



UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
LAB. SOFTWARE AVANZADO
AUX. DIEGO MOLINA

DOCUMENTACIÓN FASE 2

JONATHAN MARCOS VALIENTE GONZÁLEZ - 201931581
DENILSON FLORENTÍN DE LEÓN AGUILAR - 201830313

3 DE ABRIL DE 2024, GUATEMALA

ÍNDICE

DOCUMENTACIÓN DE REQUISITOS

REQUISITOS

Requerimientos Funcionales

Requerimientos no Funcionales

FUNCIONALIDADES

Creación de Tickets

Seguimiento de Tickets

Resolución de Tickets

Gestión de Usuarios

Informes y Análisis

Funcionalidades Adicionales

TIPOS DE USUARIOS Y PERMISOS

CLIENTE

AGENTE

ADMINISTRADOR

SPRINT PLANNING

DIAGRAMA DE ARQUITECTURA

DIAGRAMA DE ARQUITECTURA DE ALTO NIVEL

CONTRATOS DE MICROSERVICIOS

Get Tickets By Email

Descripción:

Endpoint:

Parámetros:

Respuesta:

json

Get Tickets By TicketNumber

Descripción:

Endpoint:

Parámetros:

Respuesta:

Descripción de la Respuesta:

Get Unsolved Tickets

Descripción:

Endpoint:

Encabezado:

Respuesta:

Descripción de la Respuesta:

Get Agent Tickets

Descripción:

Endpoint:

Encabezado:

Respuesta:

Descripción de la Respuesta:

Login

Descripción:

Endpoint:

Cuerpo de la Solicitud (Body):

Respuesta:

Descripción de la Respuesta:

Register

Descripción:

Endpoint:

Cuerpo de la Solicitud (Body):

Respuesta:

Descripción de la Respuesta:

Register Protected

Descripción:

Endpoint:

Encabezado:

Cuerpo de la Solicitud (Body):

Respuesta:

Descripción de la Respuesta:

Assign Agent

Descripción:

Endpoint:

Encabezado:

Cuerpo de la Solicitud (Body):

Respuesta:

Descripción de la Respuesta:

create-ticket

Descripción:

Endpoint:

Encabezado:

Cuerpo de la Solicitud (Body):

Respuesta:

Descripción de la Respuesta:

Get Logs

Descripción:

Endpoint:

Cuerpo de la Solicitud (Body):

Respuesta:

Descripción de la Respuesta:

upload

Descripción:

Endpoint:

Cuerpo de la Solicitud (Body):

[Respuesta:](#)

[Descripción de la Respuesta:](#)

[Validate JWT](#)

[Descripción:](#)

[Endpoint:](#)

[Encabezado:](#)

[Respuesta:](#)

[Descripción de la Respuesta:](#)

[saveMessage](#)

[Descripción:](#)

[Endpoint:](#)

[Cuerpo de la Solicitud \(Body\):](#)

[Respuesta:](#)

[Descripción de la Respuesta:](#)

[changeBlock](#)

[Descripción:](#)

[Endpoint:](#)

[Cuerpo de la Solicitud \(Body\):](#)

[Respuesta:](#)

[Descripción de la Respuesta:](#)

[changeBlock](#)

[Descripción:](#)

[Endpoint:](#)

[Cuerpo de la Solicitud \(Body\):](#)

[Respuesta:](#)

[Descripción de la Respuesta:](#)

[getDataReport](#)

[Descripción:](#)

[Endpoint:](#)

[Cuerpo de la Solicitud \(Body\):](#)

[Respuesta:](#)

[Descripción de la Respuesta:](#)

[DEFINIR CI-CD](#)

[developCI.yaml](#)

[Etapas 1: build_stage](#)

[Trabajo 1: build_stage](#)

[Etapas 2: delivery-stage](#)

[Trabajo 1: delivery-stage](#)

[featureCI.yaml](#)

[Etapas 1: build_stage](#)

[Trabajo 1: build_stage](#)

[Etapas 2: delivery-stage](#)

[Trabajo 1: delivery-stage](#)

[mainCI.yaml](#)

[Etapa 1: delivery-stage](#)

[Trabajo 1: delivery-stage](#)

[releaseCI-yaml](#)

[Etapa 1: build_stage](#)

[Trabajo 1: build_stage](#)

[Etapa 2: delivery-stage](#)

[Trabajo 1: delivery-stage](#)

[Manual de Instalación de Runner](#)

[Requisitos Previos](#)

[Pasos de Instalación](#)

[1. Descargar el Código Fuente del Runner](#)

[2. Preparar el Entorno](#)

[3. Configurar el Runner](#)

[4. Registrar el Runner](#)

[5. Configurar y Ejecutar los Trabajos](#)

[6. Verificar el Estado del Runner](#)

DOCUMENTACIÓN DE REQUISITOS

REQUISITOS

Requerimientos Funcionales

- El sistema debe contar con un apartado de creación de tickets, dónde el usuario especifica parámetros como datos de contacto, archivos de evidencia, descripción del problema y tipo de problema.
- El sistema cuenta con un apartado para buscar los tickets pendientes en base a los datos de contacto del cliente.
- El usuario tiene permitido ver los tickets que le pertenecen, rastrear el estado del ticket y mandar mensajes al agente que está resolviendo ese ticket.
- Un agente puede rastrear tickets, asignarse el ticket, cambiar el estado del ticket, cambiar la prioridad del ticket, resolver y chatear con el usuario del ticket.
- El sistema cuenta con registro de usuarios agentes y administradores, dónde solo los administradores pueden agregar a otros usuarios agentes o administradores.
- Los administradores pueden ver informes acerca de la resolución de tickets, importar a una herramienta de CRM por medio de un csv.
- Tras la resolución de un ticket el cliente llena una encuesta de retroalimentación.

Requerimientos no Funcionales

- El sistema debe funcionar con herramientas en la nube de Google.
- Manejar seguridad web en el manejo de los usuarios agente y administrador.
- El sistema debe ser escalable.

FUNCIONALIDADES

Creación de Tickets

```
@PostMapping("/create-ticket")
public ResponseEntity<?> createTicket(@Validated @RequestBody String json){
    try{
        JsonNode jsonNode = new JsonOptions().parseStringJsonNode(json);
        //TicketType ticketType =
ticketTypeService.getByType(jsonNode.get("ticketType").asText());
        TicketType ticketType =
ticketTypeService.findById(jsonNode.get("ticketType").asInt());
        //TicketPriority ticketPriority =
ticketPriorityService.getByPriority(jsonNode.get("ticketPriority").asText());
        TicketPriority ticketPriority =
ticketPriorityService.findById(jsonNode.get("ticketPriority").asInt());
        // Obtener la lista de URLs del campo "urls"
        JsonNode urlsNode = jsonNode.get("files");
        List<String> urlsList = new ArrayList<>();
        for (JsonNode urlNode : urlsNode) {
```

```

        urlsList.add(urlNode.asText());
    }

    System.out.println("createTicket:"+ticketType.getType());
    Ticket ticket = new Ticket(
        jsonNode.get("email").asText(),
        jsonNode.get("name").asText(),
        jsonNode.get("lastName").asText(),
        jsonNode.get("phone").asText(),
        jsonNode.get("description").asText(),
        ticketType,
        ticketPriority,
        null
    );
    //Creacion del ticket
    ticket = ticketService.createTicket(ticket);
    //Create Elements
    for(String url:urlsList){
        ticketElementService.saveElement(ticket,url);
    }
    //Creacion del tracking
    State_of_Ticket stateOfTicket =
stateOfTicketService.getReferencedById(1);
    TicketTracking ticketTracking =
ticketTrackingService.createTrack(ticket,stateOfTicket);

    //Creacion del historial
    historyOfCommunicationService.saveLog(
        ticketTracking,
        "Se creo el ticket de
tipo:"+ticket.getTicketType().getType()+"", de
prioridad:"+ticket.getPriority().getPriority()
    );

    return ResponseEntity.status(HttpStatus.CREATED).build();
} catch (Exception ex) {
    ex.printStackTrace();
    System.out.println("Error: "+ex.getMessage());
    return new ResponseEntity<>(new Message("Error en creacion de tickets:
"+ex.getMessage()), HttpStatus.BAD_REQUEST);
}
}

```

Seguimiento de Tickets

```
@GetMapping("/getTickets")
```

```

public ResponseEntity<?> getTickets(@Validated @RequestParam("email") String
email){
    try{
        List<Map<String, Object>> listTokens =
ticketService.getTicketsEmail(email);
        List<Map<String, Object>> modifiedList = new ArrayList<>();
        for (Map<String, Object> map : listTokens) {
            Map<String, Object> modifiedMap = new HashMap<>(map); // Crear una
copia modificable del mapa actual
            String ticket = modifiedMap.get("ticketNumber").toString();
            int ticketNumber = Integer.parseInt(ticket);
            List<String> elements =
ticketService.getTicketsElements(ticketNumber);
            modifiedMap.put("files", elements);
            modifiedList.add(modifiedMap); // Agregar el mapa modificado a la
nueva lista
        }

        Map<String, Object> provider = new HashMap<>();
        provider.put("tickets", modifiedList);
        return new ResponseEntity<>(provider, HttpStatus.OK);
    } catch (Exception ex) {
        ex.printStackTrace();
        return new ResponseEntity<>(new Message("Error obtencion de tickets:
"+ex.getMessage()), HttpStatus.BAD_REQUEST);
    }
}

```

Resolución de Tickets

```

@PostMapping("/change-priority")
public ResponseEntity<?> changeTicketPriority(@Validated @RequestBody String
json){
    try{
        JsonNode jsonNode = new JsonOptions().parseStringJsonNode(json);
        // Find the new Priority
        TicketPriority priority = ticketPriorityService.getByPriority(
            jsonNode.get("priority").asText()
        );
        //Find the ticket
        Ticket ticket = ticketService.getReferenceById(
            jsonNode.get("ticketNumber").asInt()
        );

        ticket.setPriority(priority);
        ticketService.createTicket(ticket);

        TicketTracking ticketTracking = ticketTrackingService.findById(
            jsonNode.get("ticketNumber").asInt()
        );
        ticketTracking.setDateLastUpdation(new Date());
        ticketTracking = ticketTrackingService.saveTicket(ticketTracking);
    }
}

```



```

        historyOfCommunicationService.saveLog(
            ticketTracking,
            "Se cambio la prioridad del ticket a
"+jsonNode.get("priority").asText()
        );
        return ResponseEntity.status(HttpStatus.CREATED).build();
    } catch (Exception ex) {
        return new ResponseEntity<>(new Message("Error en cambio de prioridad
en ticket: "+ex.getMessage()), HttpStatus.BAD_REQUEST);
    }
}

@PostMapping("/change-state")
public ResponseEntity<?> changeTicketState(@Validated @RequestBody String
json,

                                           @RequestHeader("Authorization")
String authorizationHeader) {
    try {
        if (!jwtChecker.initCheck(authorizationHeader)) {
            return new ResponseEntity<>(new Message("Sesion invalida"),
HttpStatus.FORBIDDEN);
        }
        JsonNode jsonNode = new JsonOptions().parseStringJsonNode(json);
        // Find the new Priority
        State_of_Ticket stateOfTicket = stateOfTicketService.getByState(
            jsonNode.get("state").asText()
        );
        //Find the ticket
        TicketTracking ticketTracking = ticketTrackingService.findById(
            jsonNode.get("ticketNumber").asInt()
        );
        ticketTracking.setState(stateOfTicket);
        ticketTracking.setDateLastUpdation(new Date());
        ticketTracking=ticketTrackingService.saveTicket(ticketTracking);
        historyOfCommunicationService.saveLog(
            ticketTracking,
            "Se cambio el estado del ticket a
"+(jsonNode.get("state").asText())
        );
        return ResponseEntity.status(HttpStatus.OK).build();
    } catch (Exception ex) {
        return new ResponseEntity<>(new Message("Error en cambio de estado del
ticket: "+ex.getMessage()), HttpStatus.BAD_REQUEST);
    }
}

@PostMapping("/solve")
public ResponseEntity<?> solve(@Validated @RequestBody String json) {
    try {
        JsonNode jsonNode = new JsonOptions().parseStringJsonNode(json);
        TicketTracking ticketTracking = ticketTrackingService.findById(
            jsonNode.get("ticketNumber").asInt()
        );
        ticketTracking.setProblemSolved(true);
    }
}

```

```

        ticketTracking.setDateLastUpdate(new Date());
        ticketTracking=ticketTrackingService.saveTicket(ticketTracking);
        historyOfCommunicationService.saveLog(
            ticketTracking,
            "Se resolvió el ticket"
        );
        return ResponseEntity.status(HttpStatus.OK).build();
    }catch(Exception ex){
        return new ResponseEntity<>(new Message("Error al resolver el ticket: "+ex.getMessage()), HttpStatus.BAD_REQUEST);
    }
}

@PostMapping("/trackLogs")
public ResponseEntity<?> getLogs(@Validated @RequestBody String json){
    try{
        JsonNode jsonNode = new JsonOptions().parseStringJsonNode(json);
        int ticketNumber = jsonNode.get("ticketNumber").asInt();
        List<History_of_CommunicationDTO> list =
historyOfCommunicationService.getLogs(ticketNumber);
        return new ResponseEntity<>(list, HttpStatus.OK);
    }catch(Exception ex){
        return new ResponseEntity<>(new Message("Error, no se pudieron obtener los logs: "+ex.getMessage()), HttpStatus.BAD_REQUEST);
    }
}

@PostMapping("/assign-agent")
public ResponseEntity<?> assignAgent(@Validated @RequestBody String json,
                                     @RequestHeader("Authorization") String
authorizationHeader){
    try{
        if (!jwtChecker.initCheck(authorizationHeader)) {
            // Registra una sesión inválida
            return new ResponseEntity<>(new Message("Sesión inválida"),
HttpStatus.FORBIDDEN);
        }
        JsonNode jsonNode = new JsonOptions().parseStringJsonNode(json);
        String agentUsername = jwtChecker.getSubject(authorizationHeader);
        User agent = userService.getUser(agentUsername);
        TicketTracking ticketTracking =
ticketTrackingService.findById(jsonNode.get("ticketNumber").asInt());
        ticketTracking.setAgent(agent);
        ticketTracking = ticketTrackingService.saveTicket(ticketTracking);
        historyOfCommunicationService.saveLog(
            ticketTracking,
            "El ticket se asignó a "+ticketTracking.getAgent().getName()+
            " "+ticketTracking.getAgent().getLastName()+" usuario:
"+ticketTracking.getAgent().getUsername()
        );
        return ResponseEntity.status(HttpStatus.CREATED).build();
    }catch(Exception ex){

```

```

        return new ResponseEntity<>(new Message("Error en cambio de agente en
ticket: "+ex.getMessage()), HttpStatus.BAD_REQUEST);
    }
}

@PostMapping("/visualize")
public ResponseEntity<?> visualize(@Validated @RequestBody String json){
    try{
        JsonNode jsonNode = new JsonOptions().parseStringJsonNode(json);
        int ticketNumber = jsonNode.get("ticketNumber").asInt();
        TicketTracking ticketTracking =
ticketTrackingService.findById(ticketNumber);
        return new ResponseEntity<>(ticketTracking, HttpStatus.OK);
    }catch(Exception ex){
        return new ResponseEntity<>(new Message("Error en obtencion de datos
del cliente "+ex.getMessage()), HttpStatus.BAD_REQUEST);
    }
}

@PostMapping("/visualizeTicket")
public ResponseEntity<?> viewTicket(@Validated @RequestBody String json,
                                     @RequestHeader("Authorization") String
authorizationHeader){
    try{
        //Validacion del JWT
        jwtChecker.checkJWT(authorizationHeader);

        JsonNode jsonNode = new JsonOptions().parseStringJsonNode(json);
        int ticketNumber = jsonNode.get("ticketNumber").asInt();
        //Obtener El track del ticket y el historial
        TicketTracking ticketTracking =
ticketTrackingService.findById(ticketNumber);
        List<History_of_CommunicationDTO> historyOfCommunication =
historyOfCommunicationService.getLogs(ticketNumber);

        //Armar la respuesta
        Map<String, Object> response = new HashMap<>();
        response.put("ticket", ticketTracking);
        response.put("history", historyOfCommunication);
        return new ResponseEntity<>(response, HttpStatus.OK);
    }catch(Exception ex){
        return new ResponseEntity<>(new Message("Error en obtencion de datos
del cliente "+ex.getMessage()), HttpStatus.BAD_REQUEST);
    }
}

```

Gestión de Usuarios

```

@PostMapping("/register")
public ResponseEntity<?> registerUser(@Validated @RequestBody String json) {
    try{
        return register(json);
    }
}

```

```

    }catch(Exception ex){
        ex.printStackTrace();
        return new ResponseEntity<>(new Message("Error en registro:
"+ex.getMessage()), HttpStatus.BAD_REQUEST);
    }
}

/**
 * Provide
 * {
 *     username:
 *     password:
 *     userType: [as Role or Type] [options: cliente, agente, administrador]
 *     name:
 *     lastName:
 *     phone:
 * }
 * @param json
 * @return
 */
@PostMapping("/registerSpecial")
public ResponseEntity<?> registerUserProtected(@Validated @RequestBody String
json,
                                                @RequestHeader("Authorization")
String authorizationHeader) {
    try{
        if (!jwtChecker.initCheck(authorizationHeader)) {
            // Registra una sesión inválida
            return new ResponseEntity<>(new Message("Sesión inválida"),
HttpStatus.FORBIDDEN);
        }

        return register(json);
    }catch(Exception ex){
        ex.printStackTrace();
        return new ResponseEntity<>(new Message("Error en registro:
"+ex.getMessage()), HttpStatus.BAD_REQUEST);
    }
}

private ResponseEntity<?> register(String json) throws TicketException,
NoSuchAlgorithmException {
    JsonNode jsonNode = new JsonOptions().parseStringJsonNode(json);
    String passwordEncrypted =
encrypter.encrypt(jsonNode.get("password").asText());
    UserType userType =
userService.getByType(jsonNode.get("userType").asText());

    if(userService.usernameExists(jsonNode.get("username").asText())){
        throw new TicketException("Usuario ya existe");
    }

    User user = new User(

```

```

        jsonNode.get("username").asText(),
        passwordEncrypted,
        jsonNode.get("name").asText(),
        jsonNode.get("lastName").asText(),
        jsonNode.get("phone").asText(),
        userType
    );
    //Registramos el usuario
    userService.registerUser(user);
    return ResponseEntity.status(HttpStatus.CREATED).build();
}

/**
 * Provide:{
 *     username: String
 *     password: String
 * }
 *
 * Returns: {
 *     token: String
 *     user_type: number
 * }
 * @param json
 * @return
 */
@PostMapping("/login")
public ResponseEntity<?> login(@Validated @RequestBody String json){
    try{
        JsonNode jsonNode = new JsonOptions().parseStringJsonNode(json);
        String username = jsonNode.get("username").asText();
        String passwordEncrypted =
encrypter.encrypt(jsonNode.get("password").asText());
        User user = userService.getUser(username);
        if(user.getUsername().equalsIgnoreCase(username) &&
user.getPassword().equals(passwordEncrypted)){
            //handle login
            Map<String, Object> map = new HashMap<>();
            map.put("username", username);
            Map<String, Object> response = new HashMap<>();
            response.put("jwt", jwtProvider.generateToken(username, map));
            response.put("user_type", user.getUserType());
            return new ResponseEntity<>(response, HttpStatus.OK);
        }else{
            throw new TicketException("El usuario o la contraseña son
incorrectos");
        }
    }catch(Exception ex){
        return new ResponseEntity<>(new Message("Error en Inicio de sesion:
"+ex.getMessage()), HttpStatus.NOT_ACCEPTABLE);
    }
}

@PostMapping("/validate")

```

```

public ResponseEntity<?> validateJWT(@RequestHeader("Authorization") String
authorizationHeader) {
    try{
        jwtChecker.checkJWT(authorizationHeader);
        System.out.println("JWT Valido");
        return new ResponseEntity<>(new Message("Autorizado"), HttpStatus.OK);
    }catch(Exception ex) {
        System.out.println("Sesion invalida");
        return new ResponseEntity<>(new Message("Sesion invalida:
")+ex.getMessage(), HttpStatus.FORBIDDEN);
    }
}

```

Informes y Análisis

```

@PostMapping("/getSurveys")
public ResponseEntity<?> getSurveys(@RequestHeader("Authorization") String
authorizationHeader) {
    try{
        jwtChecker.checkJWT(authorizationHeader);
        return new ResponseEntity<>(surveyService.getSurveys(),
HttpStatus.OK);
    }catch(Exception ex) {
        return new ResponseEntity<>(new Message("Error al obtener las
encuestas"), HttpStatus.BAD_REQUEST);
    }
}

```

Funcionalidades Adicionales

```

@PostMapping("/saveSurvey")
public ResponseEntity<?> responseSurver(@Validated @RequestBody String json) {
    try{
        JsonNode jsonNode = new JsonOptions().parseStringJsonNode(json);
        int ticketNumber = jsonNode.get("ticketNumber").asInt();
        Ticket ticket = ticketService.getReferenceById(ticketNumber);
        int satisfaction = jsonNode.get("satisfaction").asInt();
        int timeService = jsonNode.get("timeService").asInt();
        int qualityService = jsonNode.get("qualityService").asInt();
        Survey survey = new Survey(
            ticket,
            satisfaction,
            timeService,
            qualityService
        );
        surveyService.saveSurvey(survey);
        return ResponseEntity.status(HttpStatus.OK).build();
    }catch(Exception ex) {
        return new ResponseEntity<>(new Message("Error en la respuesta de la
encuesta :"+ ex.getMessage()), HttpStatus.BAD_REQUEST);
    }
}

```

```
}  
}  
  
@PostMapping("/getCSV")  
public ResponseEntity<?> getCSV(@RequestHeader("Authorization") String  
authorizationHeader) {  
    try {  
        jwtChecker.checkJWT(authorizationHeader);  
        return new ResponseEntity<>(surveyService.getCsv(), HttpStatus.OK);  
    } catch (Exception ex) {  
        return new ResponseEntity<>(new Message("Error en obtener el csv"),  
HttpStatus.BAD_REQUEST);  
    }  
}
```

TIPOS DE USUARIOS Y PERMISOS

CLIENTE

- Crear tickets
- Rastrear Tickets
- Ver tutoriales
- Mensajear con agente atendiente del ticket
- Resolver encuestas de retroalimentación.

AGENTE


- Asignar agente a un ticket
- Resolver ticket
- Cambiar estado del ticket
- Cambiar prioridad del ticket
- Ver detalles del ticket
- Mensajear con cliente solicitante del ticket

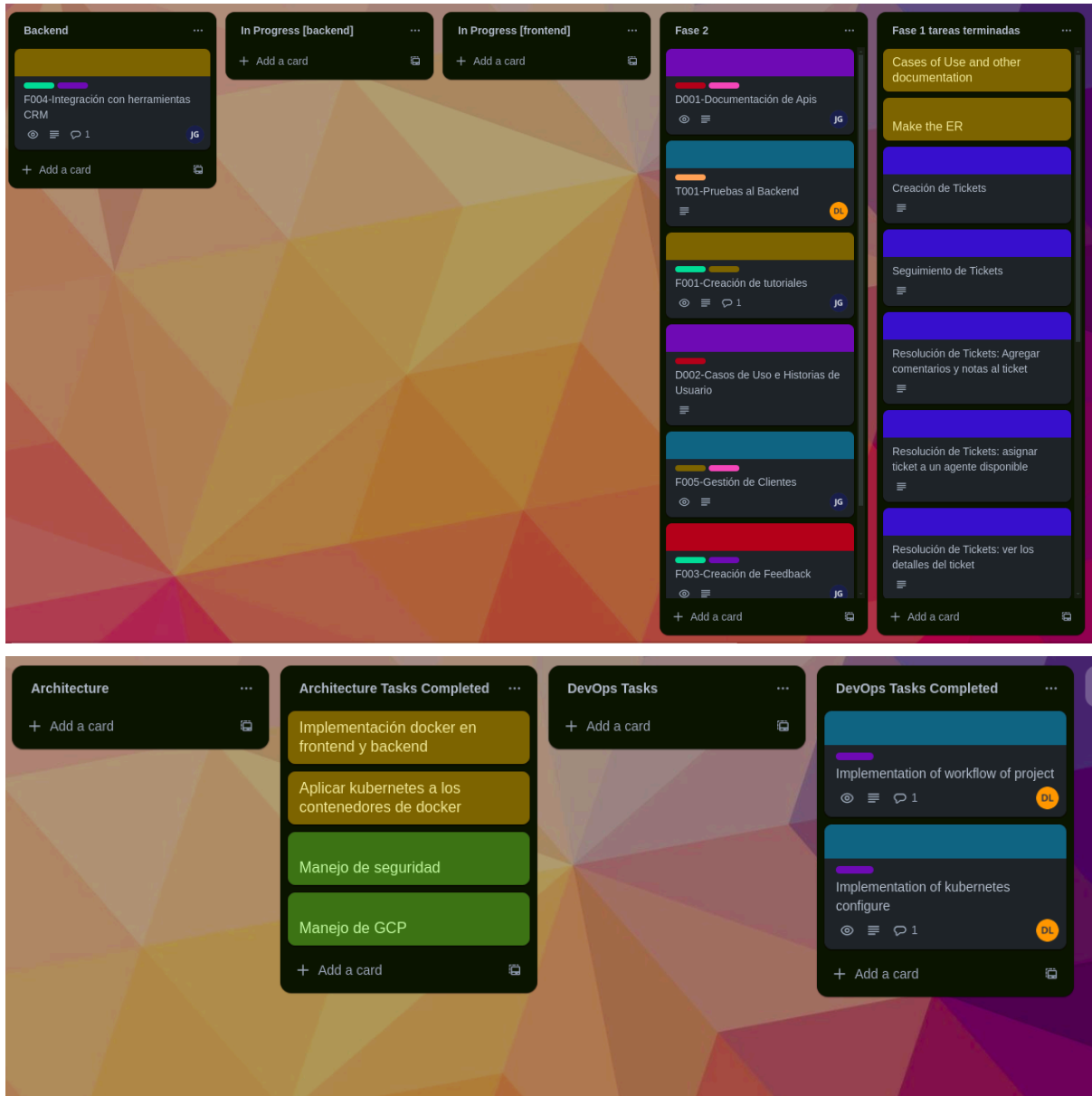
ADMINISTRADOR

- Crear usuarios agente y administrador
- Generar informes de retroalimentación

Generar archivos csv para conexión con herramientas CRM

SPRINT PLANNING

Video:  Spring Planning.mkv



<https://trello.com/b/WwScNTLN/proyectosa>

DIAGRAMA DE ARQUITECTURA

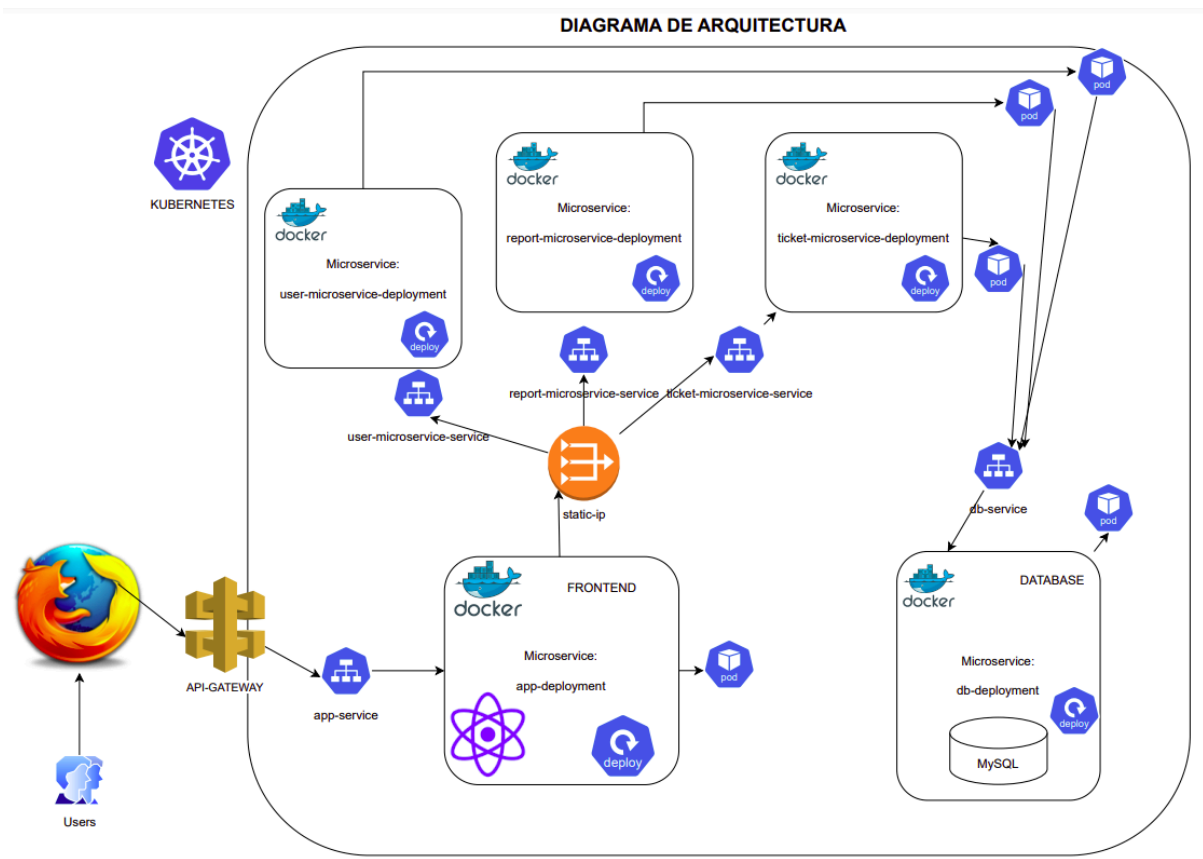
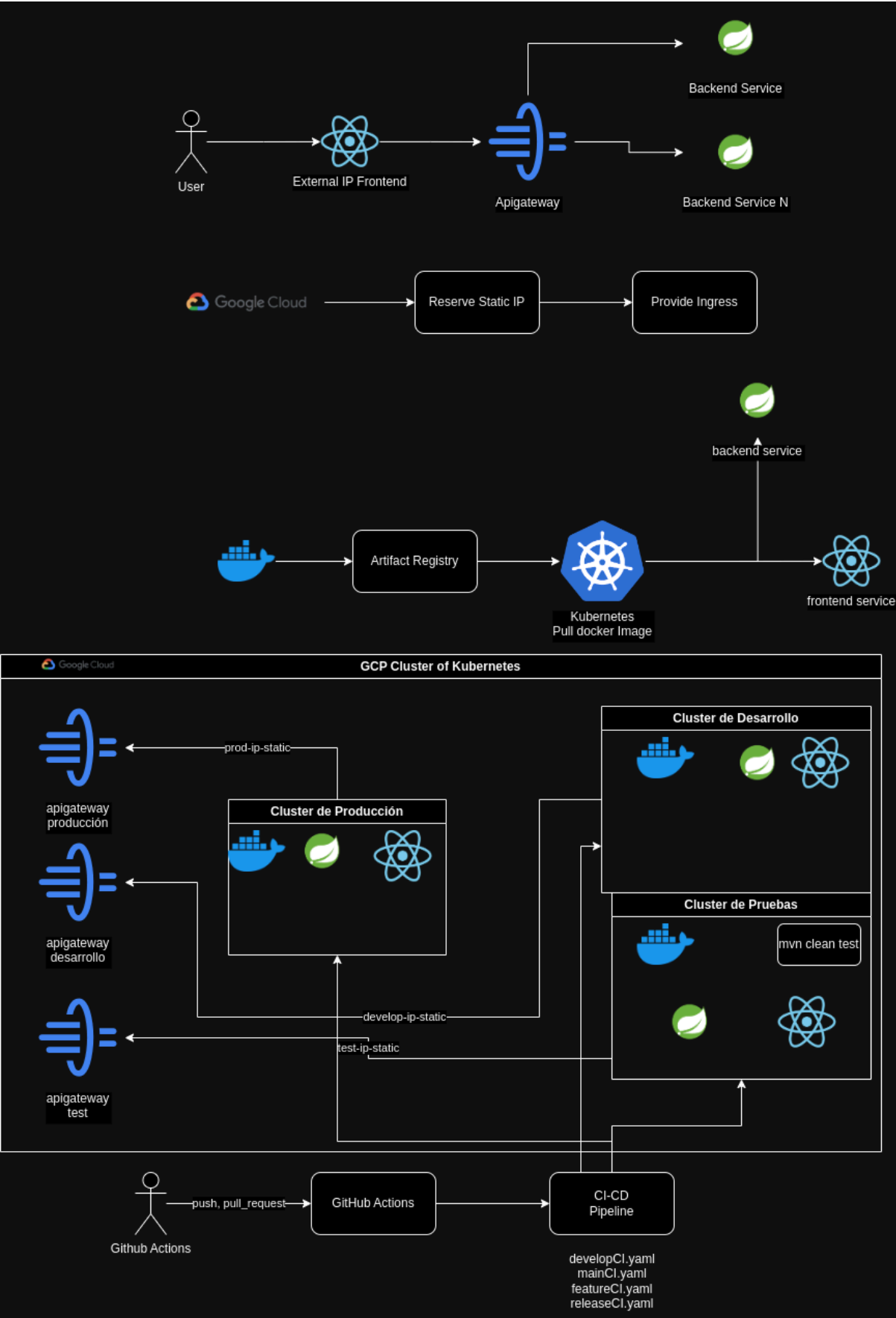


DIAGRAMA DE ARQUITECTURA DE ALTO NIVEL



CONTRATOS DE MICROSERVICIOS

Get Tickets By Email

Descripción:

Este microservicio permite a los usuarios obtener una lista de tickets asociados con una dirección de correo electrónico específica.

Endpoint:

- **URL:** <http://127.0.0.1:33065/api/ticket/getTickets>
- **Método HTTP:** GET

Parámetros:

- **email** (query parameter): Dirección de correo electrónico del usuario para filtrar los tickets.

Respuesta:

json

```
{
  "tickets": [
    {
      "owner": null,
      "lastName": "Perez",
      "ticketNumber": 48,
      "agent": {
        "idUser": 14,
        "username": "agente",
        "name": "Uriel",
        "lastName": "Gutierrez",
        "phone": "1234-1234",
        "userType": 2
      },
      "problemSolved": 0,
      "description": "Descripcion del problema del ticket",
      "ticketType": 1,
      "priority": 3,
      "dateOfCreation": "10/03/2024",
      "dateLastUpdate": "17/03/2024",
      "phone": "1234-1234",
      "name": "Juan",
      "files": [
        "https://storage.googleapis.com/ticketsys_bucket/uploaded-files/89697427-226f-42ae-b27a-f0088fe735a4gato2.png",
        "https://storage.googleapis.com/ticketsys_bucket/uploaded-files/35c4585b-03b8-4c79-ae97-c84c074d0517gato1.jpg"
      ],
      "state": 1,
      "email": "camran@gmail.com"
    },
    {
      "owner": null,
      "ticketNumber": 49,
      "lastName": "ASD",
      "agent": {
        "idUser": null,
        "username": null,
        "name": null,
        "lastName": null,
        "phone": null,
        "userType": null
      },
      "problemSolved": 0,
      "description": "ASDAOWDJIAJWDoi",
      "ticketType": 1,
      "priority": 3,
      "dateOfCreation": "12/03/2024",
      "dateLastUpdate": "12/03/2024",
      "phone": "1234-1234",
      "name": "Juan",
      "files": [
        "https://storage.googleapis.com/ticketsys_bucket/uploaded-files/249825a6-1e52-4447-b67b-f0aaadb976d3gato2.png",
        "https://storage.googleapis.com/ticketsys_bucket/uploaded-files/cc18f99b-7146-45c4-9343-664c20974337gato1.jpg"
      ],
      "state": 1,
      "email": "camran@gmail.com"
    }
  ]
}
```

Get Tickets By TicketNumber

Descripción:

Este microservicio permite a los usuarios obtener información detallada de un ticket específico utilizando su número de ticket.

Endpoint:

- **URL:** `http://127.0.0.1:33065/api/ticket/getTicket-ticketNumber`
- **Método HTTP:** GET

Parámetros:

- `ticketNumber` (query parameter): Número de ticket que se desea obtener.

Respuesta:

json

```
{
  "tickets": {
    "owner": null,
    "lastName": "Perez",
    "ticketNumber": 48,
    "agent": "{ \"username\": \"agente\", \"name\": \"Uriel\",
    \"lastName\": \"Gutierrez\", \"phone\": \"1234-1234\", \"userType\":
    2} ",
    "problemSolved": 0,
    "description": "Descripcion del problema del ticket",
    "ticketType": 1,
    "priority": 3,
    "dateOfCreation": "10/03/2024",
    "dateLastUpdate": "17/03/2024",
    "phone": "1234-1234",
    "name": "Juan",
    "files": [

      "https://storage.googleapis.com/ticketsys_bucket/uploaded-files/8969
      7427-226f-42ae-b27a-f0088fe735a4gato2.png",

      "https://storage.googleapis.com/ticketsys_bucket/uploaded-files/35c4
      585b-03b8-4c79-ae97-c84c074d0517gato1.jpg"
    ],
    "state": 1,
    "email": "camran@gmail.com"
  }
}
```

Descripción de la Respuesta:

- `tickets`: Objeto que contiene la información detallada del ticket correspondiente al número proporcionado.

- Los campos incluidos son similares a los de la respuesta anterior, pero solo para el ticket específico solicitado.

Get Unsolved Tickets

Descripción:

Este microservicio permite a los usuarios obtener una lista de todos los tickets almacenados en el sistema.

Endpoint:

- **URL:** `http://127.0.0.1:33065/api/ticket/getTickets-all`
- **Método HTTP:** GET

Encabezado:

- **jwt:** Token de autenticación JWT proporcionado para la autorización.

Respuesta:

json

```
[
  {
    "ticketNumber": 48,
    "email": "camran@gmail.com",
    "name": "Juan",
    "lastName": "Perez",
    "phone": "1234-1234",
    "dateLastUpdate": "10/03/2024",
    "dateOfCreation": "10/03/2024",
    "description": "Descripcion del problema del ticket",
    "ticketType": 1,
    "priority": 3
  },
  {
    "ticketNumber": 49,
    "email": "camran@gmail.com",
    "dateOfCreation": "12/03/2024",
    "name": "Juan",
    "lastName": "ASD",
    "description": "ASDAOWDJIAJWDoi",
  }
]
```

```
        "phone": "1234-1234",
        "ticketType": 1,
        "dateLastUpdate": "12/03/2024",
        "priority": 3
    }
]
```

Descripción de la Respuesta:

- La respuesta es una lista de objetos, donde cada objeto representa un ticket.
- Cada objeto contiene detalles como el número de ticket, dirección de correo electrónico, nombre, apellido, teléfono, fecha de creación, fecha de última actualización, descripción, tipo de ticket y prioridad.

Get Agent Tickets

Descripción:

Este microservicio permite a los usuarios obtener una lista de todos los tickets asignados a un agente específico.

Endpoint:

- **URL:** `http://127.0.0.1:33065/api/tracking/get-agent-all`
- **Método HTTP:** GET

Encabezado:

- **jwt:** Token de autenticación JWT proporcionado para la autorización.

Respuesta:

```
json
[
    {
        "lastName": "Perez",
        "ticketNumber": 48,
        "agent": "{ \"username\": \"agente\", \"name\": \"Uriel\",
        \"lastName\": \"Gutierrez\", \"phone\": \"1234-1234\", \"userType\":
        2}",
        "problemSolved": 0,
        "description": "Descripcion del problema del ticket",
        "ticketType": 1,
        "priority": 3,
```

```
    "dateOfCreation": "10/03/2024",
    "dateLastUpdate": "17/03/2024",
    "phone": "1234-1234",
    "name": "Juan",
    "files": [

      "https://storage.googleapis.com/ticketsys_bucket/uploaded-files/89697427-226f-42ae-b27a-f0088fe735a4gato2.png",

      "https://storage.googleapis.com/ticketsys_bucket/uploaded-files/35c4585b-03b8-4c79-ae97-c84c074d0517gato1.jpg"
    ],
    "state": 1,
    "email": "camran@gmail.com"
  }
]
```

Descripción de la Respuesta:

- La respuesta es una lista de objetos, donde cada objeto representa un ticket asignado al agente especificado.
- Cada objeto contiene detalles como el número de ticket, nombre, apellido, teléfono, fecha de creación, fecha de última actualización, descripción, tipo de ticket, prioridad y estado del ticket.
- El campo **agent** contiene información sobre el agente responsable del ticket, en formato JSON.

Login

Descripción:

Este microservicio permite a los usuarios iniciar sesión proporcionando un nombre de usuario y una contraseña.

Endpoint:

- **URL:** `http://34.29.173.17:33065/api/user/login`
- **Método HTTP:** POST

Cuerpo de la Solicitud (Body):

```
json
{
  "username": "admin",
```



```
    "password": "admin"
}
```

Respuesta:

json

```
{
  "user_type": {
    "idUserType": 3,
    "type": "administrador"
  },
  "jwt":
  "eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX3R5cGUiOiIzIiwidXNlcm5hbWUiOiJhZG1pbGlzIm1hdCI6MTcxMDY4MTA2MiwiZXhwIjoxNzEwNjk1NDYyfQ.mpusha13jAUmAfbdCeThDyCnzdVE-y0uGPPTxrLSTA4"
}
```

Descripción de la Respuesta:

- **user_type:** Objeto que contiene el tipo de usuario.
 - **idUserType:** ID del tipo de usuario.
 - **type:** Tipo de usuario (en este caso, "administrador").
- **jwt:** Token de autenticación JWT generado para el usuario.

Register

Descripción:

Este microservicio permite a los usuarios registrarse en el sistema proporcionando sus datos personales y credenciales de inicio de sesión.

Endpoint:

- **URL:** <http://34.29.173.17:33065/api/user/register>
- **Método HTTP:** POST

Cuerpo de la Solicitud (Body):

json

```
{
  "username": "admin",
  "password": "admin",
}
```

```
"userType": "administrador",
"name": "r'ha",
"lastName": "Apellido",
"phone": "0000-0000"
}
```

Respuesta:

- **Status Code:** 201 (Created)

Descripción de la Respuesta:

- El servidor responderá con un código de estado HTTP 201 (Created) para indicar que el registro se ha completado con éxito.

Register Protected

Descripción:

Este microservicio permite a los usuarios registrar un tipo especial de usuario en el sistema, como un agente.

Endpoint:

- **URL:** `http://127.0.0.1:33065/api/user/registerSpecial`
- **Método HTTP:** POST

Encabezado:

- `jwt`: Token de autenticación JWT proporcionado para la autorización.

Cuerpo de la Solicitud (Body):

json

```
{
  "username": "camran",
  "password": "admin",
  "userType": "agente",
  "name": "r'ha",
  "lastName": "Calgar",
  "phone": "0000-0000"
}
```

Respuesta:

- **Status Code:** 201 (Created)

Descripción de la Respuesta:

- El servidor responderá con un código de estado HTTP 201 (Created) para indicar que el registro especial se ha completado con éxito.

Assign Agent

Descripción:

Este microservicio permite asignar un agente a un ticket específico.

Endpoint:

- **URL:** <http://127.0.0.1:33065/api/tracking/assign-agent>
- **Método HTTP:** POST

Encabezado:

- **jwt:** Token de autenticación JWT proporcionado para la autorización.

Cuerpo de la Solicitud (Body):

json

```
{  
  "ticketNumber": 48  
}
```

Respuesta:

- **Código de Estado:** 201 (Creado)

Descripción de la Respuesta:

- El servidor responde con un código de estado HTTP 201 (Creado) para indicar que la asignación de agente al ticket se ha completado con éxito.

create-ticket

Descripción:

Este microservicio permite crear un nuevo ticket en el sistema.

Endpoint:

- **URL:** `http://127.0.0.1:33065/api/ticket/create-ticket`
- **Método HTTP:** POST

Encabezado:

- `jwt`: Token de autenticación JWT proporcionado para la autorización.

Cuerpo de la Solicitud (Body):

json

```
{
  "email": "camran@gmail.com",
  "name": "name",
  "lastName": "lastName",
  "phone": "5967-0379",
  "ticketType": 1,
  "ticketPriority": 3,
  "description": "description",
  "files": {}
}
```

Respuesta:

- **Código de Estado:** 201 (Creado)

Descripción de la Respuesta:

- El servidor responde con un código de estado HTTP 201 (Creado) para indicar que el ticket se ha creado con éxito.

Get Logs

Descripción:

Este microservicio permite a los usuarios obtener el historial de seguimiento de un ticket específico.

Endpoint:

- **URL:** `http://127.0.0.1:33065/api/tracking/trackLogs`
- **Método HTTP:** POST

Cuerpo de la Solicitud (Body):

```
json
{
  "ticketNumber": 48
}
```

Respuesta:

```
json
[
  {
    "idHistory": 1,
    "ticketNumber": 48,
    "dateTimeContacted": "2024-03-10T08:25:58.000+00:00",
    "sent": "",
    "received": "",
    "description": "Se creo el ticket de tipo:tecnico, de prioridad:baja"
  }
]
```

Descripción de la Respuesta:

- La respuesta es una lista de objetos que representan el historial de seguimiento del ticket solicitado.
- Cada objeto contiene detalles como el ID del historial, el número de ticket, la fecha y hora del contacto, detalles sobre el envío y la recepción (si corresponde) y una descripción del evento.

upload

Descripción:

Este microservicio permite a los usuarios cargar un archivo en un servicio de almacenamiento en la nube.

Endpoint:

- **URL:** `http://127.0.0.1:33065/api/cloud-storage/uploadFile`
- **Método HTTP:** POST

Cuerpo de la Solicitud (Body):

- Tipo de contenido: `form-data`
- Campos:
 - `file`: El archivo que se va a cargar, con el nombre `imagen_gato`.

Respuesta:

```
json
{
  "fileUrl":
  "https://storage.googleapis.com/ticketsys_bucket/uploaded-files/imagen_gato"
}
```

Descripción de la Respuesta:

- `fileUrl`: La URL del archivo cargado en el servicio de almacenamiento en la nube.

Validate JWT

Descripción:

Este microservicio permite a los usuarios validar si su sesión de usuario es válida.

Endpoint:

- **URL:** `http://127.0.0.1:33065/api/user/validate`
- **Método HTTP:** GET

Encabezado:

- `jwt`: Token de autenticación JWT proporcionado para la autorización.

Respuesta:

- **Código de Estado:** 200 (OK)

Descripción de la Respuesta:

- Un código de estado HTTP 200 indica que la sesión del usuario es válida.

saveMessage

Descripción:

Este microservicio permite a los usuarios guardar un mensaje en el historial de seguimiento de un ticket específico.

Endpoint:

- **URL:** `http://127.0.0.1:33065/api/tracking/saveMessage`
- **Método HTTP:** POST

Cuerpo de la Solicitud (Body):

json

```
{
  "ticketNumber": 48,
  "description": "Soy roboute Guilliman",
  "sent": "usuario",
  "date": "2024-03-14T19:50:28.366Z"
}
```

Respuesta:

- **Código de Estado:** 200 (OK)
- **Cuerpo de la Respuesta:**

json

```
{
  "idHistory": 2,
  "ticketNumber": 48,
  "dateTimeContacted": "2024-03-15T00:50:28.366+00:00",
  "sent": "usuario",
  "received": "",
  "description": "Soy roboute Guilliman"
}
```

Descripción de la Respuesta:

- Un código de estado HTTP 200 indica que el mensaje se ha guardado correctamente en el historial de seguimiento del ticket.
- La respuesta incluye detalles sobre el mensaje guardado, como el ID del historial, el número de ticket, la fecha y hora del contacto, el remitente, el destinatario (en caso de existir) y la descripción del mensaje.

changeBlock

Descripción:

Este microservicio cambia el estado de bloqueo de un usuario, si está bloqueado no podrá iniciar sesión. Se envía un body con username y blocked. En *blocked* 1 significa que será bloqueado el usuario y en 0 el usuario no está bloqueado.

Endpoint:

- **URL:** `http://127.0.0.1:33065/api/user/changeBlock`
- **Método HTTP:** POST

Cuerpo de la Solicitud (Body):

json

```
{
  "username": "admin",
  "blocked": 0
}
```

Respuesta:

- **Código de Estado:** 200 (OK)
- **Cuerpo de la Respuesta:**

json

```
{
}
```

Descripción de la Respuesta:

- Un código de estado HTTP 200 indica que la actualización de estado se ha realizado correctamente

changeBlock

Descripción:

Este microservicio cambia el estado de bloqueo de un usuario, si está bloqueado no podrá iniciar sesión. Se envía un body con username y blocked. En *blocked* 1 significa que será bloqueado el usuario y en 0 el usuario no está bloqueado.

Endpoint:

- **URL:** `http://127.0.0.1:33065/api/user/changeBlock`
- **Método HTTP:** POST

Cuerpo de la Solicitud (Body):

json

```
{  
  "username": "admin",  
  "blocked": 0  
}
```

Respuesta:

- **Código de Estado:** 200 (OK)
- **Cuerpo de la Respuesta:**

json

```
{  
  
}
```

Descripción de la Respuesta:

- Un código de estado HTTP 200 indica que la actualización de estado se ha realizado correctamente.

getDataReport

Descripción:

Este microservicio obtiene un informe del volumen de los tickets y métricas de los tickets calificados en base a las encuestas. Se envía los parámetros ticketNumber, una descripción en description, al usuario y la fecha.

Endpoint:

- **URL:** `http://127.0.0.1:33065/api/survey/getDataReport`
- **Método HTTP:** POST

Cuerpo de la Solicitud (Body):

json

```
{
  "ticketNumber": 48,
  "description": "Soy Roboute Guilliman",
  "sent": "usuario",
  "date": "2024-03-14T19:50:28.366Z"
}
```

Respuesta:

- **Código de Estado:** 200 (OK)
- **Cuerpo de la Respuesta:**

json

```
{
  "unsolvedTickets": [],
  "unqualifiedTickets": [],
  "surveys": [
    {
      "# Ticket": 51,
      "Calidad de Servicio": 3,
      "Satisfaccion": 4,
      "Tiempo de Servicio": 2
    },
    {
      "# Ticket": 50,
      "Tiempo de Servicio": 5,
      "Satisfaccion": 3,
      "Calidad de Servicio": 4
    },
  ],
}
```

```
{
  "# Ticket": 52,
  "Satisfaccion": 2,
  "Tiempo de Servicio": 3,
  "Calidad de Servicio": 4
},
{
  "# Ticket": 53,
  "Satisfaccion": 5,
  "Calidad de Servicio": 4,
  "Tiempo de Servicio": 2
}
]
```

Descripción de la Respuesta:

- Despliega un listado de tickets con los resultados de los tickets, esto en base a las encuestas respondidas por el usuario.

DEFINIR CI-CD

developCI.yaml

Etapas 1: build_stage

Esta etapa se encarga de compilar y construir los microservicios de tu proyecto. Contiene los siguientes trabajos:

Trabajo 1: build_stage

- Plataforma: Ubuntu Latest
- Estrategia: Utiliza una matriz para ejecutar el trabajo con diferentes versiones de Node.js y Java JDK 17.
- Pasos:
 - Verifica y copia el código del repositorio.
 - Configura Node.js con la versión especificada en la matriz.
 - Configura Java JDK 17.
 - Instala y construye el microservicio frontend.
 - Instala y construye el microservicio tickets (usando Maven).

Etapas 2: delivery-stage

Esta etapa se encarga de implementar y entregar los microservicios construidos a un entorno específico. Contiene los siguientes trabajos:

Trabajo 1: delivery-stage

- Plataforma: Ubuntu Latest
- Estrategia: Utiliza una matriz para ejecutar el trabajo con diferentes versiones de Node.js y Java JDK 17.
- Variables de Entorno:
 - REGION_REGISTRY: us-central1-docker.pkg.dev
 - PROJECT_ID: proyectosa-415901
 - REPOSITORY: docker-repository
 - ENVIRONMENT: development
- Pasos:
 - Verifica y copia el código del repositorio.
 - Configura Node.js con la versión especificada en la matriz.
 - Configurar Java JDK 17.
 - Configura las credenciales de Google Cloud Platform.
 - Configura Cloud SDK.
 - Usa gcloud CLI.
 - Configura permisos de ejecución para un script.
 - Realiza el etiquetado automático de versiones de releases.
 - Configura Docker con autenticación de Google Cloud.
 - Configurar variables de entorno para el frontend.
 - Sube la imagen Docker del microservicio MySQL al Registro de Artefactos.
 - Sube la imagen Docker del microservicio Frontend al Registro de Artefactos.
 - Sube la imagen Docker del microservicio Ticket al Registro de Artefactos.
 - Elimina la versión anterior del despliegue en el clúster GKE (Google Kubernetes Engine).
 - Implementa el despliegue en el clúster GKE.

featureCl.yaml

Etapas 1: build_stage

Esta etapa se encarga de compilar y construir los microservicios del proyecto. Contiene los siguientes trabajos:

Trabajo 1: build_stage

- Plataforma: Ubuntu Latest
- Estrategia: Utiliza una matriz para ejecutar el trabajo con diferentes versiones de Node.js y Java JDK 17.
- Pasos:
 - Verifica y copia el código del repositorio.
 - Configura Node.js con la versión especificada en la matriz.
 - Configura Java JDK 17.
 - Instala y construye el microservicio frontend.
 - Instala y construye el microservicio tickets (usando Maven).

Etapa 2: delivery-stage

Esta etapa se encarga de implementar y entregar los microservicios construidos a un entorno específico. Contiene los siguientes trabajos:

Trabajo 1: delivery-stage

- Plataforma: Ubuntu Latest
- Estrategia: Utiliza una matriz para ejecutar el trabajo con diferentes versiones de Node.js y Java JDK 17.
- Variables de Entorno:
 - REGION_REGISTRY: us-central1-docker.pkg.dev
 - PROJECT_ID: proyectosa-415901
 - REPOSITORY: docker-repository
 - ENVIRONMENT: test
- Pasos:
 - Verifica y copia el código del repositorio.
 - Configura Node.js con la versión especificada en la matriz.
 - Configura Java JDK 17.
 - Configura las credenciales de Google Cloud Platform.
 - Configura Cloud SDK.
 - Usa gcloud CLI.
 - Configura permisos de ejecución para un script.
 - Realiza el etiquetado automático de versiones de releases.
 - Configura Docker con autenticación de Google Cloud.
 - Configura variables de entorno para el frontend.
 - Sube la imagen Docker del microservicio MySQL al Registro de Artefactos.
 - Sube la imagen Docker del microservicio Frontend al Registro de Artefactos.
 - Sube la imagen Docker del microservicio Ticket al Registro de Artefactos.

Elimina la versión anterior del despliegue en el clúster GKE (Google Kubernetes Engine).
Implementa el despliegue en el clúster GKE.

mainCI.yaml

Etapas 1: delivery-stage

Esta etapa se encarga de implementar y entregar los microservicios construidos a un entorno de producción. Contiene los siguientes trabajos:

Trabajo 1: delivery-stage

- Plataforma: Ubuntu Latest
- Pasos:
 - Verifica y copia el código del repositorio.
 - Configura las credenciales de Google Cloud Platform.
 - Configura Cloud SDK.
 - Usa gcloud CLI.
 - Elimina la versión anterior del despliegue en el clúster GKE (Google Kubernetes Engine).
 - Implementa el despliegue en el clúster GKE.

releaseCI.yaml

Etapas 1: build_stage

Esta etapa se encarga de construir los microservicios del proyecto. Contiene los siguientes trabajos:

Trabajo 1: build_stage

- Plataforma: Ubuntu Latest
- Estrategia:
 - Se especifica la versión de Node.js y Java JDK para la matriz de ejecución.
- Pasos:
 - Verifica y copia el código del repositorio.
 - Configura Node.js con la versión especificada.
 - Configura Java JDK 17.
 - Instala y construye el microservicio frontend.
 - Instala y construye el microservicio tickets.

Etapa 2: delivery-stage

Esta etapa se encarga de implementar y entregar los microservicios construidos a un entorno de producción. Contiene los siguientes trabajos:

Trabajo 1: delivery-stage

- Plataforma: Ubuntu Latest
- Estrategia:
 - Se especifica la versión de Node.js y Java JDK para la matriz de ejecución.
- Variables de entorno:
 - REGION_REGISTRY: Región del registro de artefactos.
 - PROJECT_ID: ID del proyecto en Google Cloud Platform.
 - REPOSITORY: Repositorio de Docker.
 - ENVIRONMENT: Entorno de implementación (production).
- Pasos:
 - Verifica y copia el código del repositorio.
 - Configura Node.js con la versión especificada.
 - Configura Java JDK 17.
 - Configura las credenciales de Google Cloud Platform.
 - Configura Cloud SDK.
 - Usa gcloud CLI.
 - Establece permisos de ejecución para un script.
 - Realiza un etiquetado automático de las versiones.
 - Configura Docker con autenticación de Google Cloud.
 - Configura las variables de entorno del frontend.
 - Publica las imágenes Docker de los microservicios en el Registro de Artefactos de Google Cloud.

Manual de Instalación de Runner

El GitHub Actions Runner es una aplicación que permite ejecutar trabajos de GitHub Actions en tu propio entorno, lo que proporciona flexibilidad y control sobre la ejecución de tus flujos de trabajo. A continuación, se detallan los pasos para instalar y configurar el GitHub Actions Runner para tus repositorios.

Requisitos Previos

- Sistema Operativo: Ubuntu Latest (o cualquier sistema compatible con Docker)
- Acceso a Internet
- Cuenta de GitHub: Debes tener acceso al repositorio en el que deseas configurar el runner.

Pasos de Instalación

1. Descargar el Código Fuente del Runner

El código fuente del GitHub Actions Runner está disponible en el repositorio oficial de GitHub. Descarga la última versión estable del runner desde el siguiente enlace: [GitHub Actions Runner Releases](#).

2. Preparar el Entorno

Asegúrate de tener Docker instalado en tu sistema. Si aún no lo tienes instalado, puedes seguir las instrucciones de instalación en la documentación oficial de Docker.

3. Configurar el Runner

Una vez descargado el código fuente del runner, descomprímelo en la ubicación deseada en tu sistema. Abre una terminal y navega hasta el directorio donde se encuentra el código fuente del runner.

4. Registrar el Runner

Para registrar el runner en tu repositorio de GitHub, sigue estos pasos:

- a. En la terminal, ejecuta el siguiente comando para iniciar el proceso de registro:

```
bash
```

Copy code

Sustituye `tu-usuario` y `tu-repositorio` con tu nombre de usuario y el nombre de tu repositorio, respectivamente. Además, reemplaza `TU_TOKEN` con un token de acceso personal generado desde tu cuenta de GitHub con los permisos necesarios (p. ej., `repo, write:packages, read:packages, delete:packages, workflow`).

b. Sigue las instrucciones que aparecen en la terminal para completar el proceso de registro. Deberás proporcionar un nombre para el runner y seleccionar las etiquetas opcionales según sea necesario.

c. Una vez completado el registro, el runner estará listo para ejecutar los trabajos de GitHub Actions en tu repositorio.

5. Configurar y Ejecutar los Trabajos

Los archivos de configuración de GitHub Actions que proporcionaste (`.yml`) definen los flujos de trabajo y los trabajos a ejecutar en tu repositorio. Asegúrate de que estos archivos estén presentes en tu repositorio y que los trabajos estén configurados para usar el runner registrado.

6. Verificar el Estado del Runner

Puedes verificar el estado del runner desde la sección "Acciones" de tu repositorio en GitHub. Además, puedes ver registros detallados de la ejecución de los trabajos en la pestaña "Acciones" de tu repositorio.