Question 1: Explain the setup and the architecture

Answer:

- Backend will listen on port 8080:

  python3 -m venv .venv; source .venv/bin/activate; pip3 install -r requirements.txt; python main.py

- Frontend will listen on port 3000:

  npm install; npm run start

- use the browser to access http://localhost:3000 to signup and signin. Frontend uses conditional rendering based on the state of authentication instead of routing.
- Backend uses Python and FastAPI, frontend uses React and MUI, user information is stored in SQLite database, backend access database through ORM, a successful authentication request generates JWT token and it is stored in browser cookie
- /api/me is protected, it does not require user information exist in request parameter or request body, instead, the access_token is automatically included in request Cookie because of the use of “withCredentials”

Question 2: Potential weaknesses

Answer:

- File database cannot be accessed by application replicas
- Table is created at the first time of running the application, when there is change for domain class, the database has to be recreated or the tables has to be manually updated
- Id is incrementally generated, it is easy to be guessed and cause risks of security
- To address above issues,
  - Use separate database server
  - Use python migration library(Alembic etc)
  - Generate random string or UUID as key

Question 3: If you have more time, what would you improve next?

Answer:

- Avoid hard coded values and backend root url, use environment variable
- Adding logging information
- Add database index if there are more fields and complex queries
- Implement refresh token
- Implement layout at frontend
- Use react-router to create multiple route in SPA

- Add test case to increase coverage, mockup test data and parameterise it

Question 4: state management approach and validation library

Answer:

- In this implement, I stored authentication via state management approach for convenience
- An ecommerce website normally stores shopping cart information in state
- Redux is a widely used central state management library

- In this implementation, frontend used MUI to have the basic validation(required, email), to implement complex frontend validation, we can use Yup. Backend should also validate user's entry before persist it

Question 5: Type and contracts

Answer:

- Backend exposed API through Swagger, the data types can be synced by using openapi-generator-cli, the generated code can be imported to application component files to process RESTful requests

Question 6: Brute-force attack on logins

Answer:

- Implement these solutions:
    - Limit number of login attempts
    - Lockout the user if login attempts failed more than allowed times
    - Use captcha
- If it is deployed to AWS, some of these solutions can be implemented in WAF
- Some solutions can also be implemented in AWS Cognito if users are stored in Cognito user pool

Question 7: Handle millions request/sec

Answer:

- In a traditional implementation, in order to improve concurrency throughput, the infrastructure could be changed a lot, involving scaling horizontally at multiple layers, use dedicate database to handle specific data in readonly mode, implement table index, use cache(database, orm, application level, Redis, CDN), if the request calls slow third party services, we can consider preload data and store some data locally, use Kafka to process heavy task asynchronously, allow using multiple partitions to process multiple messages at the same time

- In AWS environment, if the requests are implemented by using Lambda function and API Gateway, we only need to contact AWS support to allow it to scale horizontally more. We can schedule preloading some data into S3, or use CloudFront to cache response for a certain time if it does not change frequently
- To enable scaling, the request has to be stateless
- When unstable web services are involved, we have to consider resilience.