Real-Time Chat Application Design

Designing a comprehensive real-time chat application addressing all the required components with detailed explanations and sample code.

Technology Stack Selection

Cloud Provider: AWS

AWS provides a robust ecosystem for real-time applications with services like:

- **Amazon EC2** for hosting the Node.js backend servers
- **Amazon ElastiCache** for Redis caching to improve message delivery performance
- **AWS Lambda** for serverless functions handling specific tasks (image processing, notifications)
- **Amazon CloudFront** for content delivery
- **Amazon S3** for storing media files shared in chats

Database: MongoDB

MongoDB is ideal for chat applications because:

- **Document-oriented structure** aligns perfectly with chat messages and user profiles
- **Horizontal scaling** via sharding for handling growing user bases
- **Change streams** support for real-time updates
- **Flexible schema** allows easy addition of new message features without migrations

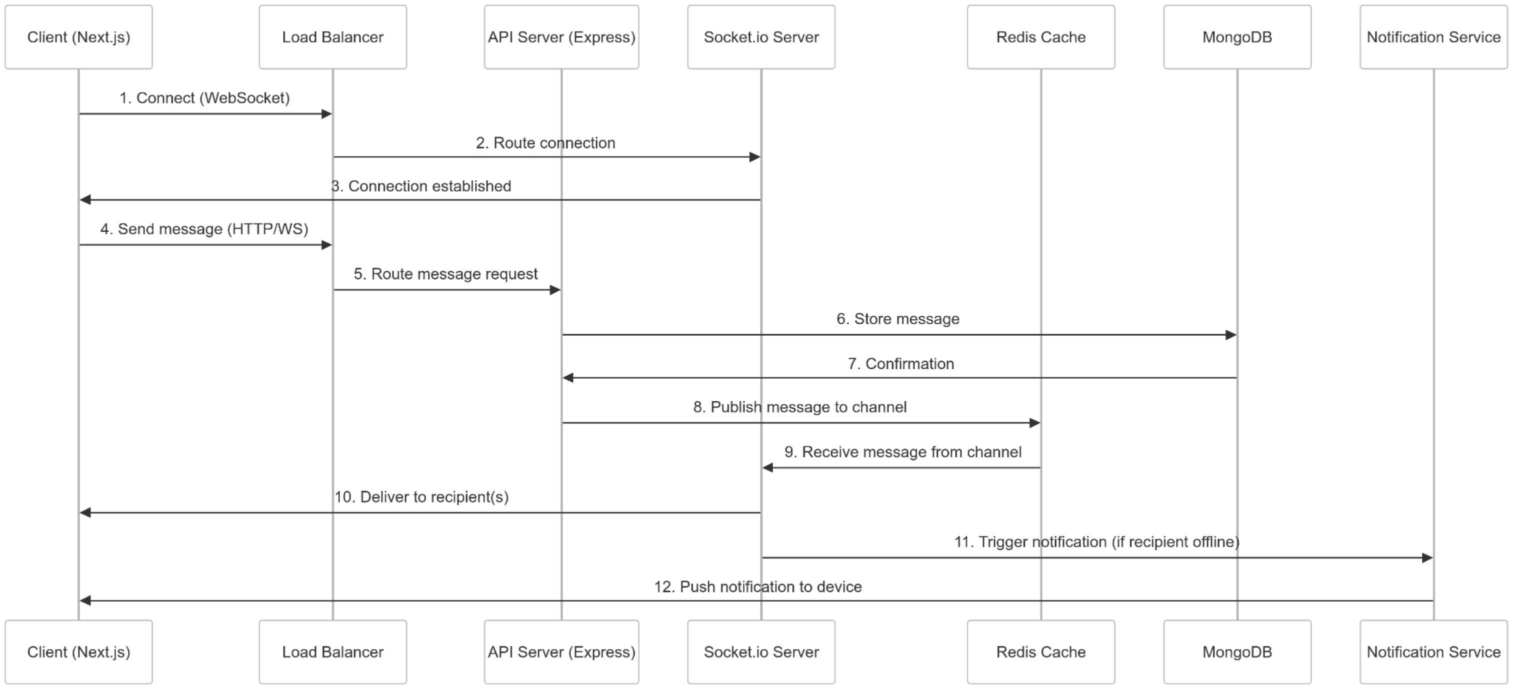
Backend: Node.js with Express and Socket.io

- **Express** for RESTful API endpoints
- **Socket.io** for real-time bidirectional communication
- **JWT** for authentication
- **Mongoose** as ODM for MongoDB interactions

Frontend: Next.js

- **React** for component-based UI development
- **Next.js** for server-side rendering and improved performance
- **Tailwind CSS** for responsive design

Message Flow Architecture



API Design

Here’s your content organized into a clean table format:

📌 API Endpoints

Category	Method	Route	Description	Access
Authentication	POST	/api/auth/register	Register a new user	Public
	POST	/api/auth/login	Authenticate user & get token	Public
	GET	/api/auth/me	Get current user's profile	Private
Conversations	POST	/api/conversations	Create a new conversation (1:1 or group)	Private
	GET	/api/conversations	Get all conversations for current user	Private

	GET	/api/conversations/:id	Get a specific conversation with messages	Private
	PUT	/api/conversations/:id	Update conversation (rename, add/remove members)	Private
Messages	POST	/api/messages	Send a new message	Private
	GET	/api/messages/:conversationId	Get messages for a conversation (with pagination)	Private
	PUT	/api/messages/:id/read	Mark message as read	Private
	DELETE	/api/messages/:id	Delete a message	Private
Users	GET	/api/users/search	Search users by username or email	Private
	GET	/api/users/:id/status	Get online status of a user	Private

📡 WebSocket Events

Event Name	Description
`connection`	Client connects to WebSocket server
`disconnect`	Client disconnects from WebSocket server
`join_conversation`	Client joins a specific conversation's room
`leave_conversation`	Client leaves a conversation's room
`typing`	User is typing in a conversation
`stop_typing`	User stopped typing
`new_message`	New message broadcast to conversation members

`message_read`	Message marked as read by recipient

Optimization Strategies for Low Latency and Scalability

1. Database Optimization

- **Sharding** MongoDB across multiple servers based on conversation IDs
- **Indexing** critical fields (userId, conversationId, timestamps)
- **Time-To-Live (TTL)** indexes for temporary data
- **Read replicas** for distributing read operations

2. Caching Strategy

- **Redis** for:
  - User presence information (online/offline status)
  - Recent messages (LRU cache)
  - Active conversations
  - Rate limiting data

3. Message Delivery Optimization

- **Fan-out on write** for active users
- **Lazy loading** for inactive conversations
- **Pagination** for message history
- **Compression** for message payloads

4. Horizontal Scaling

- **Stateless API servers** behind load balancers
- **Socket.io with Redis adapter** for multi-server setup
- **Microservices** for specialized functions (notification delivery, file processing)
- **Auto-scaling** based on traffic patterns

5. Performance Monitoring

- **Prometheus** for metrics collection
- **Grafana** for visualization
- **Distributed tracing** to identify bottlenecks
- **APM tools** for real-time monitoring

Demo Code

In the github link provided