TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE

COMPUTER SCIENCE DEPARTMENT

DISTRIBUTED SYSTEMS

Laboratory Assignment 3

Chat System for Client Support

Student: Horvath Ariana-Cristine
Group: 30441
Teacher assistant: Oana-Andreea Marin

Table of Contents

1. Requirements 3
2. Design 3
2.1 gRPC part 3
2.2 Class Diagram Backend 6
3. CI/CD Deployment 7
3.1 Configuration Files 7
3.1.2 Backend 7
3.1.2 Frontend 9
3.1.3 gRPC Server (on Docker) 10

1. Requirements

Develop a chat system to offer support for the clients of the energy platform if they have questions related with their energy consumption. The chat system should allow communication between the clients and the administrator of the system.

1.1. Functional requirements:

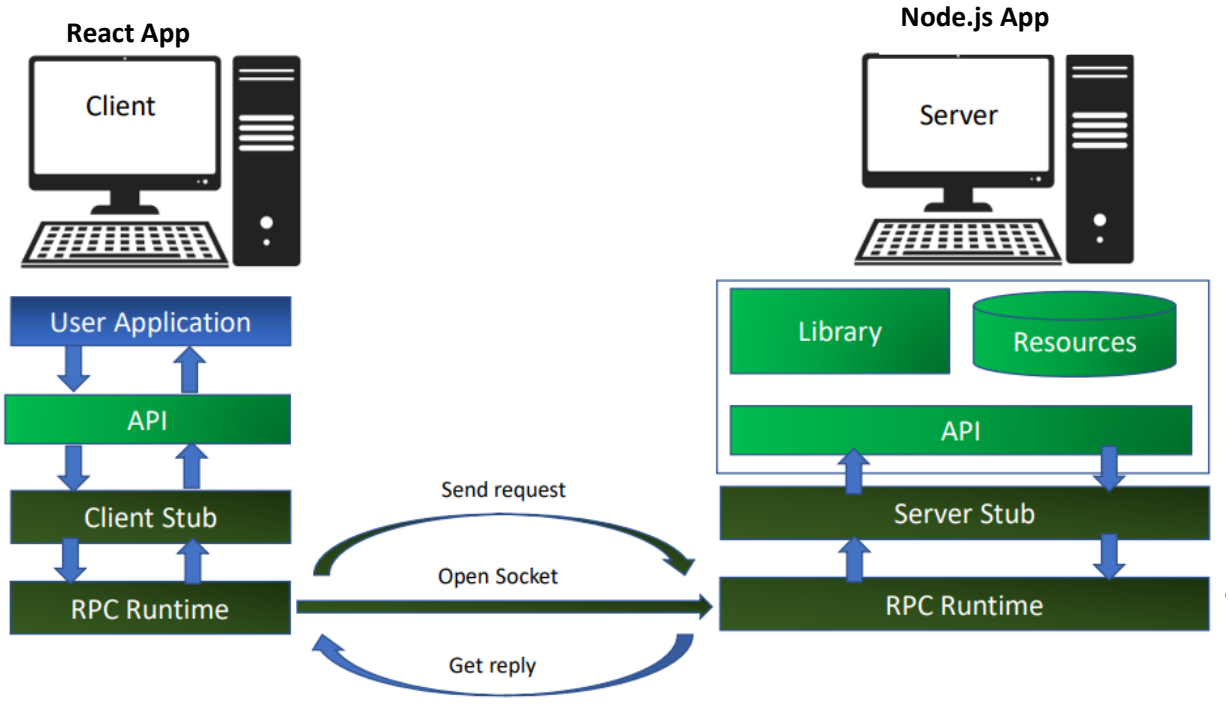
- The client application displays a chat box where clients can type messages.
- The message is sent asynchronously to the administrator, that receives the message together with the client identifier, being able to start a chat with the client.
- Messages can be sent back and forth between the client and the administrator during chat session.
- The administrator can chat with multiple clients at once.
- A notification is displayed for the user when the other user reads the message.
- A notification is displayed for the user (e.g., typing) while the user from the other end of communication types its message.

1.2. Implementation technologies:

- Choose one of the following technologies: any RPC framework supporting data streaming (for instance: gRPC)

2. Design

2.1 gRPC part



- gRPC is an inter-process communication technology that is used to execute remote sub-routines in a different address space. It uses the concept of message passing to signal a sub-routine residing in another system to execute. gRPC is awesome in that it creates a server service in a language and the service can be consumed from another platform in another language. This server service contains methods that can be called by the gRPC clients from anywhere in the world from any machine or platform.

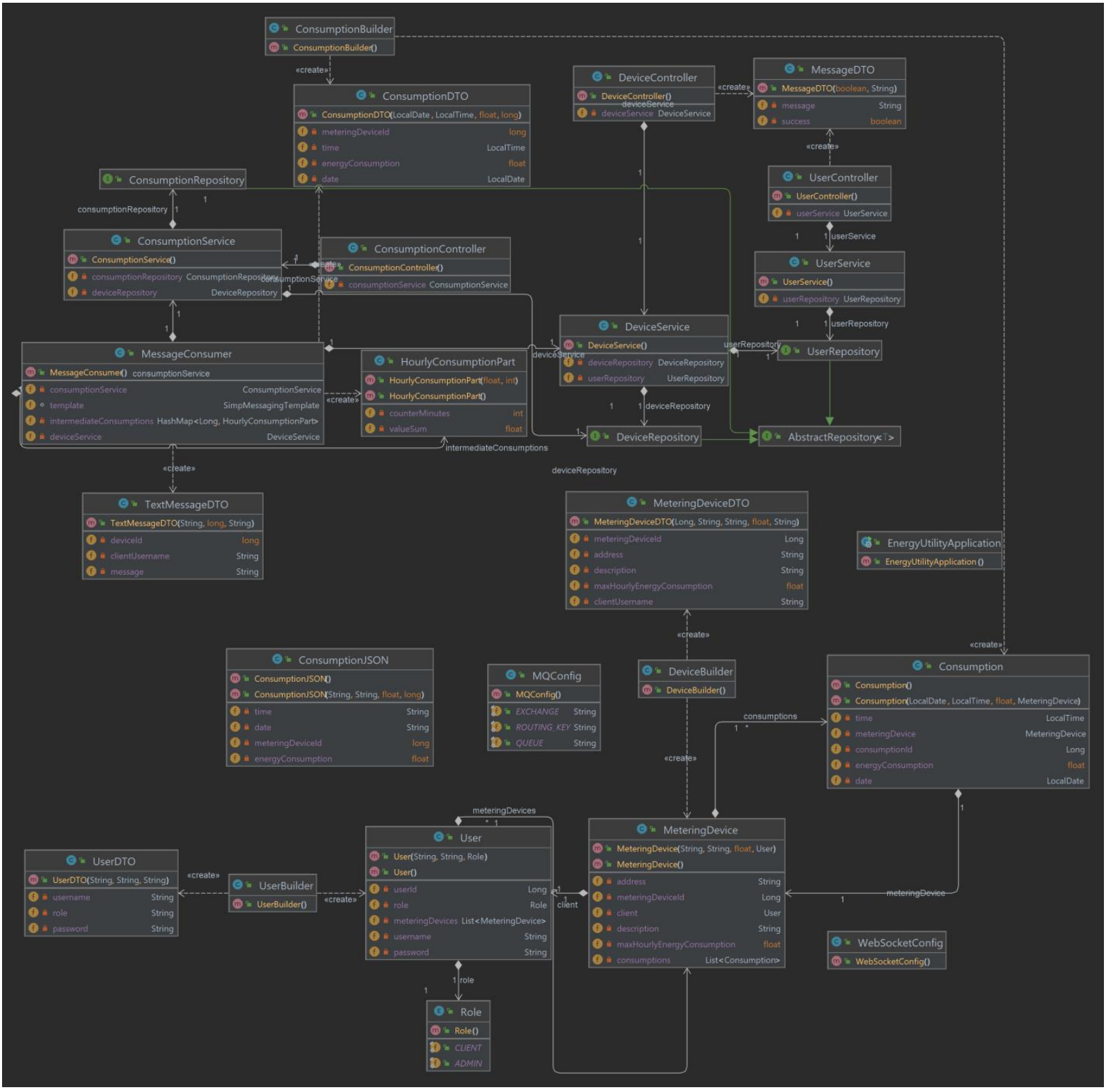
- gRPC has three components: Protocol Buffer, server, and client. Protocol Buffer is an open-source serialization tool built by Google. gRPC uses this to serialize the request and response message format between the server and the client. This is also where the service interface between the server and the client is defined. The service interface definition contains information on how your service can be consumed by consumers, what methods you allow the consumers to call remotely, what method parameters and message formats to use when invoking those methods, and so on.

  - *chat.proto* file

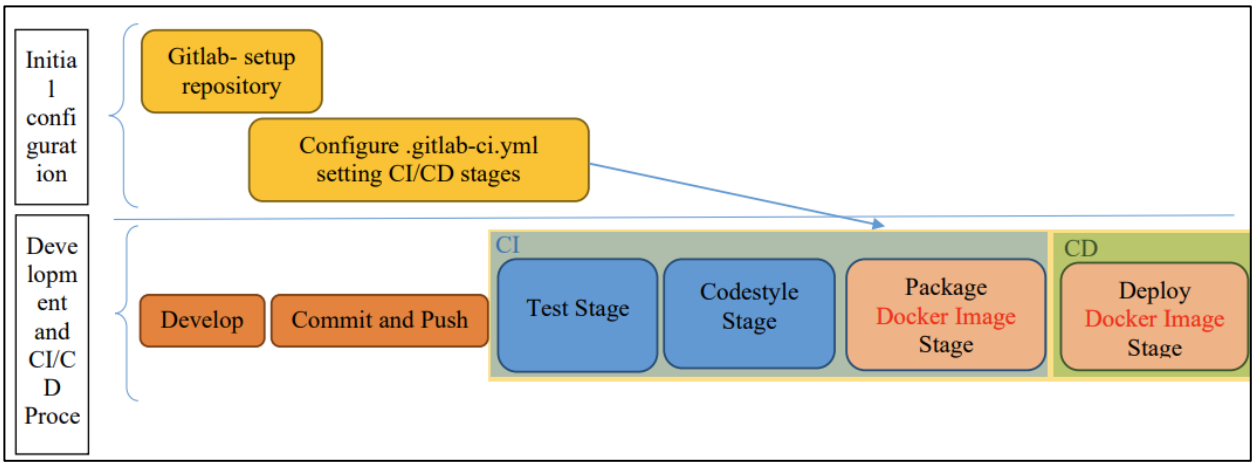
```
syntax = "proto3";

message ChatMessage {
   string from = 1;
   string msg = 2;
   string time = 3;
}

message User {
   string id = 1;
   string name = 2;
}

message Empty {}

message UserList {
   repeated User users = 1;
}

message JoinResponse {
   int32 error = 1;
   string msg = 2;
}

message ReceiveMsgRequest {
   string user = 1;
```

```
}

service ChatService {
   rpc join(User) returns (JoinResponse) {}
   rpc sendMsg(ChatMessage) returns (Empty) {}
   rpc receiveMsg(Empty) returns (stream ChatMessage) {}
   rpc getAllUsers(Empty) returns (UserList) {}
```

2.2 Class Diagram Backend



3. CI/CD Deployment



3.1 Configuration Files

3.1.2 Backend

- docker-compose.yml

```
version: "3.4"

services:
   api:
      image: "containerregistryarianahorvath30441.azurecr.io/arianahorvath30441backend:latest"
      domainname: "arianahorvath30441backend"
      ports:
        - 8080:8080
      environment:
        SPRING_RABBITMQ_HOST: rabbitmq
        DB_IP: demo-db
        RABBIT_IP: demo-rabbit
        DB_PORT: 3306
        DB_USER: root
        DB_PASSWORD: root123
        DB_DBNAME: energy_utility
      deploy:
        resources:
          reservations:
            cpus: '1'
            memory: 2G

   db:
      image: "containerregistryarianahorvath30441.azurecr.io/db:latest"
      environment:
```

```
      MYSQL_DATABASE: energy_utility
      MYSQL_ROOT_PASSWORD: root123
      MYSQL_HOST_AUTH_METHOD: trust
    domainname: "arianahorvath30441backend"
    ports:
      - 3306:3306
    deploy:
      resources:
        reservations:
          cpus: '1'
          memory: 2G
  rabbitmq:
    image: "containerregistryarianahorvath30441.azurecr.io/rabbitmq:latest"
    domainname: "arianahorvath30441backend"
    ports:
      - 15672:15672
      - 5672:5672
    deploy:
      resources:
        reservations:
          cpus: '1'
          memory: 2G
```

- azure-pipelines.yml

```
# Docker
# Build and push an image to Azure Container Registry
# https://docs.microsoft.com/azure/devops/pipelines/languages/docker

trigger:
- master

resources:
- repo: self

variables:
  # Container registry service connection established during pipeline creation
  dockerRegistryServiceConnection: '5e9f645c-6548-46fc-bb7f-25853a03db67'
  imageRepository: 'arianahorvath30441backend'
  containerRegistry: 'containerregistryarianahorvath30441.azurecr.io'
  dockerfilePath: '$(Build.SourcesDirectory)/energyUtility/Dockerfile'
  tag: 'latest'

  # Agent VM image name
  vmImageName: 'ubuntu-latest'

stages:
- stage: Build
  displayName: Build and push stage
```

```
jobs:
- job: Build
  displayName: Build
  pool: 'local'
  steps:
  - task: Docker@2
    displayName: Build and push an image to container registry
    inputs:
      command: buildAndPush
      repository: $(imageRepository)
      dockerfile: $(dockerfilePath)
      containerRegistry: $(dockerRegistryServiceConnection)
      tags: |
        $(tag)
```

3.1.2 Frontend

- docker-compose.yml

```
version: "3.4"

services:

  react:
    image: "containerregistryarianahorvath30441.azurecr.io/arianahorvath30441frontend:latest"
    domainname: "arianahorvath30441frontend"
    ports:
      - 80:80
    deploy:
        resources:
          reservations:
            cpus: '1'
            memory: 2G
```

- azure-pipelines.yml

```
# Docker
# Build and push an image to Azure Container Registry
# https://docs.microsoft.com/azure/devops/pipelines/languages/docker

trigger:
- master

resources:
```

```
- repo: self

variables:
  # Container registry service connection established during pipeline creation
  dockerRegistryServiceConnection: 'c2d2c1a7-db5d-49c1-a26d-982391bf363e'
  imageRepository: 'arianahorvath30441frontend'
  containerRegistry: 'containerregistryarianahorvath30441.azurecr.io'
  dockerfilePath: '$(Build.SourcesDirectory)/Dockerfile'
  tag: 'latest'

  # Agent VM image name
  vmImageName: 'ubuntu-latest'

stages:
- stage: Build
  displayName: Build and push stage
  jobs:
  - job: Build
    displayName: Build
    pool: 'local'
    steps:
    - task: Docker@2
      displayName: Build and push an image to container registry
      inputs:
        command: buildAndPush
        repository: $(imageRepository)
        dockerfile: $(dockerfilePath)
        containerRegistry: $(dockerRegistryServiceConnection)
        tags: |
          $(tag)
```

3.1.3 gRPC Server (on Docker)

- Dockerfile

```
FROM node:16

# Create app directory
WORKDIR /usr/src/app2

COPY package*.json ./
```

```
RUN npm install

# Bundle app source
COPY . .

EXPOSE 9090
CMD [ "node", "server.js" ]
```

- docker-compose.yml

```
version: '3'

services:

  react:
    image: grpcnodeserver
    ports:
      - "9090:9090"
```

3.1.4 Envoy proxy

- Dockefile

```
FROM envoyproxy/envoy-dev:latest
COPY envoy.yaml /etc/envoy/envoy.yaml
RUN chmod go+r /etc/envoy/envoy.yaml
```

- envoy.yaml

```
admin:
  access_log_path: /tmp/admin_access.log
  address:
    socket_address: { address: 0.0.0.0, port_value: 9901 }

static_resources:
  listeners:
    - name: listener_0
      address:
        socket_address: { address: 0.0.0.0, port_value: 8080 }
      filter_chains:
        - filters:
```

```
          - name: envoy.filters.network.http_connection_manager
            typed_config:
              "@type":
type.googleapis.com/envoy.extensions.filters.network.http_connection_manager.v3.HttpConnectionMana
ger
              codec_type: auto
              stat_prefix: ingress_http
              stream_idle_timeout: 0s
              route_config:
                name: local_route
                virtual_hosts:
                  - name: local_service
                    domains: ["*"]
                    routes:
                      - match: { prefix: "/" }
                        route:
                          cluster: chat_service
                          max_grpc_timeout: 0s
                          max_stream_duration:
                            grpc_timeout_header_max: 0s
                    cors:
                      allow_origin_string_match:
                        - prefix: "*"
                      allow_methods: GET, PUT, DELETE, POST, OPTIONS
                      allow_headers: keep-alive,user-agent,cache-control,content-type,content-transfer-
encoding,custom-header-1,x-accept-content-transfer-encoding,x-accept-response-streaming,x-user-
agent,x-grpc-web,grpc-timeout
                      max_age: "1728000"
                      expose_headers: custom-header-1,grpc-status,grpc-message
              http_filters:
                - name: envoy.filters.http.grpc_web
                  typed_config:
                    "@type": type.googleapis.com/envoy.extensions.filters.http.grpc_web.v3.GrpcWeb
                - name: envoy.filters.http.cors
                  typed_config:
                    "@type": type.googleapis.com/envoy.extensions.filters.http.cors.v3.Cors
                - name: envoy.filters.http.router
                  typed_config:
                    "@type": type.googleapis.com/envoy.extensions.filters.http.router.v3.Router
  clusters:
    - name: chat_service
      connect_timeout: 0.25s
      type: logical_dns
      http2_protocol_options: {}
```

```
    lb_policy: round_robin
    # win/mac hosts: Use address: host.docker.internal instead of address: localhost in the line below
    load_assignment:
      cluster_name: cluster_0
      endpoints:
        - lb_endpoints:
            - endpoint:
                address:
                  socket_address:
                    address: host.docker.internal
                    port_value: 9090
```