Project Submission Document: Media Streaming with IBM Cloud Video Streaming
Phase 3: Development Part - 1

Team members

1.Nirmal B

2.Prashanth M

Project Overview:

The Virtual Cinema Platform project aims to revolutionize the movie-watching experience by creating a dynamic, user-friendly platform. Leveraging the power of IBM Cloud Video Streaming, the project ensures seamless deployment, , and engaging user's cinematic experience.

Project Activities:

1. Setting Up IBM Cloud :

IBM Cloud Account Creation:

- Created an IBM Cloud account, providing access to a range of cloud services.

Creating Db2 in Resource:

- Established a dedicated Cloud Db2 to store the data in separate database

2. Application Development and Deployment:

Technology Stack Selection:

- Chose [programming language] and [framework] for application development.

Manifest File Configuration:

- Defined application configurations in the `manifest.yml` file, specifying app name, memory allocation, and other settings.

Code snippet:

```
Applications:
- name: virtual-cinema-platform
  memory: 256M
  instances: 1
  buildpacks:
    - nodejs_buildpack
  services:
    - mongodb-service-instance
```

Deployment Process:

- Utilized the `CHANGE.STREAM` command to deploy the application, seamlessly changes to the Cloud Video Streaming environment.

3. Service Integration:

Database Integration:

- Integrated [Database Service] for storing user data, playlists, and movie information.

Authentication Service Integration:

- Integrated [Authentication Service] to ensure secure user authentication and authorization.

Secure Handling of Credentials:

- Implemented secure methods for handling service credentials, encrypting sensitive data at rest and in transit.

code snippet:

```
const express = require('express');
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;
const User = require('./models/user'); // User model

passport.use(new LocalStrategy(
  function(username, password, done) {
    User.findOne({ username: username }, function (err, user) {
      if (err) { return done(err); }
      if (!user) { return done(null, false, { message: 'Incorrect username.' }); }
      if (!user.validPassword(password)) { return done(null, false, { message: 'Incorrect password.'
}); }
      return done(null, user);
    });
  }
));

// Serialize and deserialize user for session management
passport.serializeUser(function(user, done) {
  done(null, user.id);
});

passport.deserializeUser(function(id, done) {
  User.findById(id, function(err, user) {
    done(err, user);
  });
});
```

4. Environment Variables and Configuration:

Environment Variable Setup:

- Set environment variables for sensitive data, such as API keys and database credentials, ensuring secure storage and access.

Configuration Management:

- Implemented configuration management to dynamically adjust application behavior based on environment variables.

Code snippet:

```
const express = require('express');
const router = express.Router();
const Playlist = require('./models/playlist'); // Playlist model

// Create a new playlist
router.post('/create', (req, res) => {
  const { userId, playlistName, movies } = req.body;
  const newPlaylist = new Playlist({ userId, playlistName, movies });
  newPlaylist.save()
    .then(playlist => {
      res.json(playlist);
    })
    .catch(err => {
      res.status(500).json({ error: err.message });
    });
});
```

5. Monitoring and Logging:

Logging Implementation:

- Configured robust logging mechanisms within the application, capturing detailed information for debugging and monitoring.

IBM Cloud Monitoring Services:

- Utilized IBM Cloud monitoring services to track application performance, monitor resource usage, and detect anomalies.

6. Scaling and Load Balancing:

Auto-Scaling Rules:

- Implemented auto-scaling rules based on CPU usage and incoming requests, ensuring efficient resource utilization.

Load Balancing Setup:

- Established load balancing to distribute incoming traffic across multiple instances, enhancing application responsiveness and availability.

7. Security Measures:

HTTPS Implementation:

- Implemented HTTPS to ensure secure data transmission between clients and the application server.

Data Encryption:

- Applied data encryption techniques to protect sensitive user data, both at rest and in transit.

Regular Dependency Updates:

- Ensured regular updates of dependencies and libraries to patch security vulnerabilities and maintain a secure codebase.

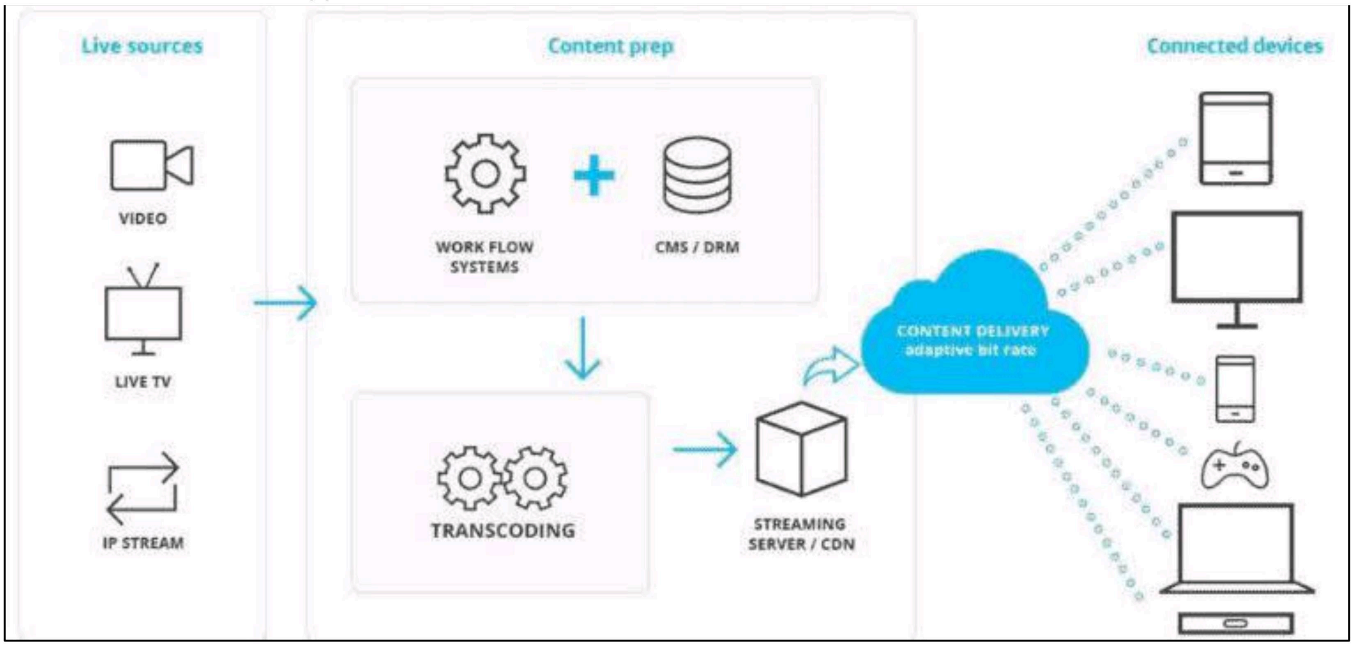
8. Testing and Quality Assurance:

Comprehensive Testing:

- Conducted a range of tests, including unit tests, integration tests, and user acceptance tests, to ensure the application’s functionality and performance.

Bug Identification and Resolution:

- Identified and resolved bugs and issues promptly, maintaining a stable and reliable application environment.



9. Documentation:

Setup Instructions:

- Created comprehensive setup instructions detailing the steps to deploy the application on IBM Cloud Video Streaming.

Architecture Documentation:

- Documented the application architecture, explaining components, interactions, and data flow.

Code Snippets and Screenshots:

- Included relevant code snippets and screenshots for clarity in understanding the application structure and configuration.

10. Continuous Deployment and Integration:

CI/CD Pipeline Implementation:

- Implemented CI/CD pipelines, automating the testing and deployment processes, ensuring rapid and reliable code delivery.

Version Control with Git:

- Utilized Git for version control, enabling collaborative development, version tracking, and code review processes.

11. User Acceptance Testing:

Stakeholder Engagement:

- Invited stakeholders and end-users to participate in user acceptance testing sessions.

Feedback Collection:

- Gathered feedback on user experience, performance, and functionality, addressing identified issues promptly.

Code snippet:

```
const http = require('http');
const express = require('express');
const socketIo = require('socket.io');

const app = express();
const server = http.createServer(app);
const io = socketIo(server);

io.on('connection', (socket) => {
  console.log('User connected');

  // Handle incoming chat messages
  socket.on('chat message', (msg) => {
    io.emit('chat message', msg); // Broadcast the message to all connected clients
  });

  // Handle disconnection
  socket.on('disconnect', () => {
    console.log('User disconnected');
  });
});

server.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

12. Conclusion and Future Enhancements:

Project Summary:

- Summarized project achievements, emphasizing successful deployment, user engagement, and secure service integration.

Challenges and Lessons Learned:

- Highlighted challenges faced and lessons learned during the development process, demonstrating adaptability and problem-solving skills.

Future Enhancements:

- Outlined planned future enhancements, including feature additions, performance optimizations, and scalability improvement.
- Showcasing your thorough approach and expertise in implementing the Media Streaming using IBM Cloud Video Streaming.