15ª Emisión

DIPLOMADO

Desarrollo de Sistemas con Tecnología Java

Módulo 5-6

Desarrollo de aplicaciones empresariales con Jakarta EE

Uriel Hernández

DGTIC

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Dirección General de Cómputo y de Tecnologías de información y Comunicación
Dirección de Docencia en TIC

50 AÑOS
Educación Continua
1971 - 2021

API Usuario

Registrar Usuario

Usuario

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
public class Usuario {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    private String nombre;
    @Column(name="primer_apellido")
    private String primerApellido;
    @Column(name="segundo_apellido")
    private String segundoApellido;
    private String password;
    private String email;
    private String rfc;

}
```

JpaUsuarioRepository

```java
@Singleton
public class JpaUsuarioRepository implements UsuarioRepository {

    @PersistenceContext(unitName="pixup")
    private EntityManager entityManager;

    @Override
    public Optional<Usuario> findByEmail(String email) {
        TypedQuery<Usuario> query = entityManager.createQuery(
                "SELECT u FROM Usuario u WHERE u.email = :email", Usuario.class);
        query.setParameter("email", email);
        List<Usuario> usuarios = query.getResultList();
        return !usuarios.isEmpty() ? Optional.of(usuarios.get(0)) : Optional.empty();
    }

    @Override
    public Usuario save(Usuario usuario) {
        entityManager.persist(usuario);
        return usuario;
    }

}
```

UsuarioService

```java
@Local
public interface UsuarioService {

    Usuario registrarUsuario(Usuario usuario, Domicilio domicilio);

}
```

UsuarioServiceImpl

```java
@Stateless
public class UsuarioServiceImpl implements UsuarioService {

    @Inject
    private UsuarioRepository usuarioRepository;
    @Inject
    private DomicilioRepository domicilioRepository;
    @Inject
    private ColoniaRepository coloniaRepository;
    @Inject
    private TipoDomicilioRepository tipoDomicilioRepository;

    @Override
    @Transactional(value=Transactional.TxType.REQUIRED)
    public Usuario registrarUsuario(Usuario usuario, Domicilio domicilio) {
        // validacion usuario duplicado
        Optional<Usuario> usuarioExistente =
                usuarioRepository.findByEmail(usuario.getEmail());
        if (usuarioExistente.isPresent()) {
            throw new UsuarioAlreadyExistsException(usuario.getEmail());
        }
```

JTA (Java Transaction API)

- Es una especificación Java que define un estándar de interfaces entre un gestor de transacciones (transaction manager) y las partes involucradas en un sistema de transacciones distribuidas.
- Permite a las aplicaciones realizar transacciones distribuidas por medio del contenedor (application server). Las aplicaciones pueden definir las transacciones de forma declarativa por medio de la anotación **@Transactional**

Propagation Behaviour (Comportamiento de propagación)

- Se refiere al comportamiento que debe tener la ejecución de un método en un contexto transaccional, existen los siguientes comportamientos:
  - **REQUIRED**: Debe invocarse dentro de una transacción, si existe una transacción en curso se el método se ejecutará dentro de esa transacción, si no existe una transacción en curso se creará una nueva.
  - **MANDATORY**: Debe invocarse dentro de una transacción, si no existe una transacción en curso se generará una excepción.
  - **NEVER**: No debe invocarse dentro de una transacción, si existe una transacción en curso se generará una excepción.
  - **NOT_SUPPORTED**: No debería participar dentro de una transacción, si existe una transacción en curso se suspende temporalmente, reanuda la ejecución de la transacción cuando acaba la ejecución del método.
  - **REQUIRES_NEW**: Siempre inicia una nueva transacción incluso si una transacción en curso existe.
  - **SUPPORTS**: Permite participar al método dentro una transacción si es que existe, si no existe una transacción en curso no se crea una nueva.

ACID

- Conjunto de propiedades que deben exhibir las transacciones de base de datos para garantizar la validez de los datos a pesar de los errores.
- **Atomicity:** Cada transacción (conjunto de operaciones) es tratada como una unidad, por lo que se ejecuta todo el conjunto de manera exitosa (commit) o en caso de error no se aplica ninguna operación (rollback).
- **Consistency:** Una transacción sólo puede llevar a la BD de un estado consistente a otro, cada dato escrito a la BD debe ser válido de acuerdo a los constraints, cascades, triggers e integridad referencial.
- **Isolation:** Las transacciones se ejecutan de manera concurrente (múltiples transacciones pueden estar leyendo/escribiendo en una tabla al mismo tiempo). El nivel de aislamiento define el comportamiento que sucederá entre transacciones (por ejemplo: si una transacción puede ver la modificación de datos realizada por otra transacción que aún no ha dado commit).
- **Durability:** Las transacciones completadas son almacenadas en memoria no volátil (discos duros).

Práctica No. 3

RegistroUsuarioDTO

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class RegistroUsuarioDTO {

    @NotNull(message="Usuario es requerido para realizar el registro")
    @Valid
    private UsuarioRequestDTO usuario;
    @NotNull(message="Domicilio es requerido para realizar el registro")
    @Valid
    private DomicilioDTO domicilio;

}
```

UsuarioApi

```java
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@Path("usuarios")
public interface UsuarioApi {

    @POST
    @Path("registro")
    Response registrarUsuario(
        @NotNull @Valid RegistroUsuarioDTO registroUsuarioDTO);

}
```

UsuarioResource

```
public class UsuarioResource implements UsuarioApi {

    @Inject
    private RegistroUsuarioMapper registroUsuarioMapper;
    @Inject
    private UsuarioService usuarioService;

    @Override
    public Response registrarUsuario(RegistroUsuarioDTO registroUsuarioDTO) {
        Usuario usuarioCreado = usuarioService.registrarUsuario(
                registroUsuarioMapper.toUsuario(registroUsuarioDTO.getUsuario()),
                registroUsuarioMapper.toDomicilio(registroUsuarioDTO.getDomicilio()));
        return Response.status(Response.Status.CREATED)
                .entity(registroUsuarioMapper.toDto(usuarioCreado)).build();
    }

}
```

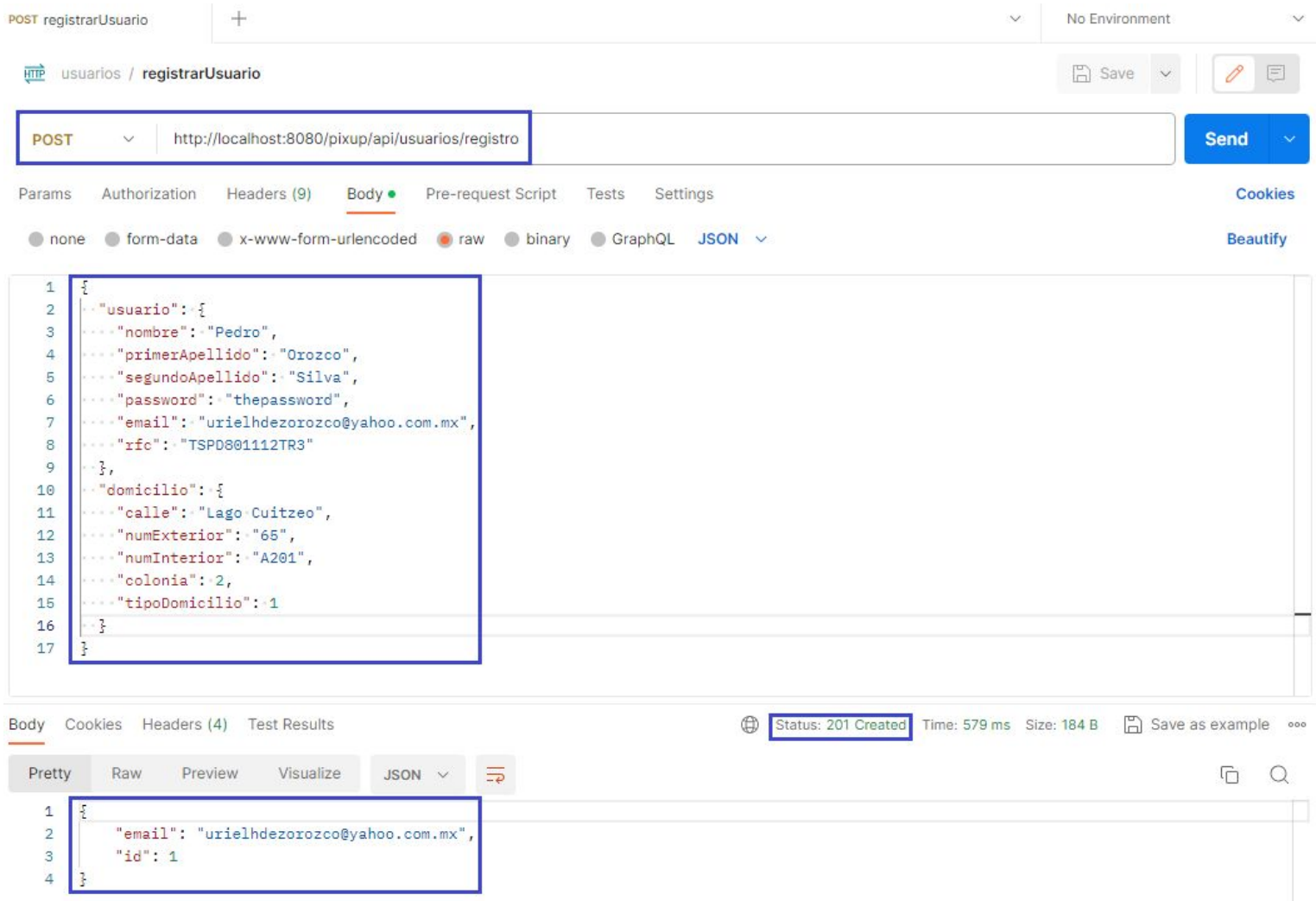
ErrorResponse

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ErrorResponse {

    private Integer estatus;
    private String tipo;
    private String mensaje;

}
```

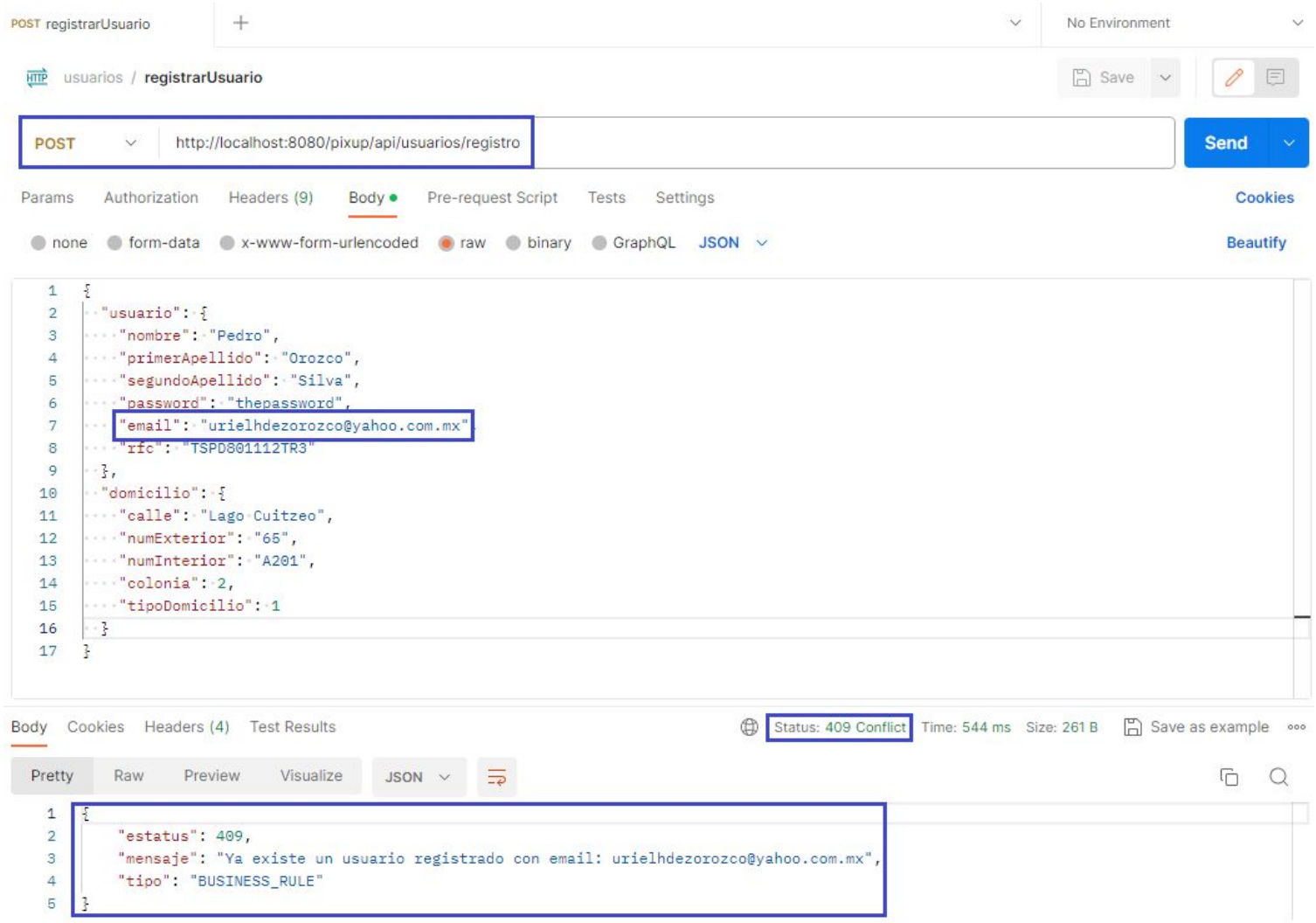
ExceptionMapper

```java
@Provider
public class UsuarioAlreadyExistsExceptionMapper
        implements ExceptionMapper<UsuarioAlreadyExistsException> {

    @Override
    public Response toResponse(UsuarioAlreadyExistsException e) {
        return Response
                .status(Response.Status.CONFLICT)
                .entity(new ErrorResponse(
                        Response.Status.CONFLICT.getStatusCode(),
                        "BUSINESS_RULE",
                        e.getMessage()))
                .build();
    }

}
```

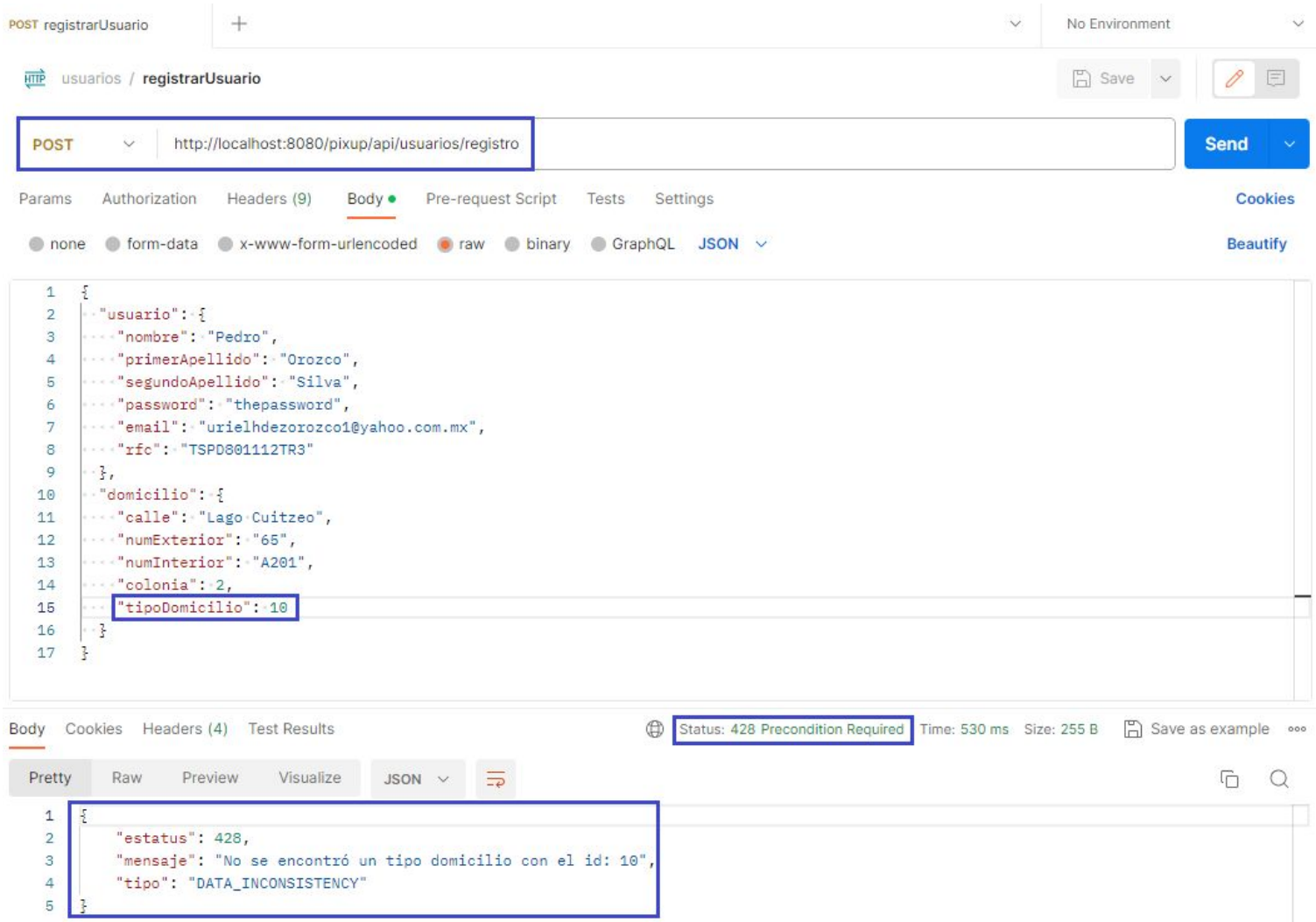
Postman: UsuarioResource POST registrarUsuario



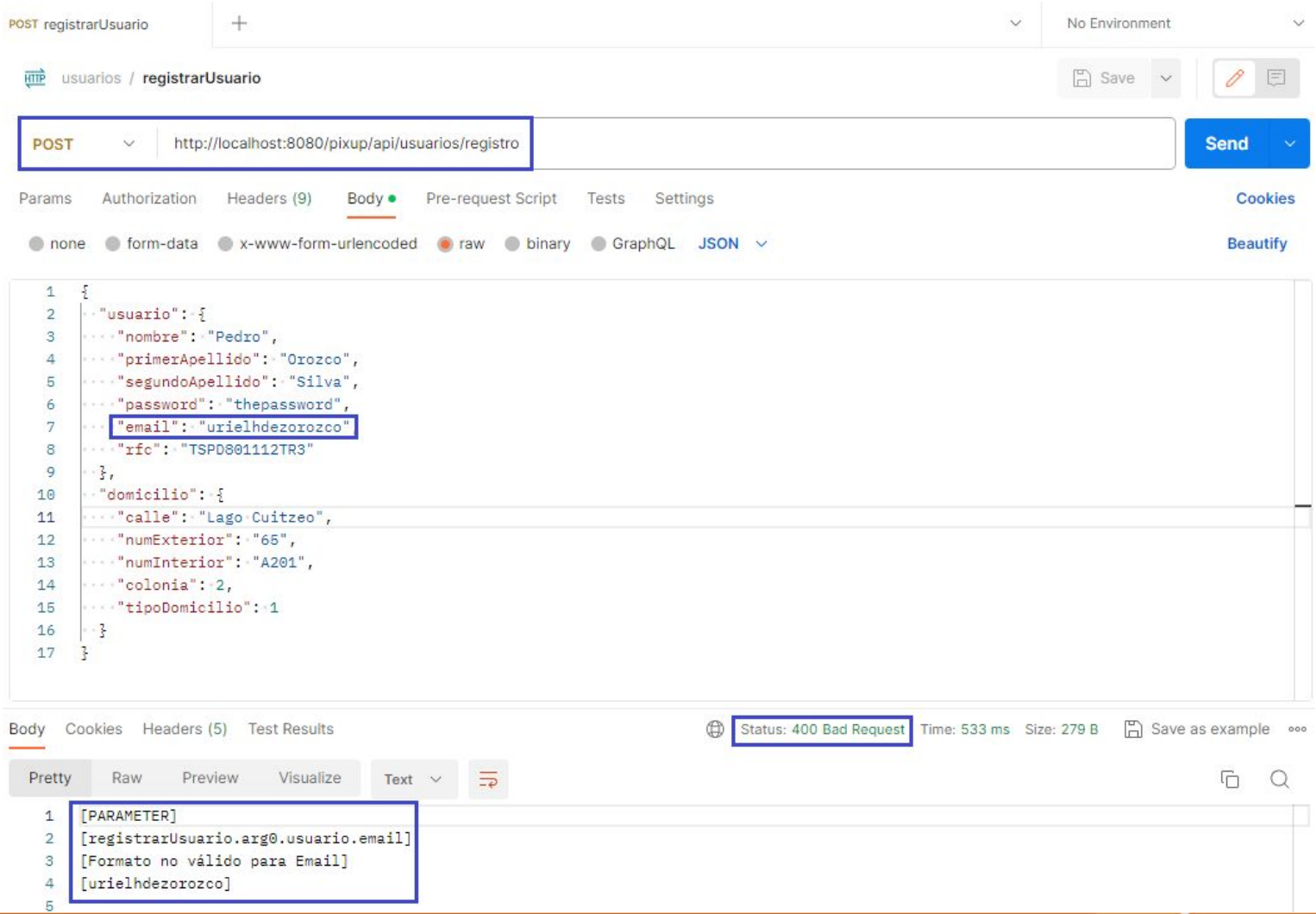
Postman: UsuarioResource POST registrarUsuario – UK duplicated



Postman: UsuarioResource POST registrarUsuario – TipoDomicilio inexistente



Postman: UsuarioResource POST registrarUsuario – Validator



Práctica Final

Contacto

Uriel Hernández

Solution Architect

urielhdezorozco@yahoo.com.mx

Redes sociales:

https://www.linkedin.com/in/juho-mex