Title: Building a WebRTC Application: A Software Engineering Approach

Introduction

- What is WebRTC?
  - Web Real-Time Communication (WebRTC) is a collection of open-source standards and APIs that enable real-time, peer-to-peer communication (audio, video, and data) directly within web browsers and mobile applications.
- Motivation
  - Overcoming Plugin Limitations: WebRTC eliminates the need for external plugins, simplifying development and improving the user experience.
  - Cross-Platform Compatibility: WebRTC functions seamlessly across various devices and operating systems.
  - Open-Source and Cost-Effectiveness: Its open-source nature promotes innovation and avoids licensing fees.
  - Security Emphasis: WebRTC prioritizes security with protocols like DTLS and SRTP.

Software Requirements Specification (SRS)

- Functional Requirements
  - Media Capture: The ability to access and stream audio and video data from the user's devices.
  - Peer-to-Peer Connectivity: Establishing direct connections between clients, including NAT traversal techniques.
  - Codec Support: Implementing mandatory codecs (Opus, VP8/VP9) and optional codecs for extended functionality.
  - Data Sharing: Reliable and efficient transfer of arbitrary data between peers.
- Non-Functional Requirements
  - Performance: Low latency and high frame rates for real-time communication.
  - Scalability: Accommodating a variable number of participants.
  - Security: Enforcing encryption and authentication to protect user data.
  - Cross-Browser Compatibility: Consistent functionality across major browsers.

Design

- System Architecture
  - Client-Side (Web Browser): Core WebRTC JavaScript APIs (, ,).
  - Signaling Server: Handles peer discovery, session negotiation, and message routing (often implemented with WebSockets).
  - STUN/TURN Servers: Assist with NAT traversal for connections behind firewalls.
- Design Patterns
  - Observer Pattern: For notifications about media stream or connection state changes.
  - Adapter Pattern: To manage differences in WebRTC implementations across

browsers.

Coding/Implementation

- Languages and Frameworks
    - JavaScript: Primary language for client-side WebRTC logic.
    - Node.js: Popular choice for building signaling servers.
    - WebRTC Libraries: Consider libraries like simple-peer or PeerJS to streamline development.
- Key Code Considerations
    - Media Stream Acquisition: Use to obtain camera and microphone streams.
    - Peer Connection Management: Create objects and handle ICE candidates and SDP offers/answers.
    - Data Channel Establishment: Set up instances for data transfers.

Testing

- Unit Testing: Test individual WebRTC components in isolation.
- Integration Testing: Verify interactions between WebRTC, the signaling server, and other application parts.
- End-to-End Testing: Simulate real-world scenarios for a comprehensive evaluation.
- Performance Testing: Measure and optimize latency, packet loss, and frame rate under various network conditions.

Maintenance

- Monitoring: Track usage, errors, and performance metrics.
- Library and Browser Updates: Stay updated for compatibility and new features.
- Dependency Management: Address security vulnerabilities in external libraries.
- Scalability Management: Plan for capacity adjustments to handle usage fluctuations.