MediFusion Technology Stack Q&A

1) Why is JWT used in web applications?

JWT (JSON Web Token) is used primarily for **stateless authentication**.

- **General**: Unlike traditional sessions where the server stores a session ID in a database/memory for every logged-in user, JWTs are self-contained. The server gives the user a "badge" (token) and doesn't need to remember it. When the user comes back, the server just checks the signature on the badge to validity.
- **In Your Project**: It allows your FastAPI backend to scale easily. You don't need a database lookup just to check if a user is logged in for every API call.

2) What is Redis Cluster?

- **Definition**: Redis Cluster is a distributed implementation of Redis that automatically splits your dataset across multiple nodes (sharding). It provides high availability and keeps running even if some nodes fail.
- **In Your Project**: You are **NOT** using Redis Cluster. You are using a **single standalone Redis instance** (`medifusion-redis` container) as defined in your `docker-compose.yml`. For this scale, a single instance is perfectly fine.

3) What is JWT? How did you implement it?

- **Definition**: **JSON Web Token (JWT)** is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. It consists of three parts separated by dots (`.`):

1. **Header**: The algorithm used (e.g., HS256).
2. **Payload**: Data (claims) like user ID, expiration time.
3. **Signature**: A hash of the Header + Payload + Secret Key to prevent tampering.

- **In Your Project**: I implemented it in `backend/routers/auth.py`:
  - **Creation**: When a user logs in, the `create_access_token` function takes their email (`sub`), sets an expiration time (`exp`), and signs it using your `SECRET_KEY`.
  - **Storage**: The frontend stores this token and sends it in the `Authorization: Bearer <token>` header for future requests.

4) What is Redis?

- **Definition**: **Redis** (Remote Dictionary Server) is an open-source, in-memory data structure store. It is extremely fast because it keeps data in RAM rather than on a slow hard disk. It is commonly used as a database, cache, and message broker.
- **In Your Project**: It is used as the **Result Backend** for Celery (`backend=REDIS_URL` in `worker.py`). When your AI worker finishes a prediction task, it writes the result (the JSON diagnosis) into Redis so the main API can retrieve it if needed.

5) What is FastAPI?

- **Definition**: FastAPI is a modern, high-performance web framework for building APIs with Python 3.8+ based on standard Python type hints. It is one of the fastest Python frameworks available.
- **In Your Project**: It is the core of your backend (`app = FastAPI()`). It handles all the routing (e.g., `@router.post("/login")`), data validation (using Pydantic models), and automatically generates the Swagger documentation.

6) What is Logstash?

- **Definition**: Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and sends it to a "stash" like Elasticsearch.
- **In Your Project**: **It is NOT used.** You are currently logging simply to the standard output (console), which Docker captures.

7) What is Celery and how is it going to work?

- **Definition**: Celery is a distributed task queue that allows you to run time-consuming tasks in the background so your website doesn't freeze while waiting for them.
- **In Your Project**:

1. **Frontend** sends a request (e.g., "Predict Disease").
2. **FastAPI** puts a task message onto the RabbitMQ queue and immediately returns "Task Started".
3. **Celery Worker** picks up the task from RabbitMQ.
4. It runs the `predict_disease` function (calling Gemini AI), which takes 2-5 seconds.
5. It saves the result to the Database and Redis.

8) How does backend verify in JWT?

- **In Your Project**: This happens in `backend/routers/auth.py` inside the `get_current_user` function.

1. **Decode**: It uses `jwt.decode(token, SECRET_KEY...)`.
2. **Signature Check**: The library automatically re-hashes the header+payload with the `SECRET_KEY` you have on the server. If it matches the signature in the token, it proves the token hasn't been tampered with.
3. **Expiration Check**: It checks if the current time > the `exp` timestamp.

9) What is RabbitMQ? How does it help in background processing?

- **Definition**: RabbitMQ is a message broker. Think of it as a "Post Office". It accepts messages from producers (FastAPI) and delivers them to consumers (Celery Workers).
- **In Your Project**: It manages the **Queue**. If 100 users request an AI diagnosis at once, RabbitMQ holds those 100 requests in a line, and your worker processes them one by one without crashing the server.

10) What is message acknowledgement in your project?

- **Definition**: Acknowledgement (ACK) is a signal sent back from the worker to the broker (RabbitMQ) saying "I have finished processing this message, you can delete it now."
- **In Your Project**: I configured `task_acks_late=True` in `backend/worker.py`. This means the worker only sends the ACK **after** the AI prediction is fully complete. If the worker crashes *during* the AI call, RabbitMQ won't see the ACK and will give the task to another worker, ensuring zero data loss.

11) What is Elasticsearch?

- **Definition**: Elasticsearch is a distributed, RESTful search and analytics engine. It stores data in an Inverted Index that makes searching through millions of documents nearly instant.
- **In Your Project**: **It is NOT used.** You are currently querying your PostgreSQL database directly using SQLAlchemy.