roadmap** from January to July 2026
- **Technical references** and troubleshooting guides
- **Collaboration workflows** for efficient teamwork

1.2 Project Vision

"Breaking down communication barriers and creating an inclusive environment for the deaf and hard-of-hearing community through advanced technology."

SilentTalk is a comprehensive sign language communication platform that combines:

- Real-time sign language recognition using machine learning
- Accessible video conferencing with live captions
- Community features (forums, resources, glossary)
- Professional interpreter booking services

1.3 Team Roles & Responsibilities

codebg **Role**	**Responsibilities**
Yasser	Backend API development, Database design, Infrastructure setup, Docker orchestration, SignalR implementation, ML service integration, Security, Deployment
Zainab	Frontend development, UI/UX design, Component library, React integration, Accessibility implementation, User testing, Documentation
Shared	Code reviews, Weekly planning, Testing, Documentation, Timeline management, Quality assurance

1.4 Timeline Overview

7-Month Timeline

January 2026: Setup & Planning
February: Core authentication & video calling
March: ML integration & caption system
April: User management & contacts
May: Community features
June: Testing & polish
July: Final testing & submission

2 Setup Guide for Zainab

2.1 Prerequisites

Before starting, ensure you have the following installed on your machine:

1. **Git** (version 2.30+)

```
# Verify installation
git --version
```

2. **Docker Desktop** (latest version)

```
# Verify installation
docker --version
docker-compose --version
```

3. **Node.js** (v20.x LTS)

```
# Verify installation
node --version  # Should be v20.x
npm --version   # Should be 10.x
```

4. **Visual Studio Code** (recommended IDE)

5. **Terminal/Command Prompt**

2.2 Step-by-Step Setup

2.2.1 Step 1: Clone Repository

```
# Clone the repository
git clone https://github.com/hamdanyasser/SilentTalkFYP.git

# Navigate to project directory
cd SilentTalkFYP

```

```
# Check current branch
git branch
# You should be on: main
```

2.2.2 Step 2: Install Frontend Dependencies

```
# Navigate to client directory
cd client

# Install Node.js dependencies
npm install

# This will install:
# - React 18
# - TypeScript
# - Vite
# - All required libraries
```

Note: The project uses `npm install` (not `npm ci`) because there's no `package-lock.json`.

2.2.3 Step 3: Start Docker Services

```
# Go back to project root
cd ~/SilentTalkFYP

# Start all services with one command
./start.sh

# Wait ~60 seconds for services to initialize
# You'll see status updates in the terminal
```

This script starts:

- PostgreSQL (database)
- MongoDB (messages/logs)
- Redis (caching)
- MinIO (file storage)
- Backend API (ASP.NET Core)
- ML Service (FastAPI)
- Frontend (React + Vite)

2.2.4 Step 4: Verify Services Are Running

```
# Check service status
docker ps

# You should see 7-8 containers running:
# - silents-talk-postgres
# - silents-talk-mongodb
# - silents-talk-redis
# - silents-talk-minio
# - silents-talk-server
# - silents-talk-ml
# - silents-talk-client
```

2.2.5 Step 5: Access the Application

Open your browser and navigate to:

codebg **Service**	**URL**
Frontend (your work!)	`http://localhost:3000`
Backend API Docs	`http://localhost:5000/docs`
ML Service API	`http://localhost:8000/docs`
MinIO Console	`http://localhost:9001`

2.3 VS Code Setup

2.3.1 Recommended Extensions

Install these VS Code extensions for optimal development experience:

1. **ESLint** - JavaScript/TypeScript linting
2. **Prettier** - Code formatting
3. **ES7+ React Snippets** - React code snippets
4. **Auto Rename Tag** - HTML/JSX tag renaming
5. **GitLens** - Git integration
6. **Thunder Client** - API testing (alternative to Postman)
7. **Error Lens** - Inline error display
8. **Path Intellisense** - File path autocomplete

2.3.2 VS Code Settings

Create `.vscode/settings.json` in project root:

```json
{
  "editor.formatOnSave": true,
  "editor.defaultFormatter": "esbenp.prettier-vscode",
  "editor.codeActionsOnSave": {
    "source.fixAll.eslint": true
  },
  "typescript.tsdk": "node_modules/typescript/lib",
  "files.exclude": {
    "**/node_modules": true,
    "**/dist": true
  }
}
```

2.4 Troubleshooting Common Issues

2.4.1 Docker Won't Start

Problem: Docker containers fail to start

Solution:

```bash
# Stop all containers
cd ~/SilentTalkFYP
./stop.sh

# Clean Docker system
docker system prune -f

# Restart Docker Desktop
# Then try starting again
./start.sh
```

2.4.2 Port Already in Use

Problem: Port 3000/5000/8000 already in use

Solution:

```
# Find what's using the port (Linux/Mac)
lsof -i :3000
lsof -i :5000

# Kill the process
kill -9 <PID>

# Or change ports in docker-compose.yml if needed
```

2.4.3 Frontend Won't Build

Problem: npm install fails or build errors

Solution:

```
cd ~/SilentTalkFYP/client

# Clear cache and reinstall
rm -rf node_modules
rm package-lock.json
npm cache clean --force
npm install

# If still issues, check Node version
node --version  # Must be v20.x
```

2.5 Verify Setup Checklist

Complete this checklist to ensure everything is working:

- ☐ Git repository cloned successfully
- ☐ Docker Desktop is running
- ☐ All 7-8 Docker containers are running (check with `docker ps`)
- ☐ Frontend accessible at `http://localhost:3000`
- ☐ Backend API docs at `http://localhost:5000/docs`
- ☐ Can register a test user successfully
- ☐ Can login with test user

☐ VS Code opens project without errors

☐ Can run `npm run dev` in client directory

Once all items are checked, you're ready to start development!

3 Project Status (Current State)

3.1 Overall Completion: ~20%

Based on thorough codebase analysis, here's the accurate current status:

codebg **Component**	**Status**	**Notes**
Infrastructure	**95%**	Docker, databases, all services running
Database	**90%**	PostgreSQL + MongoDB configured
Backend Auth	**60%**	JWT working, email verification pending
Backend Calls	**80%**	Full call management + SignalR hub
ML Service	**30%**	Demo mode, no trained model
Frontend Auth	**70%**	Login/register pages working
Frontend Video	**40%**	UI exists, WebRTC not connected
Frontend Other	**15%**	Basic pages, needs backend integration
Testing	**10%**	Minimal tests exist

3.2 What's Working RIGHT NOW

Fully Functional Features

1. **Infrastructure** - All Docker services running, data persists
2. **Authentication** - Users can register/login, JWT tokens work
3. **Call Management Backend** - Full CRUD for calls, scheduling, history
4. **SignalR Hub** - Complete WebRTC signaling (601 lines of code!)
5. **Recording Upload** - Can upload/download call recordings to MinIO
6. **Admin Panel** - User management, statistics, audit logs
7. **Frontend Login/Register** - Forms work, connected to backend
8. **Video Call Page** - UI exists with caption overlay
9. **ML Service Connection** - WebSocket streaming ready (demo mode)
10. **Design System** - Button, Input, Modal components ready

3.3 What's Missing (Needs Implementation)

Critical Missing Features

1. **Trained ML Model** - Currently using mock predictions
2. **WebRTC Video Integration** - Backend ready, frontend needs work
3. **Contact Management** - Backend controller missing
4. **Forum Feature** - Backend not implemented
5. **Resource Library** - Backend not implemented
6. **Glossary** - Backend not implemented
7. **Profile Management** - Backend partially done
8. **Frontend UI Pages** - Many pages are placeholders
9. **Comprehensive Testing** - Only 10% coverage
10. **Email Sending** - No SMTP configured

3.4 Technology Stack Overview

3.4.1 Backend (Yasser's Domain)

- **Framework:** ASP.NET Core 8.0
- **Language:** C# 12
- **Database:** PostgreSQL (primary), MongoDB (messages)
- **Caching:** Redis
- **Storage:** MinIO (S3-compatible)
- **Real-time:** SignalR (WebSocket-based)
- **ORM:** Entity Framework Core 8.0
- **Auth:** ASP.NET Core Identity + JWT
- **API Docs:** Swagger/OpenAPI

3.4.2 Frontend (Zainab's Domain)

- **Framework:** React 18
- **Language:** TypeScript 5.x
- **Build Tool:** Vite

- **State Management:** React Context API
- **Styling:** CSS Modules + Sass
- **HTTP Client:** Axios
- **WebRTC:** simple-peer library
- **SignalR Client:** @microsoft/signalr
- **Testing:** Jest + React Testing Library

3.4.3 ML Service (Shared Integration)

- **Framework:** FastAPI (Python)
- **ML:** TensorFlow + ONNX Runtime
- **Computer Vision:** MediaPipe
- **Streaming:** WebSocket
- **Status:** Demo mode (mock predictions)

3.5 Repository Structure

```
SilentTalkFYP/
          server/                 # Backend API (Yasser)
                src/
                      SilentTalk.Api/
                            Controllers/     # API endpoints
                            Hubs/            # SignalR hub
                            Program.cs       # App entry point
                      SilentTalk.Domain/   # Entities/models
                database/migrations/
          client/                 # Frontend (Zainab)
                src/
                      pages/         # Page components
                      components/    # Reusable components
                      services/      # API clients
                      contexts/      # React contexts
                      design-system/ # Design components
                      hooks/         # Custom hooks
                      types/         # TypeScript types
                package.json
                vite.config.ts
          ml-service/            # ML Service (Integration)
                app/
                      api/          # FastAPI endpoints
                      services/     # ML inference
                      main.py       # FastAPI app
                requirements.txt
          infrastructure/        # Docker configs
                docker/
                    docker-compose.yml
          docs/                  # Documentation
          start.sh              # Start everything
          stop.sh               # Stop everything
          PROJECT_STATUS.md     # Detailed status
```

4 Task Assignments & Division of Labor

4.1 Guiding Principles

1. **Clear Boundaries:** Yasser owns backend, Zainab owns frontend
2. **Communication:** Daily updates on blockers, weekly planning
3. **Code Reviews:** Review each other's PRs within 24 hours
4. **Shared Goals:** Both responsible for project success
5. **Flexibility:** Help each other when needed

4.2 Yasser's Tasks (Backend/Infrastructure)

codebg #	**Task**	**Priority**	**Time**	**Status**
Y1	Complete Contact Management Backend	High	1 week	Todo
Y2	Implement Email Service (SMTP)	High	3 days	Todo
Y3	Connect Profile Endpoints to DB	Medium	2 days	Todo
Y4	Forum Backend Implementation	Medium	2 weeks	Todo
Y5	Resource Library Backend	Medium	1.5 weeks	Todo
Y6	Glossary Backend API	Low	1 week	Todo
Y7	ML Model Training Setup	Critical	2 weeks	Todo
Y8	Backend Unit Tests (80% coverage)	High	3 weeks	Todo
Y9	API Performance Optimization	Medium	1 week	Todo
Y10	Security Audit & Fixes	High	1 week	Todo
Y11	Deployment Scripts & CI/CD	Medium	1 week	Todo
Y12	Documentation Updates	Low	Ongoing	Todo

Table 1: Yasser's Task List

4.3 Zainab's Tasks (Frontend/UI)

code #	**Task**	**Priority**	**Time**	**Status**
Z1	Complete WebRTC Video Integration	Critical	2 weeks	Todo
Z2	Connect Call History Page to Backend	High	3 days	Todo
Z3	Profile Page Backend Integration	High	4 days	Todo
Z4	Contact Management UI	High	1 week	Todo
Z5	Call Scheduling UI	High	1 week	Todo
Z6	Forum Frontend Implementation	Medium	2 weeks	Todo
Z7	Resource Library Frontend	Medium	1.5 weeks	Todo
Z8	Glossary Frontend	Low	1 week	Todo
Z9	Responsive Design Polish	High	1 week	Todo
Z10	Accessibility Testing (WCAG 2.1)	High	1 week	Todo
Z11	Frontend Unit Tests	Medium	2 weeks	Todo
Z12	User Documentation	Medium	1 week	Todo
Z13	UI/UX Polish	Medium	Ongoing	Todo

Table 2: Zainab's Task List

4.4 Shared Tasks (Collaboration Required)

code #	**Task**	**Who Leads**	**Time**
S1	ML Model Integration	Yasser (backend), Zainab (UI)	1 week
S2	End-to-End Testing	Both	2 weeks
S3	Performance Testing	Yasser (API), Zainab (UI)	1 week
S4	Security Testing	Both	3 days
S5	User Acceptance Testing	Both	1 week
S6	Bug Fixing Sprint	Both	2 weeks
S7	Final Documentation	Both	1 week
S8	Deployment	Yasser (infra), Zainab (test)	3 days

4.5 Dependencies Matrix

Understanding task dependencies prevents blockers:

codebg **Frontend Task**	**Depends On (Backend)**
WebRTC Video (Z1)	SignalR Hub Already done!
Call History (Z2)	Call endpoints Ready!
Profile Page (Z3)	User endpoints Yasser: Y3
Contact UI (Z4)	Contact backend Yasser: Y1
Forum UI (Z6)	Forum backend Yasser: Y4
Resource Library (Z7)	Resource backend Yasser: Y5
Glossary (Z8)	Glossary backend Yasser: Y6

Key Insight: Zainab can start Z1 (WebRTC) and Z2 (Call History) immediately because backend is ready!

5 Implementation Roadmap (January - July 2026)

5.1 Month-by-Month Plan

5.1.1 January 2026: Setup & Foundation

Goals:

- Zainab completes environment setup
- Both team members understand codebase
- Weekly meeting cadence established

Deliverables:

☐ Zainab can run project locally

☐ Git workflow established

☐ Weekly meeting schedule set

☐ Communication channels confirmed

Tasks (Week by Week):

codebg **Week**	**Activities**
Week 1 (Jan 6-12)	**Zainab:** Environment setup, run project locally, explore codebase **Yasser:** Code walkthrough for Zainab, document current APIs
Week 2 (Jan 13-19)	**Zainab:** Study existing components, review auth flow **Yasser:** Start Y1 (Contact backend)
Week 3 (Jan 20-26)	**Zainab:** Start Z2 (Call history page) **Yasser:** Continue Y1
Week 4 (Jan 27-31)	**Both:** First mini-sprint review, plan February

5.1.2 February 2026: Core Features

Goals:

- WebRTC video calling working end-to-end
- Contact management complete
- Call history integrated

Deliverables:

☐ Users can make 1-on-1 video calls

☐ Contact list shows online/offline status

☐ Call history displays past calls with filters

☐ Profile page connected to backend

Tasks (Week by Week):

codebg **Week**	**Activities**
Week 5 (Feb 3-9)	**Zainab:** Start Z1 (WebRTC integration - Part 1) **Yasser:** Finish Y1 (Contact backend), start Y3 (Profile DB)
Week 6 (Feb 10-16)	**Zainab:** Z1 (WebRTC - Part 2), Z2 complete **Yasser:** Y3 complete, start Y2 (Email service)
Week 7 (Feb 17-23)	**Zainab:** Z3 (Profile page), Z4 start (Contact UI) **Yasser:** Y2 complete, review Z1
Week 8 (Feb 24-28)	**Both:** Integration testing video calls, sprint review

5.1.3 March 2026: ML Integration & Captions

Goals:

- ML model trained and integrated
- Real-time captions working in video calls
- Call scheduling implemented

Deliverables:

☐ Sign language recognition (85%+ accuracy)

☐ Live captions during calls

☐ Call scheduling calendar

☐ Contact management complete

Tasks:

- **Yasser:** Y7 (ML training), S1 (ML integration)
- **Zainab:** Z4 complete, Z5 (Scheduling UI), S1 (ML UI)

5.1.4 April 2026: Community Features

Goals:

- Forum operational
- Resource library functional
- Glossary searchable

Deliverables:

☐ Users can create/reply to forum posts

☐ Resource library with videos/tutorials

☐ Searchable sign language glossary

Tasks:

- **Yasser:** Y4 (Forum backend), Y5 (Resources), Y6 (Glossary)
- **Zainab:** Z6 (Forum UI), Z7 (Resources UI), Z8 (Glossary UI)

5.1.5 May 2026: Polish & Responsive Design

Goals:

- All features responsive on mobile/tablet
- Accessibility compliance (WCAG 2.1 AA)
- UI/UX polished

Deliverables:

☐ Mobile-responsive design

☐ Accessibility audit passed

☐ Performance optimized

Tasks:

- **Yasser:** Y9 (Performance), Y10 (Security audit)
- **Zainab:** Z9 (Responsive), Z10 (Accessibility), Z13 (UI polish)

5.1.6 June 2026: Testing & Bug Fixes

Goals:

- Comprehensive testing complete
- All critical bugs fixed
- 80%+ code coverage

Deliverables:

☐ Unit tests (80%+ coverage)

☐ Integration tests

☐ End-to-end tests

☐ User acceptance testing

Tasks:

- **Yasser:** Y8 (Backend tests)
- **Zainab:** Z11 (Frontend tests)
- **Both:** S2-S6 (E2E, performance, security, UAT, bug fixes)

5.1.7 July 2026: Final Deployment & Submission

Goals:

- Production deployment
- Documentation complete
- FYP submitted

Deliverables:

☐ Live production system

☐ Complete documentation

☐ User manual

☐ Final presentation ready

☐ FYP report submitted

Tasks:

- **Both:** S7 (Documentation), S8 (Deployment), Final report

codebg **Date**	**Milestone**	**Success Criteria**
Jan 31	Setup Complete	Zainab can develop independently
Feb 28	MVP Ready	Video calls + contact management working
Mar 31	ML Integrated	Real sign recognition in calls
Apr 30	Feature Complete	All major features implemented
May 31	Polish Complete	Responsive, accessible, performant
Jun 30	Testing Complete	80%+ coverage, bugs fixed
Jul 15	Deployment	Live production system
Jul 31	FYP Submission	Documentation & report submitted

5.2 Critical Milestones

6 Detailed Task Implementation Guides

6.1 For Zainab: WebRTC Video Integration (Z1)

Priority: Critical **Time Estimate:** 2 weeks

6.1.1 Overview

Implement peer-to-peer video calling using WebRTC and the existing SignalR hub.

6.1.2 Prerequisites

- SignalR hub is fully implemented (CallHub.cs)
- Frontend has CallSignalingClient service
- simple-peer library can be used

6.1.3 Step-by-Step Implementation

Step 1: Install Dependencies

```
cd ~/SilentTalkFYP/client
npm install simple-peer @types/simple-peer
```

Step 2: Create WebRTC Service
Create `src/services/webrtc.service.ts`:

```
import Peer from 'simple-peer';

export class WebRTCService {
  private localStream: MediaStream | null = null;
  private peers: Map<string, Peer.Instance> = new Map();

  async getLocalStream(): Promise<MediaStream> {
    if (!this.localStream) {
      this.localStream = await navigator.mediaDevices
        .getUserMedia({ video: true, audio: true });
    }
    return this.localStream;
  }

  createPeer(userId: string, initiator: boolean): Peer.Instance {
    const peer = new Peer({ initiator, stream: this.localStream });
    this.peers.set(userId, peer);
    return peer;
  }

  // More methods for sending offers/answers
}
```

Listing 1: webrtc.service.ts (partial)

Step 3: Update VideoCallPage Component

Integrate WebRTC with SignalR:

```
const VideoCallPage = () => {
  const [localStream, setLocalStream] = useState<MediaStream>();
  const [remoteStreams, setRemoteStreams] = useState<Map>();
  const signalingClient = useSignalRConnection();

  useEffect(() => {
    // Get local media
    webrtcService.getLocalStream().then(setLocalStream);

    // Listen for SignalR events
    signalingClient.on('userJoined', handleUserJoined);
    signalingClient.on('receiveOffer', handleOffer);
    signalingClient.on('receiveAnswer', handleAnswer);

    // Join call
    signalingClient.joinCall(callId);
  }, []);

  const handleUserJoined = (userId: string) => {
    // Create peer and send offer
    const peer = webrtcService.createPeer(userId, true);
    peer.on('signal', signal => {
      signalingClient.sendOffer(userId, signal);
    });
  };

  // More handlers...
};
```

Listing 2: VideoCallPage.tsx (partial)

Step 4: Test Video Calling

1. Open two browser windows

2. Login as different users
3. Start a call
4. Verify video/audio streaming

6.1.4 Success Criteria

- ☐ Local video displays in call
- ☐ Remote video displays from other user
- ☐ Audio works bidirectionally
- ☐ Can mute/unmute audio
- ☐ Can toggle video on/off
- ☐ Call ends gracefully

6.1.5 Common Issues & Solutions

Issue: Camera permission denied

Solution: Add HTTPS in development or use localhost exception. Check browser console for permission errors.

Issue: SignalR disconnects

Solution: Implement reconnection logic in CallSignalingClient. The backend hub already supports ReconnectToCall().

6.2 For Zainab: Call History Page (Z2)

Priority: High **Time Estimate:** 3 days

6.2.1 Backend API (Already Ready!)

The CallController has these endpoints ready:

```
GET /api/call/history?page=1&pageSize=20&status=Ended
GET /api/call/statistics
```

6.2.2 Implementation Steps

Step 1: Create Call History Service

```
import axios from 'axios';

export interface CallHistoryItem {
  callId: string;
  initiatorName: string;
  startTime: string;
  endTime: string;
  duration: number; // minutes
  status: 'Ended' | 'Cancelled';
  recordingUrl?: string;
}

export const callHistoryService = {
  async getHistory(page = 1, pageSize = 20) {
    const response = await axios.get('/api/call/history', {
      params: { page, pageSize }
    });
    return response.data;
  },

  async getStatistics() {
    const response = await axios.get('/api/call/statistics');
    return response.data;
  }
};
```

Listing 3: src/services/callHistory.service.ts

Step 2: Update CallHistoryPage Component

```
const CallHistoryPage = () => {
  const [calls, setCalls] = useState<CallHistoryItem[]>([]);
  const [stats, setStats] = useState(null);
  const [page, setPage] = useState(1);

  useEffect(() => {
    loadHistory();
    loadStats();
  }, [page]);

  const loadHistory = async () => {
    const data = await callHistoryService.getHistory(page);
    setCalls(data.items);
  };

  return (
    <div className="call-history">
      <h1>Call History</h1>
      <CallStatistics stats={stats} />
      <CallList calls={calls} />
      <Pagination page={page} onPageChange={setPage} />
    </div>
  );
};
```

Listing 4: src/pages/CallHistoryPage.tsx (partial)

Step 3: Test

1. Make a few test calls
2. End the calls
3. Navigate to Call History page
4. Verify calls are listed with correct details

6.2.3 Success Criteria

- ☐ Call history loads from backend
- ☐ Pagination works
- ☐ Statistics display (total calls, duration)
- ☐ Can click to view call details
- ☐ Recording download link works (if available)

6.3 For Yasser: Contact Management Backend (Y1)

Priority: High **Time Estimate:** 1 week

6.3.1 Overview

Implement backend API for contact/friend management. The Contact entity already exists.

6.3.2 Implementation Steps

Step 1: Create ContactController

Create `server/src/SilentTalk.Api/Controllers/ContactController.cs`:

```
[ApiController]
[Route("api/[controller]")]
[Authorize]
public class ContactController : ControllerBase
{
    private readonly ApplicationDbContext _context;

    [HttpGet]
    public async Task<IActionResult> GetContacts()
    {
        var userId = User.GetUserId();
        var contacts = await _context.Contacts
            .Where(c => c.UserId == userId &&
                        c.Status == ContactStatus.Accepted)
            .Include(c => c.ContactUser)
            .ToListAsync();

        return Ok(contacts);
    }

    [HttpPost("request")]
    public async Task<IActionResult> SendContactRequest(
        [FromBody] ContactRequestDto dto)
    {
        // Create contact with Pending status
        // Send notification to other user
    }

    [HttpPost("{contactId}/accept")]
    public async Task<IActionResult> AcceptRequest(Guid contactId)
    {
        // Update status to Accepted
    }

    [HttpPost("{contactId}/block")]
    public async Task<IActionResult> BlockContact(Guid contactId)
    {
        // Update status to Blocked
    }
}
```

Listing 5: ContactController.cs (partial)

Step 2: Add DTOs

```
public class ContactRequestDto
{
    public string Email { get; set; } // or UserId
}

public class ContactDto
{
    public Guid ContactId { get; set; }
    public string DisplayName { get; set; }
    public string Email { get; set; }
    public string ProfileImageUrl { get; set; }
    public ContactStatus Status { get; set; }
    public bool IsOnline { get; set; }
}
```

Listing 6: DTOs

Step 3: Test Endpoints
Use Swagger or curl:

```bash
# Get contacts
curl -H "Authorization: Bearer <token>" \
  http://localhost:5000/api/contact

# Send request
curl -X POST \
  -H "Authorization: Bearer <token>" \
  -H "Content-Type: application/json" \
  -d '{"email":"friend@example.com"}' \
  http://localhost:5000/api/contact/request
```

6.3.3 Success Criteria

- ☐ Can send contact request
- ☐ Can accept/reject requests
- ☐ Can block contacts
- ☐ Can get list of contacts
- ☐ Online status updated via SignalR
- ☐ Unit tests written

6.4 For Both: End-to-End Testing (S2)

Priority: High **Time Estimate:** 2 weeks

6.4.1 Tools

- **Frontend:** Playwright or Cypress
- **Backend:** xUnit integration tests

6.4.2 Critical User Flows to Test

1. **Authentication Flow**
   - Register → Email verification → Login → Access protected page
2. **Video Call Flow**
   - Login → Schedule call → Join call → Video/audio working → End call
3. **Contact Management Flow**
   - Login → Add contact → Accept request → Start call with contact

6.4.3 Implementation Example (Playwright)

```
import { test, expect } from '@playwright/test';

test('user can register and login', async ({ page }) => {
  // Register
  await page.goto('http://localhost:3000/register');
  await page.fill('[name="email"]', 'test@example.com');
  await page.fill('[name="password"]', 'Test123!');
  await page.fill('[name="confirmPassword"]', 'Test123!');
  await page.fill('[name="displayName"]', 'Test User');
  await page.click('button[type="submit"]');

  // Verify redirect to login
  await expect(page).toHaveURL('/login');

  // Login
  await page.fill('[name="email"]', 'test@example.com');
  await page.fill('[name="password"]', 'Test123!');
  await page.click('button[type="submit"]');

  // Verify redirect to home
  await expect(page).toHaveURL('/');
  await expect(page.locator('text=Test User')).toBeVisible();
});
```

Listing 7: tests/e2e/auth.spec.ts

7 Collaboration Workflow

7.1 Weekly Meeting Structure

7.1.1 Monday Planning Meeting (30 minutes)

Agenda:

1. Review previous week's accomplishments
2. Discuss blockers and dependencies
3. Plan current week's tasks
4. Assign priorities

Format:

- Each person shares: What I did, what I'm doing, blockers
- Update task board (Trello/Notion/GitHub Projects)
- Set 3-5 key goals for the week

7.1.2 Wednesday Quick Sync (15 minutes)

Agenda:

- Progress check
- Any blockers?
- Any help needed?

7.1.3 Friday Review (30 minutes)

Agenda:

1. Demo completed features
2. Review code quality
3. Discuss learnings
4. Preview next week

7.2 Git Workflow

7.2.1 Branch Naming Convention

```
# Feature branches
feature/z1-webrtc-integration
feature/y1-contact-backend

# Bug fixes
fix/login-validation-error
fix/video-stream-freeze

# Improvements
improve/responsive-navbar
improve/api-performance
```

7.2.2 Development Workflow

Step 1: Create Feature Branch

```
# Always branch from main
git checkout main
git pull origin main

# Create feature branch
git checkout -b feature/z2-call-history
```

Step 2: Develop & Commit

```bash
# Make changes
# ...

# Stage and commit
git add .
git commit -m "feat: add call history page with pagination"

# Follow conventional commits:
# feat: new feature
# fix: bug fix
# docs: documentation
# style: formatting
# refactor: code restructuring
# test: adding tests
# chore: maintenance
```

Step 3: Push & Create PR

```bash
# Push to remote
git push origin feature/z2-call-history

# Create Pull Request on GitHub
# - Add description
# - Link related issues
# - Request review from partner
```

Step 4: Code Review

- Partner reviews within 24 hours
- Address feedback
- Merge when approved

7.2.3 Pull Request Template

```markdown
## Description
Brief description of changes

## Type of Change
- [ ] New feature
- [ ] Bug fix
- [ ] Breaking change
- [ ] Documentation update

## Testing Done
- [ ] Unit tests added/updated
- [ ] Manually tested locally
- [ ] Tested with other features

```

```
## Screenshots (if UI change)
[Add screenshots]

## Checklist
- [ ] Code follows style guidelines
- [ ] Self-review completed
- [ ] Comments added for complex code
- [ ] No console.log() left in code
- [ ] Documentation updated
```

7.3 Code Review Guidelines

7.3.1 What to Look For

1. **Functionality:** Does it work as intended?
2. **Code Quality:** Clean, readable, maintainable?
3. **Best Practices:** Follows conventions?
4. **Performance:** Any optimization issues?
5. **Security:** No vulnerabilities?
6. **Testing:** Adequate test coverage?

7.3.2 Review Comments Guidelines

- Be constructive, not critical
- Explain the "why" behind suggestions
- Use prefixes:
  - *NIT:* Minor nitpick (optional)
  - *Q:* Question for clarification
  - *BLOCKER:* Must fix before merge
  - *SUGGESTION:* Consider this approach
- Praise good code!

7.4 Communication Best Practices

7.4.1 Daily Updates (Async)

Use WhatsApp/Slack for quick updates:

Example Daily Update

Today's Progress:
Completed call history page
Connected to backend API
Working on pagination component

Tomorrow:
Finish pagination
Start profile page integration

Blockers:
None

7.4.2 Asking for Help

When stuck, provide context:

Good Help Request

Problem: SignalR connection keeps dropping during video calls

What I tried:

- Checked network tab - seeing 101 switching protocols
- Added reconnection logic
- Still disconnects after 2 minutes

Error: [paste error message]

Question: Is there a timeout setting in the backend hub?

7.5 Task Management

7.5.1 Recommended Tool: GitHub Projects

Create a project board with columns:

- **Backlog** - All tasks
- **Todo** - Planned for current sprint
- **In Progress** - Currently working on
- **Review** - Awaiting code review
- **Done** - Completed and merged

7.5.2 Sprint Structure (2-week sprints)

1. **Sprint Planning** (Monday Week 1)
   - Select tasks from backlog
   - Assign to team members
   - Estimate time
2. **Daily Work** (async updates)
3. **Mid-Sprint Check** (Wednesday Week 1)
4. **Sprint Review** (Friday Week 2)
   - Demo completed features
   - Move unfinished tasks to next sprint

8 Technical Reference

8.1 Backend API Endpoints

8.1.1 Authentication Endpoints

codebg **Method**	**Endpoint**	**Description**
POST	/api/auth/register	Register new user
POST	/api/auth/login	Login and get JWT token
POST	/api/auth/logout	Invalidate refresh token
POST	/api/auth/refresh	Refresh access token
GET	/api/auth/verify-email	Verify email with token
POST	/api/auth/forgot-password	Request password reset
POST	/api/auth/reset-password	Reset password with token
GET	/api/auth/me	Get current user info

Table 3: Auth Endpoints

8.1.2 Call Management Endpoints

codebg **Method**	**Endpoint**	**Description**
POST	/api/call/schedule	Schedule a call
POST	/api/call/start	Start instant call
GET	/api/call/{id}	Get call details
POST	/api/call/{id}/end	End call
GET	/api/call/history	Get call history (paginated)

codebg **Method**	**Endpoint**	**Description**
GET	/api/call/statistics	Get user call statistics
POST	/api/call/{id}/recording	Upload recording
GET	/api/call/{id}/recording	Download recording URL

Table 4: Call Endpoints

8.2 SignalR Hub Methods

8.2.1 CallHub Events

codebg **Method**	**Description**
JoinCall(callId)	Join a call room
LeaveCall(callId)	Leave call room
SendOffer(userId, offer)	Send WebRTC offer
SendAnswer(userId, answer)	Send WebRTC answer
SendIceCandidate(userId, candidate)	Send ICE candidate
UpdateMediaState(audio, video)	Toggle audio/video
SendChatMessage(callId, message)	Send in-call chat
StartScreenshare()	Start screen sharing
StartRecording()	Start call recording

Table 5: SignalR Hub Methods

8.3 Essential Commands Reference

8.3.1 Docker Commands

```
# Start application
cd ~/SilentTalkFYP
./start.sh

# Stop application
./stop.sh

# View logs
cd infrastructure/docker
docker-compose logs -f server       # Backend
docker-compose logs -f client       # Frontend
docker-compose logs -f ml-service   # ML service

```

```
# Restart specific service
docker-compose restart server

# Rebuild after code changes
docker-compose up -d --build server

# Check running containers
docker ps

# Check service health
curl http://localhost:5000/health
curl http://localhost:8000/status
```

8.3.2 Frontend Development Commands

```
cd ~/SilentTalkFYP/client

# Install dependencies
npm install

# Start dev server (if not using Docker)
npm run dev

# Build for production
npm run build

# Run tests
npm test

# Run linter
npm run lint

# Format code
npm run format
```

8.3.3 Backend Development Commands

```
cd ~/SilentTalkFYP/server

# Build solution
dotnet build

# Run API (if not using Docker)
dotnet run --project src/SilentTalk.Api

# Run tests
dotnet test
```

```

# Create migration
dotnet ef migrations add MigrationName \
  --project src/SilentTalk.Infrastructure

# Update database
dotnet ef database update \
  --project src/SilentTalk.Api
```

8.4 Common Issues & Solutions

8.4.1 Frontend Issues

Issue: CORS error when calling API

Symptom: Browser console shows "CORS policy blocked"

Solution:

1. Verify backend CORS is configured in Program.cs
2. Check VITE_API_URL is correct (`http://localhost:5000`)
3. Ensure backend container is running
4. Check Network tab for actual URL being called

Issue: SignalR connection fails

Symptom: "Failed to start the connection"

Solution:

- Check VITE_WS_URL is correct (ws://localhost:5000)
- Verify backend SignalR hub is running
- Check browser console for detailed error
- Ensure JWT token is included in connection

8.4.2 Backend Issues

Issue: Database connection fails

Symptom: API throws connection timeout

Solution:

1. Check PostgreSQL container is running: `docker ps`
2. Verify connection string in appsettings.json
3. Check database is initialized: `docker exec -it silents-talk-postgres psql -U silentstalk`
4. Run migrations: `dotnet ef database update`

8.5 Performance Optimization Tips

8.5.1 Frontend Optimization

1. **Code Splitting**

```
// Lazy load pages
const VideoCallPage = lazy(() =>
  import('./pages/VideoCallPage')
);
```

2. **Memoization**

```
// Prevent unnecessary re-renders
const MemoizedComponent = React.memo(MyComponent);
```

3. **Image Optimization**
   - Use WebP format
   - Lazy load images
   - Use appropriate sizes

8.5.2 Backend Optimization

1. **Database Queries**
   - Use Select() to project only needed fields
   - Add indexes on frequently queried columns
   - Use AsNoTracking() for read-only queries
2. **Caching**

```
// Cache frequently accessed data
var cachedData = await _cache.GetOrCreateAsync(
    key: "user:profile:123",
    factory: async () => await GetUserProfile(123),
    expiration: TimeSpan.FromMinutes(5)
);
```

9 Quality Assurance Checklist

9.1 Before Each Pull Request

- ☐ Code builds without errors
- ☐ All tests pass
- ☐ No console.log() or debugger statements
- ☐ Code follows style guide
- ☐ Comments added for complex logic
- ☐ No hardcoded credentials or secrets
- ☐ Responsive design tested (mobile/tablet/desktop)
- ☐ Browser compatibility checked
- ☐ Accessibility checked (keyboard navigation, screen reader)

9.2 Before Each Demo/Milestone

- ☐ All features working end-to-end
- ☐ No critical bugs
- ☐ Performance acceptable (<2s page load)
- ☐ Data persists across restarts
- ☐ Error handling graceful
- ☐ Documentation updated
- ☐ Demo script prepared

9.3 Final Submission Checklist

- ☐ All features from requirements implemented
- ☐ ML model accuracy 85%
- ☐ Video calling works reliably
- ☐ Accessibility compliance (WCAG 2.1 AA)
- ☐ Security audit passed
- ☐ Performance benchmarks met
- ☐ Code test coverage 80%
- ☐ User documentation complete
- ☐ Admin documentation complete
- ☐ Deployment guide written
- ☐ Final report submitted

10 Conclusion & Final Notes

10.1 Keys to Success

1. **Communication is Critical**
   - Daily async updates
   - Weekly sync meetings
   - Ask questions early
   - Share blockers immediately
2. **Stay Organized**
   - Use task board (GitHub Projects)
   - Follow Git workflow
   - Document decisions
   - Track time estimates
3. **Quality Over Speed**
   - Write tests
   - Review code thoroughly
   - Refactor when needed
   - Don't skip documentation

4. **Support Each Other**
    - Help when partner is stuck
    - Share learnings
    - Celebrate small wins
    - Be patient and respectful

10.2 Emergency Contacts

codebg **Person**	**Contact**
Yasser	[Email/Phone]
Zainab	[Email/Phone]
FYP Supervisor	[Email]

10.3 Resources

- **Documentation Folder:** `/docs` in repository
- **Project Status:** `PROJECT_STATUS.md`
- **API Docs:** `http://localhost:5000/docs` (when running)
- **Quick Start:** `QUICK_START.md`

10.4 Acknowledgments

This project represents months of hard work and dedication. By following this collaboration guide, maintaining clear communication, and supporting each other, you will successfully deliver a high-quality Final Year Project that makes a real difference in the deaf and hard-of-hearing community.

Good luck, and happy coding!

Version 1.0 - January 2026
Last Updated: November 24, 2025