Technical Report

ChatsApp

Nhat Nam Ha

Table of content

Description

Chat applications have become an integral part of modern communication, offering users a platform to interact, share information, and stay connected in real-time. However, developing a chat application comes with its own set of challenges, which need to be addressed to ensure a seamless user experience. The target audiences are for the small-medium groups in the organizers, companies for private usage.

Use cases

User Registration

Actors:	New User
Description:	A new user can create an account by providing necessary information such as name, email address, and password. The system verifies the information and creates a new user profile.
Preconditions:	The user must provide valid information.
Postconditions:	A new account is created, and the user is logged into the system.
Steps:	1. User navigates to the registration page. 2. User enters required information (e.g., name, email, password). 3. System validates the information. 4. System creates a new user profile. 5. System sends a confirmation email to the user. 6. User confirms the registration through the email link. 7. User is redirected to the login page.

User Login

Actors:	Registered User
Description:	A registered user can log in to their account by entering their email address and password. The system authenticates the credentials and grants access to the user.
Preconditions:	The user must have a registered account and valid credentials.
Postconditions:	The user is authenticated and logged into the system.
Steps:	1. User navigates to the login page. 2. User enters email address and password. 3. System verifies the credentials. 4. If credentials are valid, user is granted access. 5. User is redirected to the chat interface.

Initiating a Chat

Actors:	Registered User
Description:	A user can initiate a chat with another user by selecting their contact from the contact list and sending a message.
Preconditions:	Both users must be registered and logged into the system.
Postconditions:	A chat session is initiated, and messages can be exchanged.
Steps:	1. User navigates to the contact list. 2. User selects a contact. 3. User types a message and clicks send. 4. System delivers the message to the selected contact. 5. The contact receives a notification of the new message.

Receiving Messages

Actors:	Registered User
Description:	A user receives messages from other users in real-time. The system notifies the user of new messages.
Preconditions:	The user must be logged in.
Postconditions:	The user receives and can read new messages.
Steps:	1. System detects an incoming message for the user. 2. System delivers the message to the user's chat interface.

	3. User receives a notification of the new message. 4. User opens the chat to read the message.

File Sharing in Chat

Actors:	Registered User
Description:	Users can share files (e.g., documents, images, videos) within a chat session. The system allows users to upload and download shared files.
Preconditions:	Users must be registered, logged in, and in an active chat session.
Postconditions:	The file is uploaded and shared with the chat participants.
Steps:	1. User selects the file sharing option in the chat interface. 2. User uploads the desired file. 3. System uploads the file and generates a link. 4. System delivers the link to chat participants. 5. Participants can click the link to download the file.

Functional requirements

- Users can register for a new account
- Users can log-in to use application
- Users send the chat to a recipient and recipient receives the chat instantly.
- Users can upload shared media file so that both sender and recipient access the attachment.

Non-functional requirements

- Performance
  - Chat messages must be sent and received in real-time, latest is 1 second.
  - The front-end application can be loaded within 3 seconds.
- Scalability
  - The backend application can handle at least 1000 requests in a second.
  - The chosen databases can handle at least 1000 requests in a second.
- Compatibility
  - This chat application does not have a mobile version.
  - Files sharing limited in document-based like PDF, Word, and media files like images music.
- Reliability
  - File sharing can be successful in 80% of the use cases
- Availability
  - The website is available at least 99.99% all the time for every month.
- Security
  - If the username previously existed, username needed to retype.
  - If the password does not follow strong password requirements, then retype again.
  - User credential must be encrypted on both transmission and storing.
  - Chat messages, media files and video streaming must be encrypted on transmission.
  - Chat messages have the maximum length of 255 characters.
  - Chat messages content must not contain poisonous code.
  - Disallow executable files and compressed files for sharing or size larger than 1MB.
- Privacy/GDPR
  - User credentials, chat history and media files store in Azure Database Services.
  - These databases and storages are not public and exposed to users.
  - If the users request to delete account, the users' credential will be deleted in 12 months from database.
  - Recipient accepts the sender's call invitation then establishing video call.

Estimated Impact

User Types:

- Regular User: Sends and receives messages, uploads files.
- Admin User: Manages user accounts, monitors chat logs.

Estimated Requests and Responses:

Assuming regular users:

- 20 messages sent per hour, 60 messages received per hour,
- 5 file uploads per day, 5 file downloads per day, and 2 logins per day.

	Send message	Receive Message	Upload File	Download File	User Login
Request Size	500 bytes	100 bytes	2 MB	200 bytes	300 bytes
Request size per hour	20 * 500 bytes = 10,000 bytes	60 * 100 bytes = 6,000 bytes	(5 * 2)/24 MB = 416.66 KB	(5 * 200) / 24 bytes = 41.66 bytes	(2 * 300) / 24 bytes = 25 bytes
Response Size	200 bytes	500 bytes	300 bytes	2 MB	1 KB
Response size per hour	20 * 200 bytes = 4,000 bytes	60 * 500 bytes = 30,000 bytes	(5 * 300) / 24 bytes = 62.5 bytes	(5 * 2) / 24 MB = 416.66 KB	(2 * 1) / 24 KB = 0.083 KB

Estimated Scaling Up

Calculations for 1000, 10,000, 1,000,000, and 10,000,000 Users per hour.

Requests

Users	Send message	Receive Message	Upload File	Download File	User Login	Total size
1	10 KB	6 KB	416.66 KB	0.04166 KB	0.025 KB	433.1016 KB
1000	10 MB	6 MB	416.66 MB	0.04166 MB	0.025 MB	433.1016 MB
10000	100 MB	60 MB	4,166.6 MB	0.41666 MB	0.25 MB	4,331.016 MB
100.000	1 GB	0.6 GB	41.666 GB	0.004166 GB	0.0025 GB	43.31016 GB
1.000.000	10 GB	6 GB	416.66 GB	0.04166 GB	0.025 GB	433.1016 GB
10.000.000	100 GB	60 GB	4166.6 GB	0.4166 GB	0.25 GB	4,331.016 GB

Responses

Users	Send message	Receive Message	Upload File	Download File	User Login	Total size
1	4 KB	30 KB	0.0625 KB	416.66 KB	0.083 KB	450.8055 KB
1000	4 MB	30 MB	0.0625 MB	416.66 MB	0.083 MB	450.8055 MB
10000	40 MB	300 MB	0.625 MB	4,166.6 MB	0.83 MB	4508.055 MB
100.000	400 MB	3000 MB	6.25 MB	41,666 MB	8.3 MB	45080.55 MB
1.000.000	4 GB	30 GB	0.0625 GB	416.66 GB	0.083 GB	450.8055 GB
10.000.000	40 GB	300 GB	0.625 GB	4166.6 GB	0.83 GB	4,508.055 GB

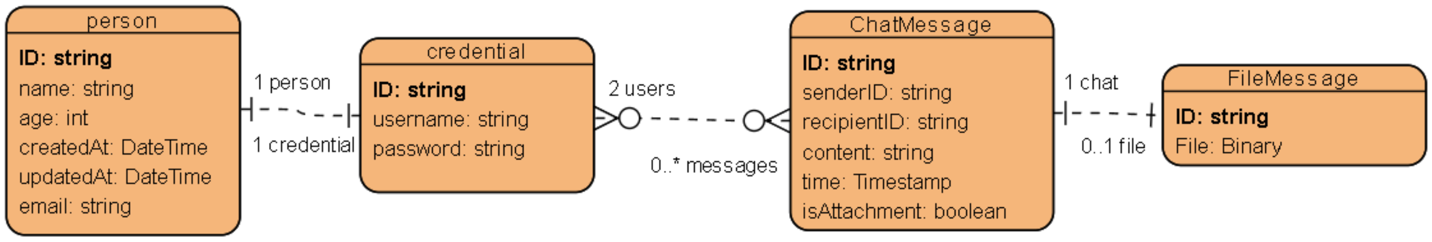
Choice of technologies

	Choice of technology	Reason of choice
Front-end framework	JavaScript, React framework	Well-known JavaScript framework, most web components follow the React architectures

Back-end language	C# and .NET	Microsoft programming language and support asynchronous programming, other computer science terminologies.
Cloud provider	Azure	Microsoft product, the services cost cheaper compares to Amazon Web Services, provide student subscription for education.

This project requires on numerous non-functional requirements (performance, security, maintainability, service costs), so the choice of technologies requires highly customizable instead of low or no-code framework.

Data Entities (fix relationship)



There are 4 main data entities will be used for ChatsApp project, every main entity will have object ID:

- Credential: store username and password for login purpose.
- Information: store personal info and date time is created and updated.
- ChatMessage: store history of chat for pair of users.
- FileMessage: store file messages with artifact stored.

A credential entity when user logged in has relationships with one person entity. A user can have multiple, or none chat messages, but a chat message must be owned by 2 different users. A chat message can have with or without a file message related to, a file message when existed must have relationships with one chat message.

Choice of data storages

For ChatsApp application, there are different data entities must be handled, by using different data storages can help increasing the performance by avoiding the bottleneck and wrong database's use cases because of single or less number database/storages.

For the CAP theorem, represents three kinds of guarantees that architects aim to provide for choosing distributed data systems:

- Consistency: All nodes in the system have the same view of the data at any time (like the C in ACID).
- Availability: The system always responds to requests.
- Partition tolerance: The system remains operational if network problems occur between system nodes.

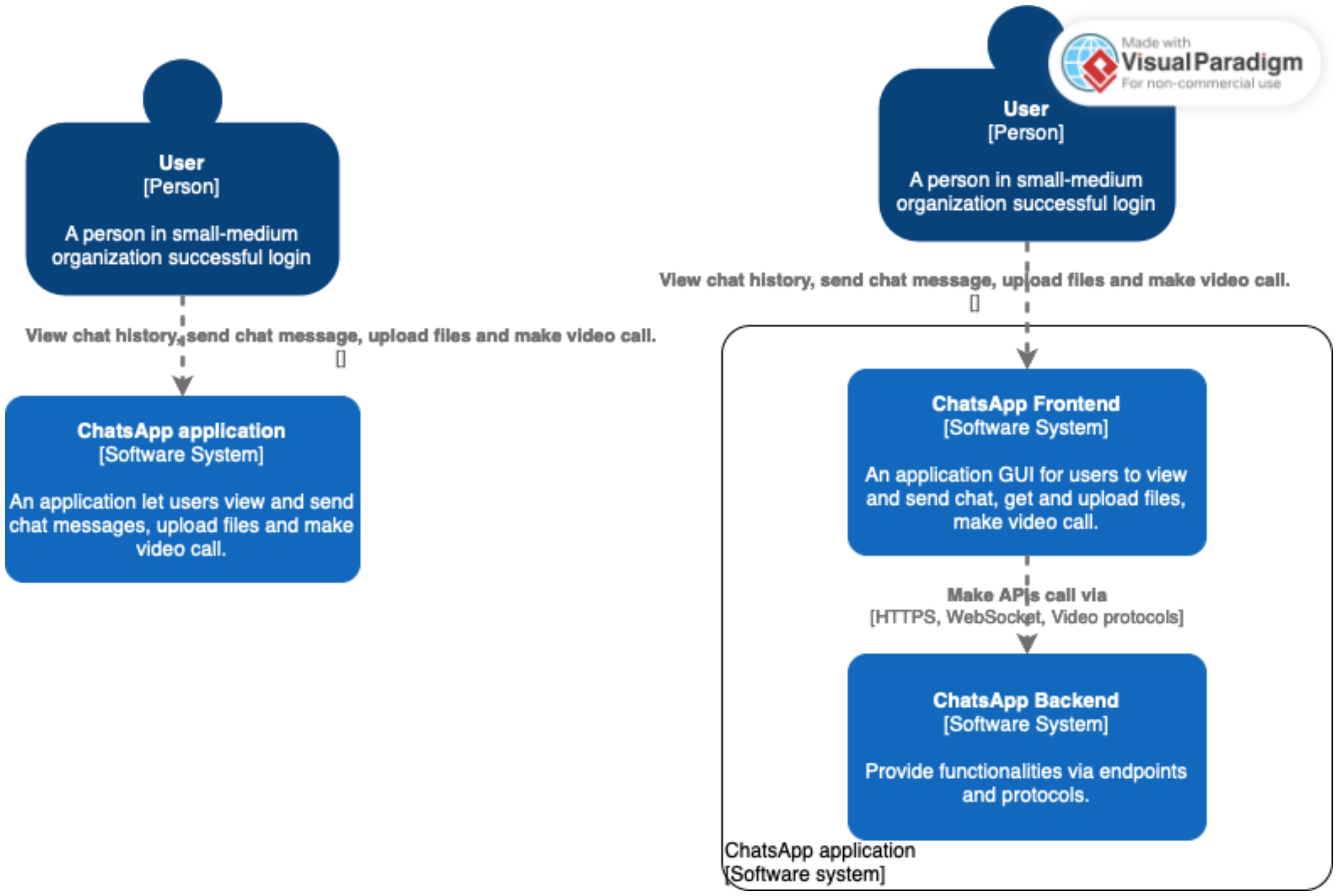
From the technical point of view, database supports consistency and partition tolerance will be preferred because of consistent data across all nodes ensures that users see the correct chat message, even if it leads to some delays for the user, however by applying message broken pattern can reduce the availability issues.

Data entities	Type of data storage	Reason of choice
User credentials and Information	SQL database	User credentials are the structured data which can be visualize as a table, a SQL database is suitable choice for the table database.
Chat history	NoSQL database	A conversation between users is unstructured entity because of number of chat messages, the chat message itself is a structure entity, NoSQL is suitable for the unstructured data entities.
Media files	Blob storage	Media files are the binary data type, which is not suitable for a database requires data in text-based, a blob storage is preferred.

The choice of data storage engines has been researched and MongoDB is the main choice for storing the messages and files since it also supporting storing binary files and using database querying.

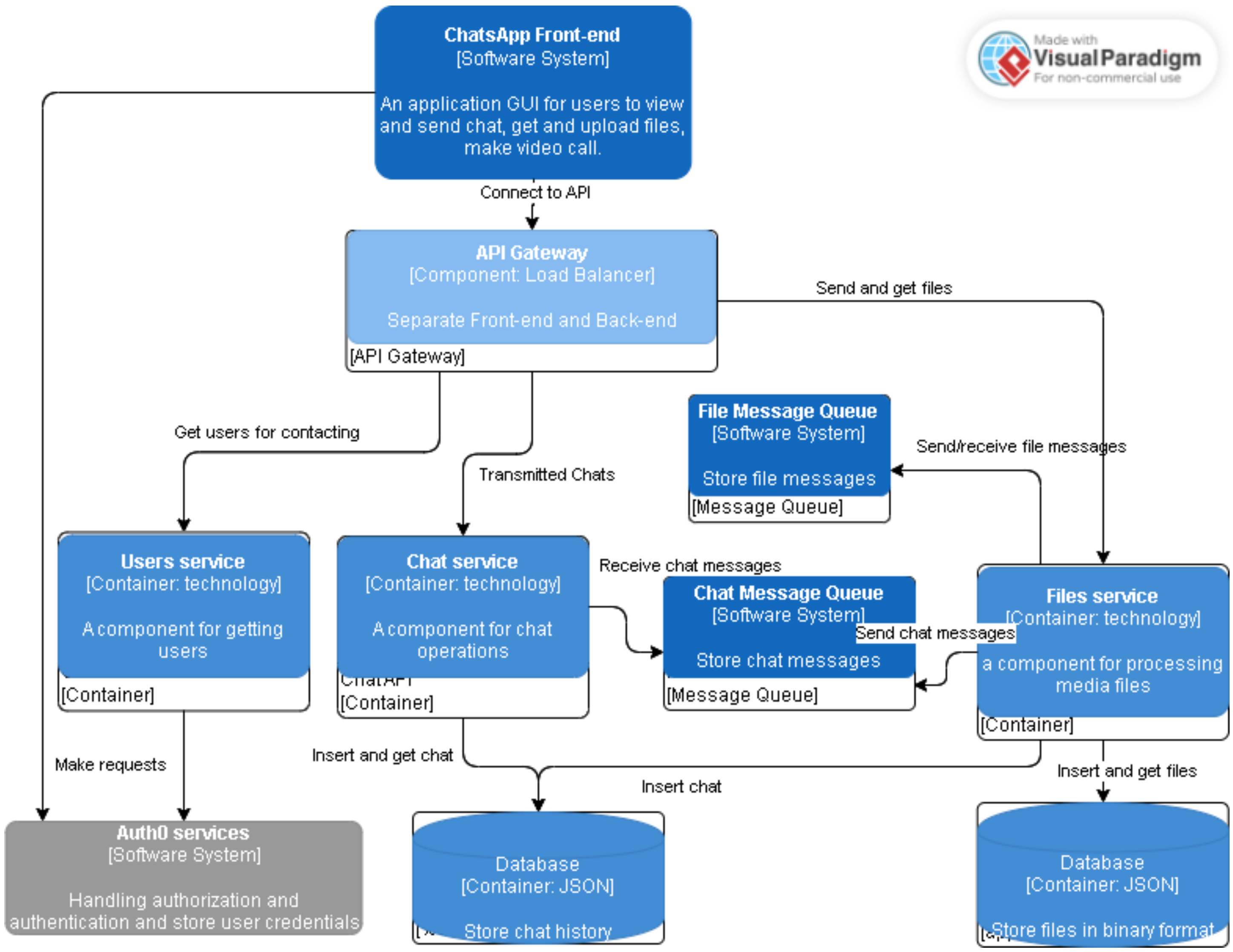
C4 Models

The below image is the C1 and C2 diagrams for the ChatsApp application which describes users interact to the Frontend application that connects to the Backend exposes endpoints for transmitting communication.


User
[Person]
A person in small-medium organization successful login
View chat history, send chat message, upload files and make video call.
ChatsApp application
[Software System]
An application let users view and send chat messages, upload files and make video call.
Made with Visual Paradigm For non-commercial use
User
[Person]
A person in small-medium organization successful login
View chat history, send chat message, upload files and make video call.
ChatsApp Frontend
[Software System]
An application GUI for users to view and send chat, get and upload files, make video call.
Make APIs call via
[HTTPS, WebSocket, Video protocols]
ChatsApp Backend
[Software System]
Provide functionalities via endpoints and protocols.
ChatsApp application
[Software system]

The image below is "more detail" C2 diagrams describes the details of backend architecture which consists of controllers for each entity, the data layer consists of multiple storages and databases.

The "more detail" C2 diagrams consist of a Front-end and an API Gateway for centralize endpoints. The message queue handling messages between microservices interaction. For each microservice will run each own environment, isolating components, and interacting by networking or message queue. Databases for storing the chat messages and file messages.



Apply API Gateway pattern

The API Gateway is a design pattern used in software engineering to facilitate communication and interaction between components or services in a distributed system. In this pattern, a vital component called the “gateway” acts as an intermediary or go-between, facilitating communication and coordination between other components or services.

The main challenge that can be solved by using API Gateway pattern is Decoupling Components, in this case separate the Front-end and backend application:

- API Gateway enables loose coupling between components.
- This decoupling improves system flexibility, as components can evolve independently without directly depending on each other’s interfaces.

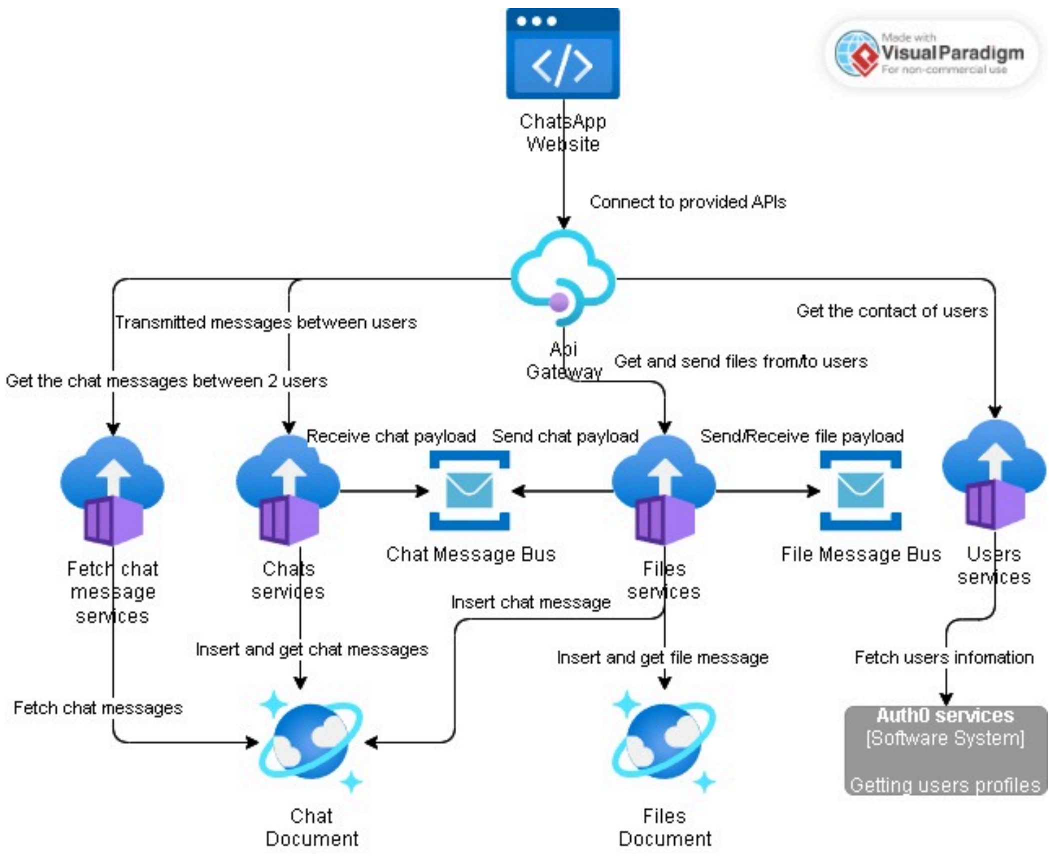
Resilience is applicable for file sharing to guarantee is available to both sender and receiver:

- API Gateway enhances system resilience by providing fault tolerance mechanisms.

Interoperability (as the Facade design pattern to hide the Backend complexities):

- Components can communicate seamlessly regardless of their implementation details or programming languages, facilitating integration and collaboration in distributed environments.
- API Gateway promotes interoperability between heterogeneous systems and technologies.

Azure diagrams



Frontend with Azure Static Website: Host your application's user interface using Azure Static Websites. This ensures fast, scalable, and secure delivery of the static assets.

Backend with API Management: Use Azure API Management to manage and secure the backend APIs that provide the dynamic functionality required for chatting, file handling, and video calls.

Azure Container Apps provide a managed way to run containerized applications in the cloud:

- Handle the logic for retrieving chat messages from a database or a similar storage system and responding to API requests.
- Handle the logic for transmitting chat messages from/to a database or a similar storage system to users using WebSocket technology.
- Handle the logic for posting and retrieving files from a database or a similar storage system and responding to API requests.
- Handle the logic for retrieving users from Auth0 services and responding to API requests.

Azure Service Bus is a fully managed enterprise message broker that allows you to decouple and scale your applications using asynchronous messaging:

- The Container App enqueues the message (chat or file) to the relevant Azure Service Bus queue.
- The message (chat or file) is held in the Azure Service Bus queue until processed.
- For Chat Messages: Dequeues the chat message, processes it, and stores it in the database.
- For File Messages: Dequeues the file message, uploads the file to Azure Cosmos DB, and stores/retrieves metadata in/from the database.

Azure Cosmos DB is a fully managed NoSQL database service designed for modern application development:

- Each chat message can be stored as a document in a Cosmos DB container.
- Each file message can be stored as a document in a Cosmos DB container.