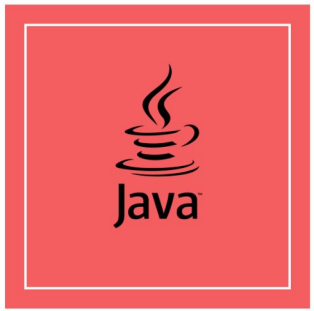
# Proyecto de Gestión de la Sala Comunitaria

Java y Aplicaciones Avanzadas

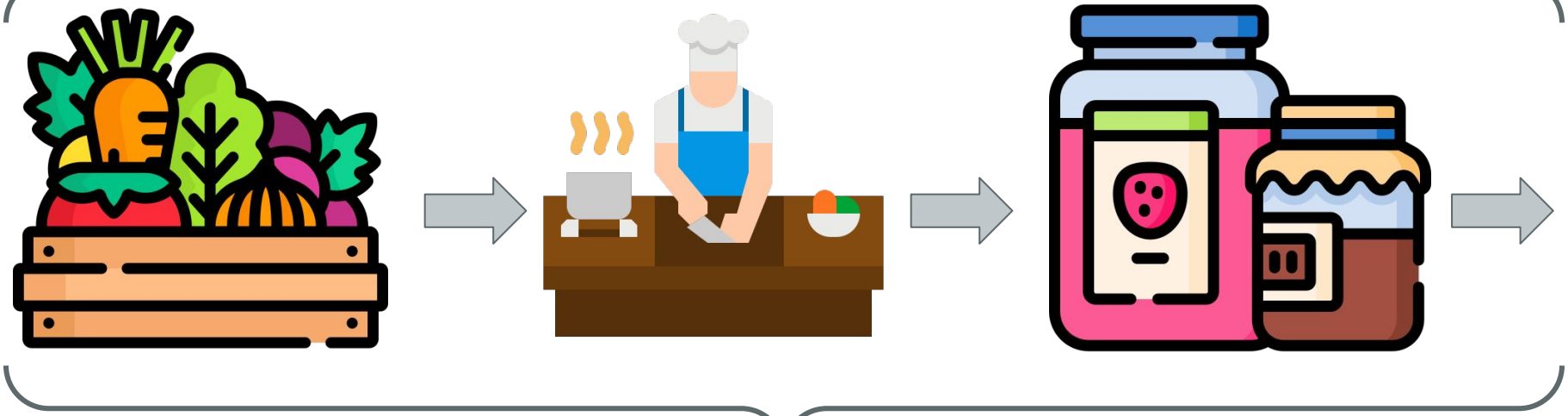
**Grupo 9**
Mateo Novomiski
Federico Herce
Gianfranco Quaranta

# Dominio del problema

# Dominio del problema

# Dominio del problema



Sistema integral de procesamiento

# Etapas del desarrollo

- Análisis, diseño y bocetado
- Definición del modelo de objetos
- Desarrollo de la capa de persistencia
- Desarrollo de la capa de servicios (API REST)
- Desarrollo de la capa de presentación e integración con la capa de servicios
- Sistema completo

# Análisis, diseño y bocetado

- **Análisis del problema presentado**
- **Elaboración de la idea principal**

- Llegada de **recursos** → materia prima → familia productora
  → insumos
- Agregado de valor → lote de elaboración
- Stock de productos → productos
  → canales de venta

# Análisis, diseño y bocetado

- ## Desarrollo de Historias de Usuario

**Editar materia prima**

Un usuario administrador desea editar un cargamento de materia prima registrado, para lo cual clickea el botón ‘Editar’ en el cargamento correspondiente y se le permite modificar cantidad, costo por kilo y la fecha de vencimiento.

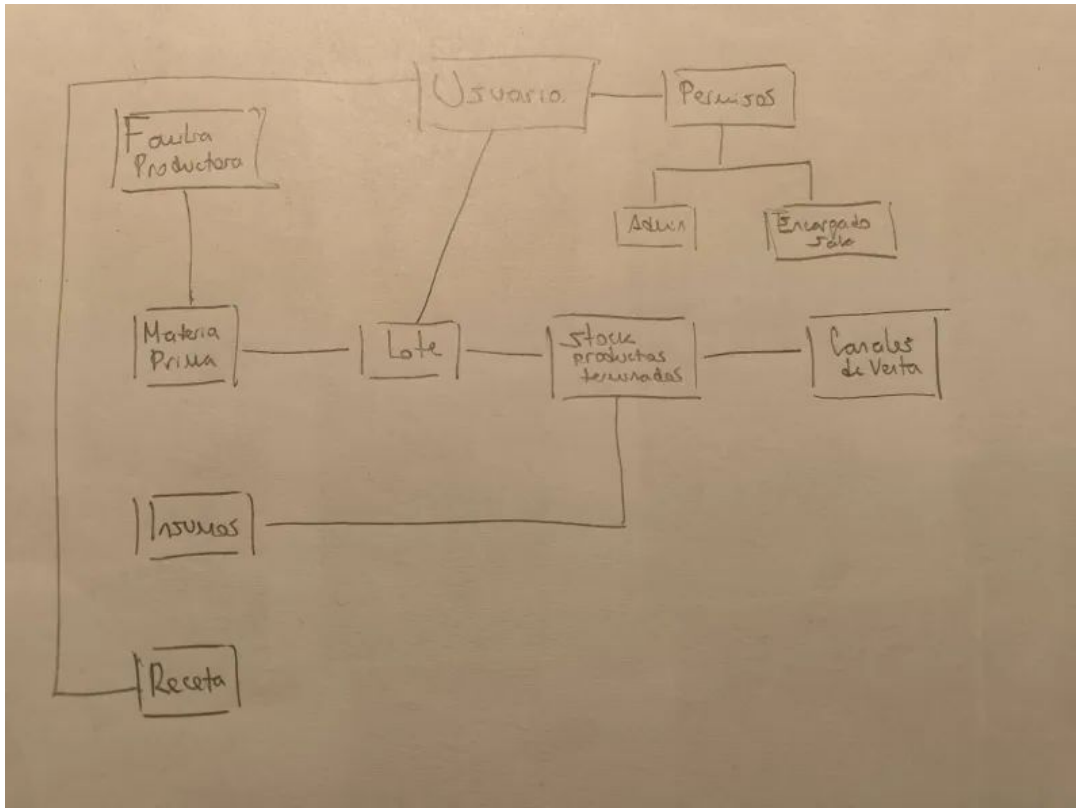
*Escenarios:*

- Edición exitosa: dado que se ingresan nuevos valores que también son válidos, el sistema almacena los cambios
- Edición fallida: dado que alguno(s) de los valores ingresados no cumple con los requisitos establecidos, el sistema despliega un mensaje que informa de la situación y no realiza la actualización.

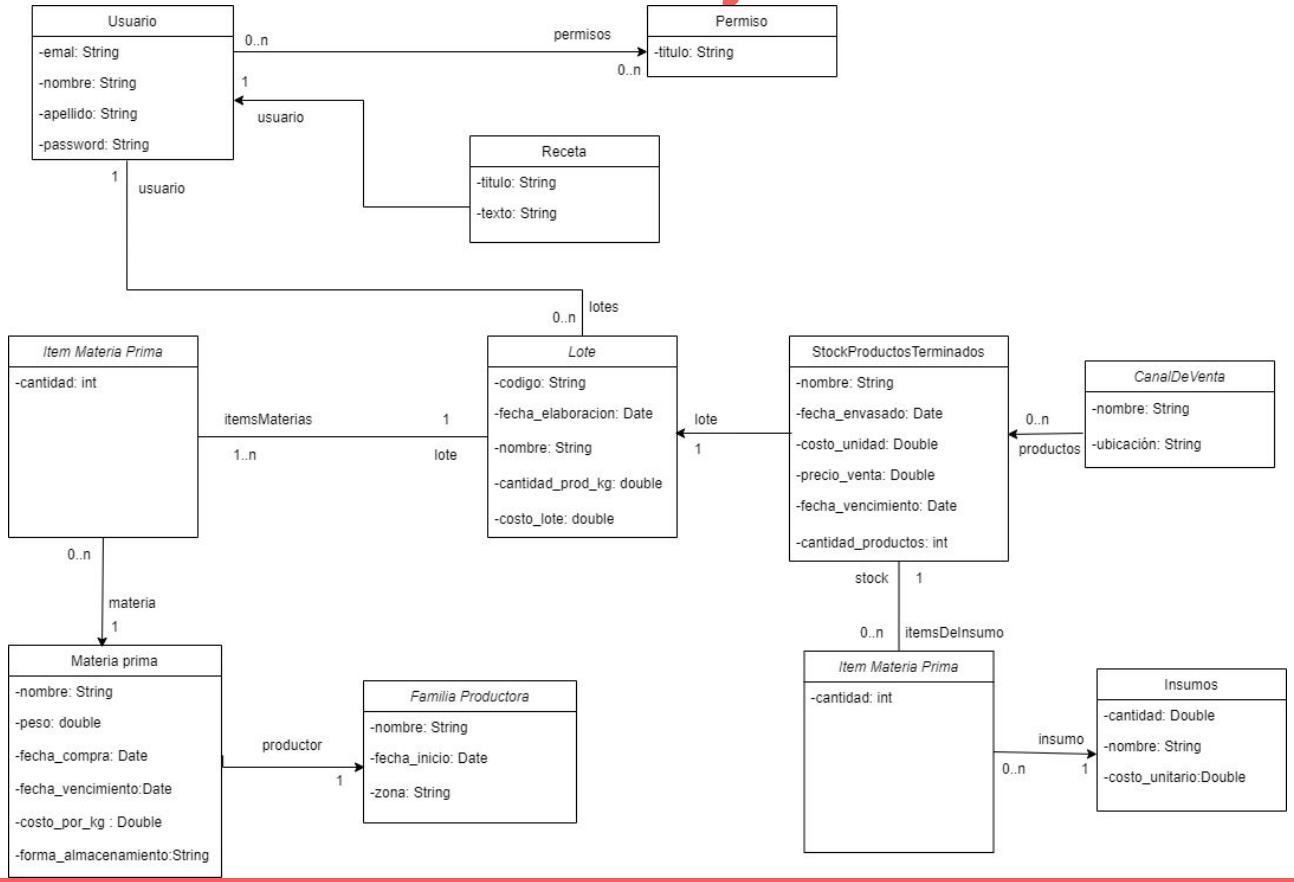
Una historia de usuario es una explicación informal de una función de software escrita en lenguaje coloquial desde la perspectiva del usuario final.

# Análisis, diseño y bocetado

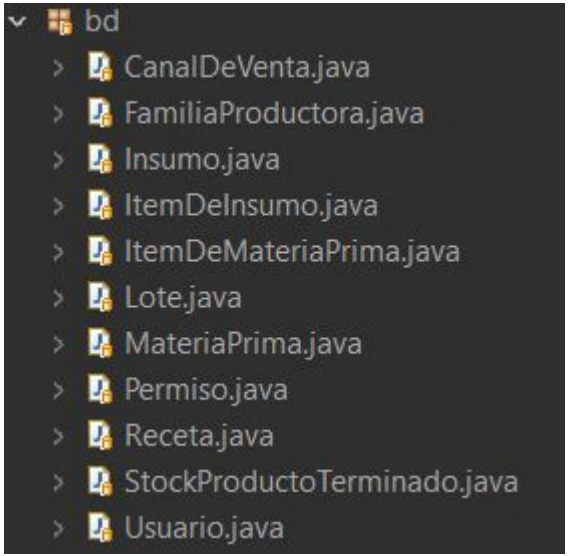
A partir del análisis, además de la funcionalidad básica del sistema, se obtienen las entidades que forman parte del dominio.



# Definición del modelo de objetos



# Definición del modelo de objetos




```java
@Entity
@Table(name = "lotes")
public class Lote {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(unique = true, nullable = false, length = 32, name = "codigo", updatable = false)
    private String codigo;

    @Column(unique = false, nullable = false, length = 64, name = "nombre")
    private String nombre;

    @Column(unique = false, nullable = false, name = "fecha_elaboracion", updatable = false)
    private LocalDate fechaElaboracion;

    @Column(unique = false, nullable = false, name = "cantidad_producida")
    private double cantidadProducida;

    @Column(unique = false, nullable = false, name = "costo_lote")
    private double costoLote;

    @OneToMany(mappedBy = "lote")
    @LazyCollection(LazyCollectionOption.FALSE)
    @JsonIgnore
    private List<ItemDeMateriaPrima> listaItemsDeMateriaPrima;

    private boolean activo;

    @ManyToOne()
    @JoinColumn(name = "usuario_id", updatable = false)
    @JsonBackReference
    private Usuario usuario;
```

# Capa de persistencia

- Patrón DAO
  - **Capa de abstracción entre la capa de negocio y la capa de acceso a datos**. El DAO define una interfaz o una clase abstracta que encapsula la lógica de acceso a datos
- 

# Capa de persistencia

- ABM (CRUD) en todas las entidades

```java
public abstract class GenericDAOImpl<T, ID> implements GenericDAO<T, ID> {

    private Class<T> entityClass;

    @Inject
    private EntityManager em;
```

```java
    @Override
    public void persist(T entity) {
        try {
            em.getTransaction().begin();
            em.persist(entity);
            em.getTransaction().commit();
        } catch (PersistenceException e) {
            if (em.getTransaction().isActive()) {
                em.getTransaction().rollback();
            }
            throw e;
        }
    }
```

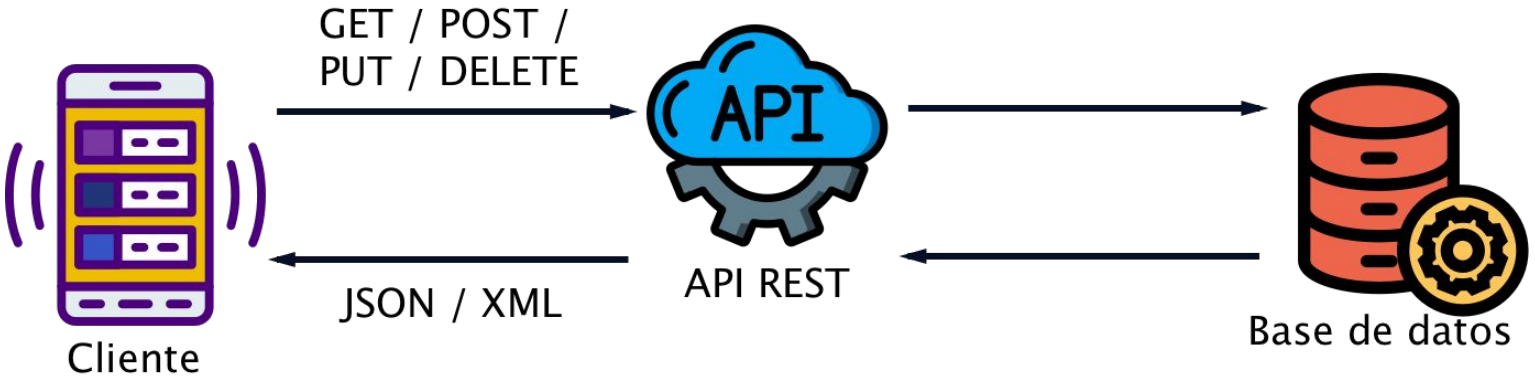
# Capa de persistencia

- Entidades concretas

```java
@Service
public class UsuarioDAO extends GenericDAOImpl<Usuario, Integer>{

    public UsuarioDAO() {
        super(Usuario.class);
    }


    public Usuario findActiveByEmail(String email) {
        try {
            return super.getEm().createQuery(
                    "SELECT e FROM Usuario e WHERE e.email = :email AND e.activo = true", Usuario.class)
                    .setParameter("email", email).getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }

    public Usuario findByEmail(String email) {
        try {
            return super.getEm().createQuery(
                    "SELECT e FROM Usuario e WHERE e.email = :email", Usuario.class)
                    .setParameter("email", email).getSingleResult();
        } catch (NoResultException e) {
            return null;
        }
    }
```

# Desarrollo de servicios - API Rest

- ¿Que es una API Rest?
  - interfaz que permite la comunicación entre diferentes sistemas a través de la web
  - uso de estándares HTTP, como GET, POST, PUT y DELETE
  - realizar operaciones sobre recursos que se representan en formato JSON o XML.



# Desarrollo de servicios - API Rest

- Utilización de Jersey -> Framework de java para desarrollar servicios RESTFul

# Desarrollo de servicios - API Rest

- Implementación de todos los servicios

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@Operation(summary = "Creacion de un nuevo usuario")
@ApiResponses(value = {
    @ApiResponse(responseCode = "201", description = "Usuario creado",
        content = @Content(mediaType = "application/json",
        schema = @Schema(implementation = Usuario.class))),
    @ApiResponse(responseCode = "409", description = "Conflicto de datos")
})
public Response createUser(@Parameter(description = "Datos del nuevo usuario", required = true) UsuarioRequest usuarioRequest) {
```

```
@PUT
@Path("/{id}")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
@Operation(summary = "Actualización de un usuario")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "Usuario actualizado",
        content = @Content(mediaType = "application/json",
        schema = @Schema(implementation = Usuario.class))),
    @ApiResponse(responseCode = "404", description = "Usuario no encontrado")
})
public Response updateUser(@Parameter(description = "Datos del usuario a actualizar", required = true) UsuarioRequest usuarioRequest,
```

# Vistas del proyecto e integración con la API

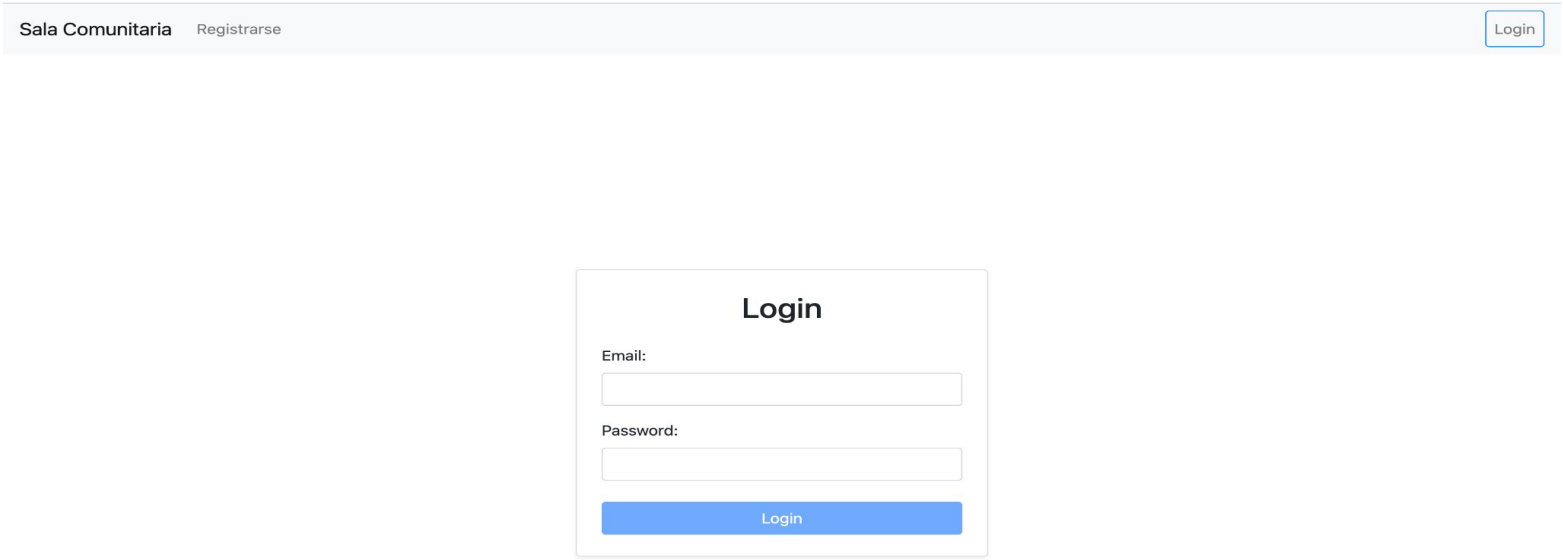
- Implementación de las vistas utilizando Angular

# Vistas del proyecto e integración con la API

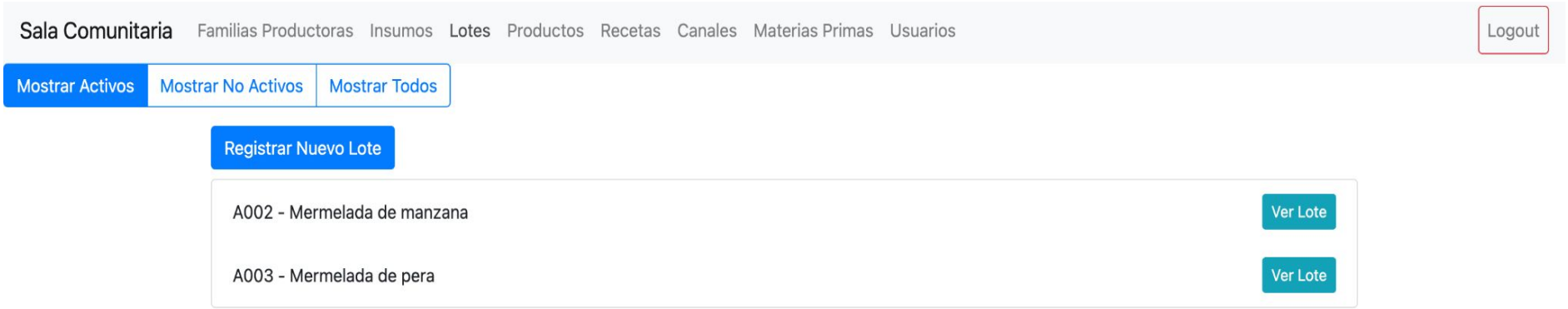
- Integración con la API Rest previamente desarrollada

```
createUser(user: UsuarioRequest): Observable<Usuario> {
  const headers = new HttpHeaders({ 'Content-Type': 'application/json' });
  return this.http.post<Usuario>(this.apiURL, user, { headers: headers })
    .pipe(
      catchError(this.handleError)
    );
}

getUser(id: number): Observable<UsuarioRequest> {
  return this.http.get<UsuarioRequest>(`${this.apiURL}/${id}`, { headers: this.headers });
}

getUsers(): Observable<Usuario[]> {
  return this.http.get<Usuario[]>(`${this.apiURL}/all`, { headers: this.headers });
}
```

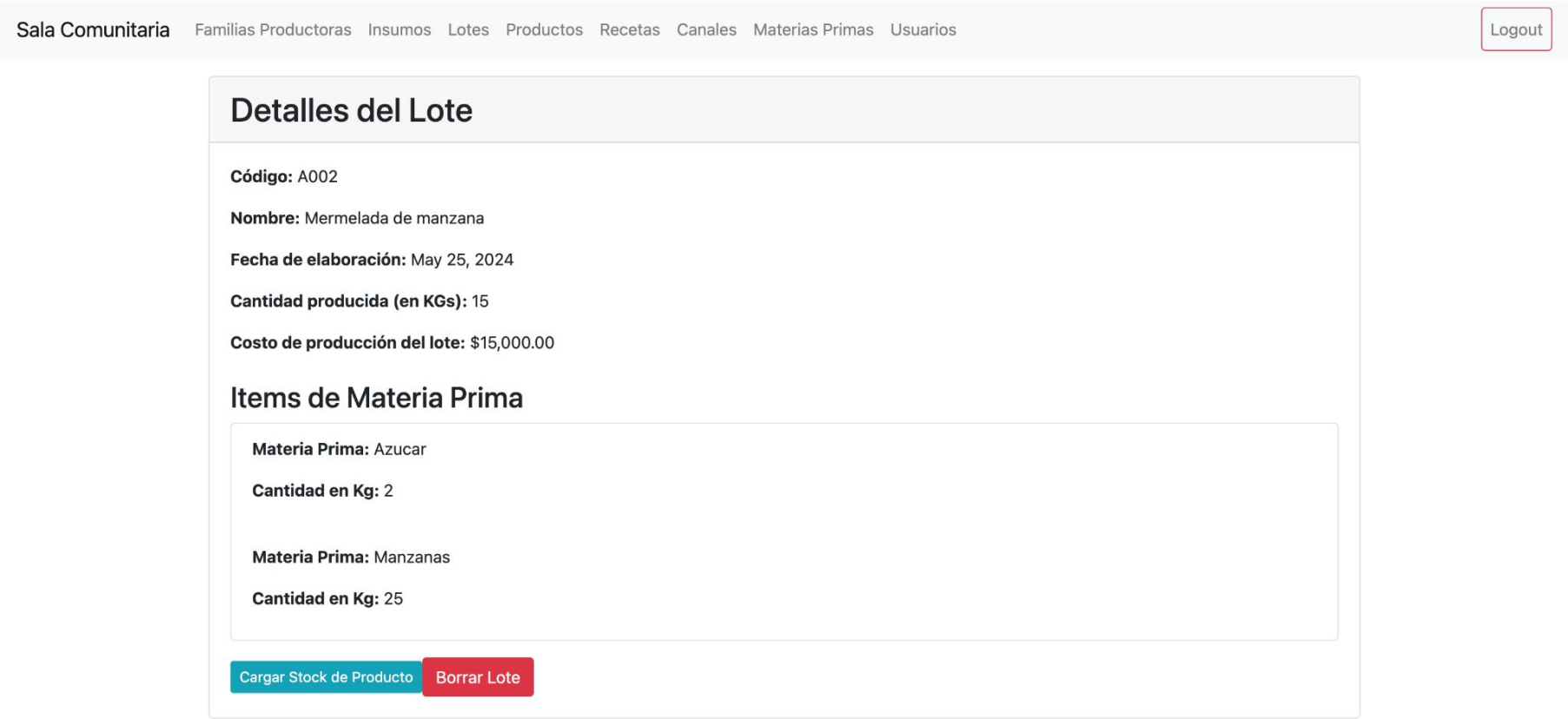
# Sistema Completo - Autenticación



# Sistema Completo - Lotes



# Sistema Completo - Lotes



# División del trabajo

# División de trabajo

# Herramientas utilizadas

# Herramientas



# Aprendizajes

# Aprendizajes

Base de datos



API REST

# ¿Preguntas?

# FIN

Grupo 9 - Novomiski, Herce, Quaranta