Documento de Arquitectura de Software

Evencos

Facultad de Ingeniería - Universidad Libre

Índice

1 Introducción

1.1 Propósito

El Documento de Arquitectura de Software presenta la arquitectura del sistema de gestión de eventos **EventHub** a través de diferentes vistas, cada una de las cuales ilustra un aspecto en particular del software desarrollado. El objetivo es brindar una visión comprensible y global del diseño general del sistema.

Luego de describir en profundidad la arquitectura del sistema Evencos, se presenta una implementación concreta como prueba de concepto de la arquitectura definida.

1.2 Alcance

El documento se centra en la vista lógica del sistema EventHub. Se mencionan aspectos fundamentales del resto de las vistas, omitiendo aquellas que no se consideren esenciales como la vista de procesos, debido a la simplicidad relativa del sistema.

En cuanto a los componentes externos que se mencionen (como servicios de correo, pasarelas de pago, etc.), se incluye una descripción apropiada con referencias donde se puede consultar más información.

1.4 Organización del Documento

El documento se desarrolla con base en la plantilla del artefacto *Software Architecture Document* del proceso RUP [2], adaptada al contexto del sistema de gestión de eventos.

La sección 2 introduce el modelo de representación de la arquitectura. Posteriormente se describen los objetivos, restricciones y las distintas vistas arquitectónicas del sistema EventHub.

2 Representación de la Arquitectura

El modelo propuesto por RUP [2] para representar la arquitectura utiliza el siguiente conjunto de vistas:

- **Vista de Casos de Uso**: Lista los casos de uso que representan funcionalidades clave del sistema, especialmente aquellas que impactan significativamente la arquitectura.
- **Vista Lógica**: Describe la estructura del sistema a nivel de subsistemas, capas o componentes.
- **Vista de Procesos**: En este proyecto no se considera relevante debido a la naturaleza monolítica del sistema.

- **Vista de Deployment**: Describe la distribución física del sistema sobre servidores, incluyendo el mapeo de los componentes lógicos.
- **Vista de Implementación**: Relaciona los componentes lógicos con los paquetes y clases reales del código.
- **Vista de Datos**: Describe el modelo de datos relacional, incluyendo las tablas principales del sistema.

3 Objetivos y Restricciones

EventHub se basa en un enfoque modular con las siguientes propiedades clave:

- **Diseño por componentes** con funcionalidades claras y específicas.
- **Desacoplamiento** entre módulos que permita su mantenimiento y escalabilidad.
- **Reutilización** de componentes en futuros sistemas similares.

3.1 Requerimientos Especiales

3.1.1 Interoperabilidad

EvenCos debe ser capaz de interoperar con servicios externos como APIs de correo electrónico, servicios de geolocalización y pasarelas de pago.

4 Vista de Casos de Uso

4.1 Introducción

Esta vista presenta un subconjunto de los casos de uso que definen funcionalidades centrales del sistema. Estos casos, junto con los requerimientos no funcionales, guían el diseño de la arquitectura.4.2 Identificación de los Casos de Uso relevantes para la arquitectura

1. Creación de Evento:
   - Permite a los organizadores crear nuevos eventos incluyendo título, descripción, fecha, ubicación y tipo.
   - Implica validación de formularios, integración con mapas, y almacenamiento en la base de datos.
2. Registro de Usuarios y Compra de Entradas
   - Implica manejo de autenticación, sesiones de usuario y conexión con pasarelas de pago (por ejemplo, PayPal o Stripe)
3. Notificaciones y Confirmaciones
   - Al registrarse a un evento, los usuarios deben recibir correos de confirmación

5 Vista Lógica

5.1 Introducción

En esta sección se presentan los sucesivos refinamientos que definen las diferentes unidades lógicas que componen la arquitectura del sistema **Evencos**.

El primer refinamiento realizado consiste en la descomposición del sistema en subsistemas o capas. Estas representan cortes verticales y horizontales del diseño del sistema. Cada subsistema agrupa funcionalidades relacionadas, con un alto grado de cohesión interna y bajo acoplamiento externo, y puede ser considerado como una unidad funcional casi autónoma.

Posteriormente, se explora la composición interna de cada subsistema, describiendo sus responsabilidades y relaciones principales.

Finalmente, se vinculan los casos de uso definidos en la sección anterior con los componentes arquitectónicos aquí descritos.

.

5.2 Descomposición en Subsistemas

La arquitectura lógica de EventHub sigue una estructura en **capas y módulos funcionales**, comúnmente utilizada en aplicaciones web modernas, lo que favorece la separación de responsabilidades, reutilización de componentes y mantenibilidad del sistema. Los subsistemas principales son:

1. Subsistema de Presentación (Frontend)

- **Responsabilidad**: Maneja la interacción con el usuario a través de una interfaz web amigable y responsiva.
- **Tecnologías sugeridas**: HTML5, CSS3, JavaScript, React.js o Vue.js.
- **Componentes clave**:
  - Módulo de visualización de eventos.
  - Módulo de formularios para creación y edición de eventos.
  - Módulo de búsqueda y filtrado.
  - Módulo de registro/login de usuario.

2. Subsistema de Lógica de Negocio (Backend)

- **Responsabilidad**: Procesa las reglas del negocio y coordina las operaciones entre los demás subsistemas.
- **Tecnologías sugeridas**: Node.js, Express, Django o Spring Boot.
- **Componentes clave**:
  - Controlador de eventos.
  - Controlador de usuarios.
  - Controlador de pagos.
  - Controlador de notificaciones.

3. Subsistema de Persistencia de Datos

- **Responsabilidad**: Gestiona el acceso a los datos persistentes y define la estructura de las entidades.
- **Tecnologías sugeridas**: PostgreSQL, MySQL o MongoDB.
- **Componentes clave**:
  - Entidad Evento.
  - Entidad Usuario.
  - Entidad Entrada (ticket).
  - Entidad Transacción.

4. Subsistema de Autenticación y Autorización

- **Responsabilidad**: Maneja el control de acceso y seguridad del sistema.
- **Funciones clave**:
  - Registro y login de usuarios.
  - Control de roles (usuario, organizador, administrador).
  - Generación y validación de tokens (JWT).

5. Subsistema de Integraciones Externas

- **Responsabilidad**: Permite la comunicación con servicios externos.
- **Servicios conectados**:
  - API de envío de correos (por ejemplo, SendGrid).
  - Pasarela de pago (por ejemplo, Stripe o PayPal).
  - API de mapas (por ejemplo, Google Maps).

5.3 Descripción de los Subsistemas

- Subsistema de Gestión de Eventos: Este subsistema proporciona las herramientas necesarias para que los organizadores puedan **crear, editar y eliminar eventos** de forma amigable e intuitiva. Incluye formularios web validados, funciones de carga de imágenes, definición de ubicaciones (a través de APIs de mapas), fechas, cupos, precios y categorías. También transforma los datos ingresados en estructuras internas listas para ser persistidas en la base de datos y posteriormente consumidas por otros módulos del sistema (como la vista de usuario final o el sistema de entradas).
- Subsistema de Visualización y Navegación: Es responsable de **mostrar los eventos activos al público** general. Filtra por ubicación, fecha, categoría y popularidad, y proporciona detalles completos del evento (descripción, organizador, precio, lugar, etc.). Permite a los usuarios registrados **inscribirse o comprar entradas**, interactuar con los eventos (por ejemplo, marcar como favoritos) y recibir recomendaciones personalizadas.
- Subsistema de Gestión de Usuarios: Este módulo maneja el **registro, autenticación y gestión de perfiles de usuario**. Soporta múltiples tipos de usuarios (organizadores, asistentes y administradores).
  Incluye validación de credenciales, recuperación de contraseñas, manejo de sesiones, y generación de tokens de acceso (por ejemplo, usando JWT).
- Subsistema de Notificaciones: Este subsistema **gestiona la comunicación con los usuarios** mediante correos electrónicos u otras vías (por ejemplo, notificaciones push). Se encarga de enviar confirmaciones de compra, recordatorios de eventos, notificaciones de cambios o cancelaciones, y newsletters.

5.4 Diseño de Subsistemas

5.4.1 Definición de Procesos

El diseño de este subsistema sigue el patrón Modelo-Vista-Controlador (MVC). Las principales clases y componentes incluyen:

- **Controladores de Eventos**: Interceptan las solicitudes HTTP relacionadas con eventos, validan los datos y delegan en el servicio correspondiente.
- **Servicios de Negocio**: Contienen la lógica relacionada con la validación de reglas del evento (fechas, cupos, categorías), así como transformaciones previas al almacenamiento.
- **Modelos de Evento**: Representan las entidades almacenadas en la base de datos, incluyendo relaciones con el organizador, lugar, asistentes y entradas.
- **Repositorio de Eventos**: Implementa la lógica de acceso a datos, ya sea a través de ORM o consultas SQL personalizadas.

La salida de este subsistema es una representación estructurada del evento, lista para ser consumida por los módulos de visualización, pago y notificaciones.

6 Vista de Deployment

6.1 Introducción

Esta sección describe la configuración física sobre la cual se realiza el despliegue del sistema web de gestión de eventos y la infraestructura necesaria para su correcta operación. Se consideran aspectos como los servidores involucrados, la red de comunicación y los elementos de soporte que hacen posible la ejecución del sistema.

La arquitectura está pensada para un entorno de producción en la nube, con acceso tanto desde la web pública como desde interfaces administrativas privadas. Se consideran entornos separados para desarrollo, pruebas y producción.

6.2 Distribución y Deployment

El despliegue del sistema contempla los siguientes nodos principales:

- **Cliente Web**: Navegador desde el cual los usuarios acceden a la aplicación (asistentes o administradores). Se comunican mediante HTTPS con el backend.
- **Servidor Web (Frontend)**: Servidor responsable de entregar la aplicación web al navegador (por ejemplo, una aplicación React, Vue o Angular). Puede ser servido desde un CDN (Content Delivery Network) o servidor Nginx/Apache.
- **Servidor de Aplicaciones (Backend)**: Encargado de ejecutar la lógica de negocio. Implementado en tecnologías como Node.js, Django, Spring Boot u otro stack web. Expone una API RESTful o GraphQL.
- **Servidor de Base de Datos**: Motor relacional (por ejemplo, PostgreSQL o MySQL) encargado de almacenar eventos, usuarios, entradas, transacciones, etc.
- **Pasarela de Pagos**: Servicio externo (como Stripe, PayPal o MercadoPago) con el que el backend se comunica para procesar transacciones.
- **Servidor de Correos/Notificaciones**: Utilizado para enviar correos electrónicos automáticos. Puede ser un servicio externo (SendGrid, Amazon SES) o interno.
- **Repositorio de Archivos**: Para almacenamiento de imágenes u otros archivos subidos por los usuarios. Puede ser un servicio como Amazon S3 o un servidor FTP.

La comunicación entre los nodos está protegida con cifrado TLS/HTTPS. El sistema está desplegado sobre una infraestructura en la nube (como AWS, Azure o Google Cloud), usando contenedores (Docker) y orquestación (Kubernetes o similar) para garantizar escalabilidad y alta disponibilidad.

.

7 Arquitectura del Sistema Batuta

7.1 Introducción

Sobre la base arquitectónica definida anteriormente, esta sección describe la **realización del prototipo del sistema web de gestión de eventos**, construido como una prueba de concepto aplicando la metodología RUP.

Este sistema se desarrolló a partir de la integración de diversas tecnologías web modernas, considerando principios de separación de capas, modularidad y escalabilidad. La descripción se organiza repasando las vistas arquitectónicas presentadas, indicando para cada una:

- Los componentes implementados.
- Las tecnologías utilizadas.
- Las decisiones clave de diseño.
- Las herramientas aplicadas en el desarrollo.

En este proyecto no fue necesario adaptar los casos de uso iniciales, ya que el sistema prototipo fue diseñado específicamente para satisfacerlos.

Las decisiones tecnológicas principales fueron las siguientes:

- **Lenguaje de programación backend**: Node.js con Express (alternativamente, Python con Django o Java con Spring Boot).
- **Base de datos**: PostgreSQL, con ORM Sequelize (o SQLAlchemy / Hibernate).
- **Frontend**: React con Tailwind CSS.
- **API**: RESTful, autenticación basada en tokens JWT.
- **Notificaciones**: Integración con SendGrid para correos electrónicos.
- **Pasarela de pago**: Stripe (modo sandbox en entorno de pruebas).
- **Despliegue**: Docker + Docker Compose, con vistas a usar Kubernetes en producción

7.2 Vista Lógica

La vista lógica del sistema web de gestión de eventos muestra los **subsistemas implementados** y cómo interactúan entre sí para dar soporte a los casos de uso definidos previamente (como crear eventos, registrarse, comprar entradas, etc.).

A continuación, se describen los principales módulos lógicos implementados:

● **Módulo de Autenticación y Usuarios**

Encargado de la gestión de usuarios (registro, inicio de sesión, recuperación de contraseña, roles). Implementa autenticación mediante **JSON Web Tokens (JWT)** y autorización basada en roles (usuario común, organizador, administrador).

- **Tecnologías**: Node.js, Express, JWT, bcrypt para hash de contraseñas.
- **Interfaz con BD**: ORM Sequelize sobre PostgreSQL.

● **Módulo de Gestión de Eventos**

Permite a los organizadores crear, modificar o eliminar eventos, y a los usuarios visualizar eventos disponibles. Contempla atributos como título, fecha, lugar, aforo, tipo de entrada, etc.

- **Lógica**: Validaciones de reglas de negocio (eventos pasados no editables, campos obligatorios).
- **Interfaz de consulta**: Filtrado por fecha, categoría, ubicación.

● **Módulo de Entradas y Asistencias**

Administra la disponibilidad de entradas por evento, su compra, asignación y verificación. Genera un código QR para la entrada virtual, que puede ser escaneado en el ingreso al evento.

- **Tecnologías**: Librerías de generación QR, PDF (por ejemplo qrcode, pdfkit).
- **Lógica de negocio**: Asegura la no sobreventa (transacciones atómicas).

● **Módulo de Pagos**

Orquesta el proceso de pago con la pasarela externa (Stripe o MercadoPago). Inicia el pago, maneja la validación del mismo y genera las entradas virtuales una vez confirmado.

- **Integración**: API de Stripe en modo sandbox.
- **Seguridad**: Webhooks verificados y manejo seguro de tokens.

● **Módulo de Notificaciones**

Envía correos electrónicos de confirmación, recordatorios de eventos y recuperación de contraseñas. Utiliza plantillas HTML para mensajes amigables.

- **Servicios externos**: SendGrid o Amazon SES.
- **Eventos disparadores**: Registro, compra confirmada, cambio de contraseña.

● Módulo de Administración

Permite a los administradores monitorear la actividad del sistema, moderar eventos, gestionar reportes y generar estadísticas.

- **Funcionalidad**: Panel de control con gráficos (por ejemplo, Chart.js), sistema de logs de actividad.

Relación entre los módulos

Los módulos están organizados en una arquitectura de tipo **MVC (Modelo-Vista-Controlador)** en el backend, donde:

- Los **modelos** representan las entidades del sistema (usuarios, eventos, entradas, pagos).
- Los **controladores** encapsulan la lógica de negocio.
- Las **rutas** definen los endpoints de la API RESTful que son consumidos por el frontend.

La interacción entre subsistemas se basa en **llamadas API** internas dentro del backend o mediante eventos asincrónicos (como webhooks de Stripe o envío de correos).

Vista de Deployment

La **vista de deployment** describe la configuración física en la que fue desplegado el prototipo del Sistema Web de Gestión de Eventos. Esta vista incluye los **nodos físicos o virtuales**, la **infraestructura utilizada**, y la **distribución de los componentes de software** sobre dicha infraestructura.

Escenario de Deployment del Prototipo

El sistema fue desplegado en un entorno típico de desarrollo y pruebas en la nube, utilizando contenedores y servicios gestionados. A continuación se detalla la arquitectura física:

● Nodo 1: Cliente Web (Frontend)

- **Ubicación**: Navegador web del usuario final.

- **Tecnologías**: React.js, Tailwind CSS, Axios.
- **Funcionalidad**: Presenta la interfaz gráfica, permite la interacción con el usuario, consume los servicios del backend vía API REST.
- **Comunicación**: HTTPS hacia el servidor backend (API Gateway/Nodo 2).

● **Nodo 2: Servidor Backend (API REST)**

- **Ubicación**: Contenedor Docker desplegado en un servicio PaaS (ej. Railway, Render o Heroku).
- **Tecnologías**: Node.js con Express.js.
- **Funcionalidad**:
    - Manejo de autenticación y sesiones.
    - Gestión de usuarios y eventos.
    - Procesamiento de pagos.
    - Envío de notificaciones.
- **Dependencias**:
    - Motor de base de datos (Nodo 3).
    - Pasarela de pago (Nodo 4).
    - Servicio de correos (Nodo 5).
- **Seguridad**: Certificados SSL, tokens JWT, control de CORS.

● **Nodo 3: Base de Datos**

- **Ubicación**: Servicio gestionado de PostgreSQL (por ejemplo, ElephantSQL o Supabase).
- **Tecnologías**: PostgreSQL 14+.
- **Funcionalidad**: Almacena datos persistentes del sistema (usuarios, eventos, entradas, pagos, logs).
- **Conectividad**: Acceso restringido por IP, autenticación cifrada.

● **Nodo 4: Pasarela de Pago**

- **Ubicación**: Servicio externo (Stripe o MercadoPago).
- **Funcionalidad**: Procesa los pagos realizados por los usuarios.
- **Comunicación**:
    - Desde backend: solicitudes HTTPS para iniciar pagos.
    - Hacia backend: webhooks para confirmación de transacciones.

● **Nodo 5: Servicio de Correos**

- **Ubicación**: Plataforma de correos (SendGrid, Mailgun, o Amazon SES).
- **Funcionalidad**: Envía correos de confirmación, recuperación de cuenta y recordatorios de eventos.
- **Comunicación**: Llamadas HTTPS autenticadas desde el backend.

8 Referencias

[1] Proyecto de Grado Batuta – *Generador de Aplicaciones Orquestadoras. Glosario*. Facultad de Ingeniería, Universidad de la República, Uruguay, 2005. Disponible en: http://www.fing.edu.uy/~pgsoasem/documentos/PG-P2005_0026-Glosario.pdf

[6] [2] Rational Unified Process. Rational Software, IBM, 2003. Disponible en: http://www-306.ibm.com/software/awdtools/rup/

[7] [3] Proyecto de Grado Batuta – *Descripción del Modelo Batuta*. Facultad de Ingeniería, Universidad de la República, Uruguay, 2005. Disponible en: http://www.fing.edu.uy/~pgsoasem/documentos/PG-P2005_0026DescripcionModeloBatuta.pdf

[8] [4] Proyecto de Grado Batuta – *Estado del Arte*. Facultad de Ingeniería, Universidad de la República, Uruguay, 2005. Disponible en: http://www.fing.edu.uy/~pgsoasem/documentos/PG-P2005_0026EstadoDelArte.pdf

[9] [5] Proyecto de Grado Batuta – *Especificación Complementaria del Modelo Batuta*. Facultad de Ingeniería, Universidad de la República, Uruguay, 2005. Disponible en: http://www.fing.edu.uy/~pgsoasem/documentos/PG-P2005_0026EspecificacionComplementaria.pdf
[10] [6] *Process Modeler for Microsoft Visio™* – ITP Commerce. Disponible en: http://www.itp-commerce.com/
[11] [7] *ActiveBPEL* – Active Endpoints. Disponible en: http://www.active-endpoints.com/
[12] [8] *Jena Framework*. Disponible en: http://jena.sourceforge.net/index.html
[13] [9] *OWL-S/UDDI Matchmaker*. Disponible en: http://www.daml.ri.cmu.edu/matchmaker/