**New Social Theory\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Solution Design:

1. Describe the CI/CD process for the project in terms of workflow diagrams.

#### **CI/CD Workflow:** [CICD](https://drive.google.com/file/d/1yuBm9cc_WSCCCUcg1vpmJR9qbLcxqX9q/view?usp=sharing)

1. **Git Checkout**: Pull the latest code from the repository.
2. **Compile**: Compile the source code to ensure there are no syntax errors.
3. **Test**: Run unit and integration tests to validate the functionality.
4. **File System Scan**: Perform a security scan on the file system to check for vulnerabilities.
5. **SonarQube Analysis**: Analyze the code quality using SonarQube.
6. **Build**: Compile and package the application into a deployable artifact.
7. **Publish to Nexus**: Upload the built artifact to the Nexus repository.
8. **Build & Tag Docker Image**: Create a Docker image from the artifact and tag it with the version number.
9. **Docker Image Scan**: Scan the Docker image for vulnerabilities before deployment.
10. **Push Docker Image**: Push the Docker image to a container registry (e.g., Docker Hub, AWS ECR).
11. **Deploy to Kubernetes**: Deploy the Docker image to a Kubernetes cluster for staging or production

2. Draw the architecture diagram based on the functional requirements what AWS services will you use and why?

#### **High-Level Architecture:** [Draw link](https://drive.google.com/file/d/1uWApPhIXrcEnKoADQBtmMiv13zoy3Ymw/view?usp=sharing)

1. **User Devices**: End-user devices accessing the application.
2. **CloudFront**: CDN for fast content delivery.
3. **React Native Application**: Frontend application served to users.
4. **API Gateway**: Handles API requests with rate limiting.
5. **OAuth Provider**: Google/Facebook for user authentication.
6. **ALB**: Application Load Balancer managing HTTPS traffic.
7. **AWS WAF**: Web Application Firewall for security.
8. **ECS Cluster**: Containerized Java Spring-Boot microservices.
9. **RDS**: Relational Database Service for persistent storage.
10. **S3**: Storage for user profiles and settings.
11. **Redis**: In-memory data store for caching.
12. **Elasticsearch**: Search and analytics engine.

3. What tools are you going to use for monitoring and alerting purposes, and how they are going to be integrated with the infrastructure?

#### **Monitoring and Alerting Integration:**

1. **Prometheus**: Collects metrics from ECS services.
2. **Grafana**: Visualizes metrics from Prometheus.
3. **Alertmanager**: Routes alerts to the DevOps team.
4. **Email/Slack Notifications**: Alerts sent to communication channels.
5. **CloudWatch**: AWS service monitoring and logging.

4. Advise strategies or tools you are going to use to provide the desired security features

1. **OAuth Authentication**: Use OAuth 2.0 with Google/Facebook for secure user authentication.
2. **HTTPS Enforcement**: Use AWS Certificate Manager (ACM) for SSL/TLS certificates to secure data in transit.
3. **Rate Limiting**: Implement using AWS API Gateway to protect against brute force and DoS attacks.
4. **Geographical Blocking**: Use AWS WAF to block access from specific countries.
5. **Web Application Firewall (WAF)**: Protect against common web exploits using AWS WAF.
6. **IAM Roles & Policies**: Apply the principle of least privilege with AWS IAM for access control.
7. **Data Encryption**: Encrypt data at rest and in transit with AWS KMS and RDS/S3 encryption.
8. **Security Logging & Monitoring**: Use AWS CloudTrail and CloudWatch for logging and alerts
9. **Multi-Factor Authentication (MFA)**: Implement MFA for added account security using AWS MFA.
10. **Secrets Management**: Securely store credentials with AWS Secrets Manager.
11. **Network Security**: Use AWS VPC, Security Groups, and NACLs for network segmentation and security.

5. If you can suggest some tools or strategies for the management and maintenance of the data and infrastructure from a longer-term perspective

1. **Infrastructure as Code (IaC)**: Use Terraform or AWS CloudFormation for consistent and version-controlled infrastructure management.
2. **Automated Backups**: Implement AWS Backup and RDS automated backups to ensure data recoverability.
3. **Continuous Monitoring**: Use AWS CloudWatch, Prometheus, and Grafana for real-time monitoring and alerting.
4. **Auto-Scaling**: Utilize AWS Auto Scaling to adjust resources based on demand.
5. **Patch Management**: Automate patching with AWS Systems Manager Patch Manager
6. **Cost Management**: Track and optimize costs with AWS Cost Explorer and AWS Trusted Advisor.
7. **Centralized Log Management**: Aggregate logs with AWS CloudWatch Logs or the ELK Stack.
8. **DevOps Pipelines Maintenance**: Regularly update CI/CD pipelines with Jenkins or GitHub Actions.
9. **Governance and Policy Management**: Manage policies with AWS Organizations and IAM.