Question 4:

To migrate ORM using sequelize V3 to the V4 we need to have perfect planning and execution to move it effortless.

1) Risk Assessment:

To ensure this part we need to recognise the possible risk and challenges which we can face to make it move.
We also need to make sure to check the similarity of the code that is already done and the database to make sure it's has schema with the V4.
Judge the possible changes and features that can be split to have a sequlize V4.

2) Effort Required:

To evaluate the effort required we need to check the lenght and complication of the code that is in V3.
There will also some dependencies on the code which need to updated or optimised.
Testing phase will take some time to ensure this transition so that effort need to be counted as well.

3) Development Task:

The first thing is to place the tasks according to the functionality and all the dependencies.
Then We need to spilt the tasks in smaller parts so that they can be done easily.
And then after each task we need to check the possible conficts and resolve them to have a smooth process.

4) Rollout and Deployment Strategy:

To avoid any disturbance we should have a proper plan.
Test the updated code before pushing it into the production.
After testing then push the code in to the migration.
While this phase we need to check precisely for possbile issues.

5) Monitoring and Error Tracking:

We must have structure to get the possbile errors.
API execution and queries in database need to be observed.
For any condemning issue we must have an alert.
Check the test reports and logs carefully to recogise any issue.

Question5.1:
To have such an application, I'd use a CMS or an API to fetch the content dynamically. Additionally, I'd use a fallback content for when the CMS or API would fail.

Question 5.2:
The best approach in my opinion is that whenever a image is uploaded, a background service will scale it into a few different predefined sizes like 128x128 and 64x64, sizes that are common. The images will be hosted on a backend URL with a flag/query param that would specify the size. Something like this 'https://cdn.example.com/image123?size=128x128'. Furthermore, we can also implement caching to further reduce the server overload. Alternatively, we can also use a third party service for image resizing and management.

To make sure if the files have been transferred or not, we can install a checksum verification method that will ensure that the files transferred are not corrupted. And for weak connections, we can transfer the data in chunks instead of sending the whole file as one. This would help us verify the checksum integrity of smaller chunks and we can re-send those chunks if they're corrupted.

Question 5.3:
1. Choose a popular CI/CD platform that supports building, testing, and deploying mobile apps example GitLab CI/CD, or GitHub Actions.
2. Utilize Fastlane, a popular automation tool for both iOS and Android, to streamline the build and release process.
3. Define a configuration file that contains all the necessary build settings, environment variables, and credentials required for building and signing your app.
4. Store your app signing credentials (e.g., Keystore for Android, P12 file for iOS) securely in your CI/CD platform's secrets.
5. For iOS, to manage code signing and provisioning profiles automatically. Fastlane's match and cert tools simplify handling code signing certificates and profiles.
6. Save the generated .ipa and .apk files as artifacts after a successful build. Also, automate the deployment process to distribute the app to testers or app stores for beta and production releases.