UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
Año 2015 - 2do Cuatrimestre

TALLER DE PROGRAMACIÓN III (75.61)

TRABAJO PRÁCTICO N.º 3
TEMA: Cloud Programming: Google App Engine
FECHA: 29/10/2015

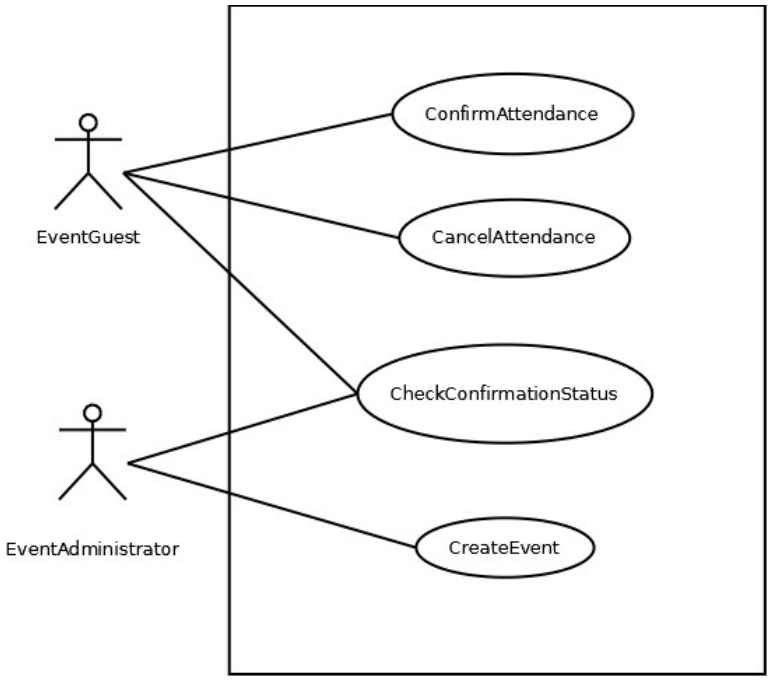
Autor	Padrón
Darío Eduardo Ramos	84885

Objetivos del sistema

- Confirmar la asistencia a un evento de parte de un conjunto de invitados, controlando el máximo de vacantes del mismo.
- Permitir en todo momento verificar si un invitado confirmó o no su asistencia al evento. Esto implica maximizar la disponibilidad del sistema, por ello se optó por una plataforma Cloud.
- Diseñar los componentes de manera que el sistema pueda reutilizarse para eventos futuros.
- El sistema debe soportar diferentes niveles de carga. Realizar pruebas y documentarlas.

Vista de casos de uso

Diagrama de casos de uso



Sólo ***CreateEvent*** requiere derechos de administrador. Cualquier tipo de usuario puede revisar si se confirmó la asistencia o no.

Especificación de casos de uso

Caso de uso	CreateEvent
Actores	Primario: EventAdministrator
Pre-condiciones	
Descripción	Flujo principal 1. EventAdministrator solicita crear un nuevo evento. 2. Sistema pide detalles del evento: fecha, lugar, duración y cantidad de vacantes. 3. EventAdministrator ingresa los datos del evento. 4. Sistema registra nuevo evento en el sistema.
Post-condiciones	Nuevo evento registrado en el Sistema

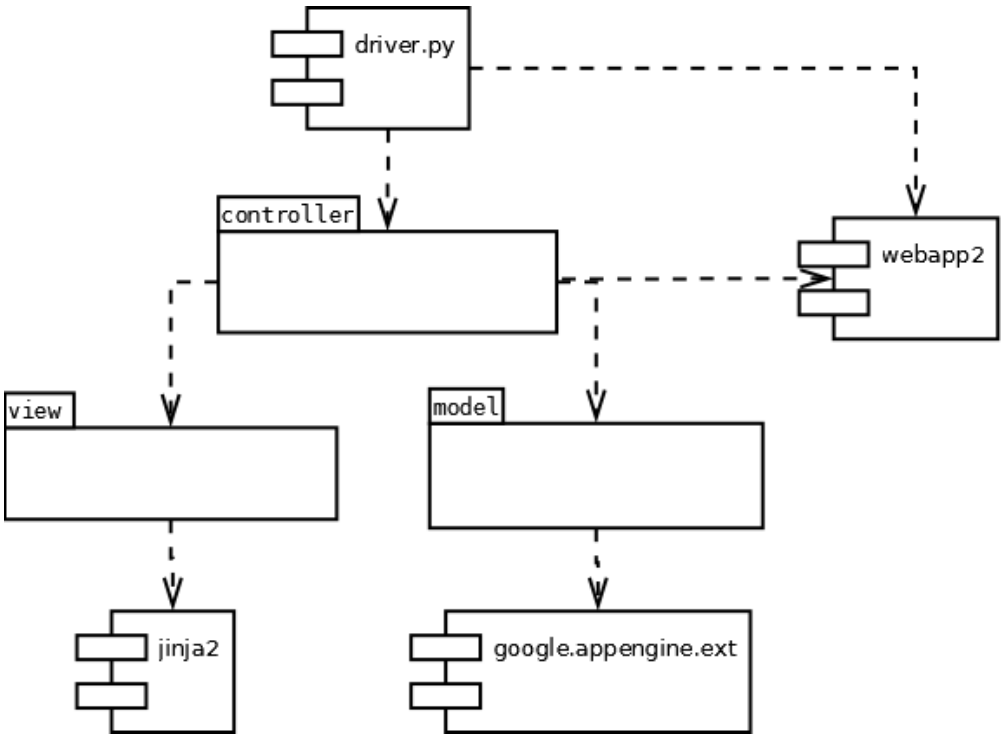
Caso de uso	ConfirmAttendance
Actores	Primario: EventGuest
Pre-condiciones	Evento creado
Descripción	Flujo principal: 1. EventGuest solicita inscribirse a evento. 2. Sistema solicita datos de EventGuest: Email, Nombre y Apellido, y Empresa para la que trabaja. 3. EventGuest completa sus datos y los entrega a Sistema. 4. Sistema deja registrada la asistencia de EventGuest al evento. 5. Sistema decrementa la cantidad de vacantes del evento. 6. Sistema confirma a EventGuest que ha quedado asentada su asistencia al evento. Flujo alternativo A1: (no hay vacantes) A1.1. Sistema informa a EventGuest que no hay más vacantes para el evento. Flujo alternativo A2: (cliente ya registrado para el evento) A2.1. Sistema responde a EventGuest que ya se ha registrado al evento.
Post-condiciones	EventGuest confirmado para asistir al evento (si había vacantes). Vacantes del evento decrementadas en uno.

Caso de uso	CancelAttendance
Actores	Primario: EventGuest
Pre-condiciones	Evento creado
Descripción	Flujo principal: 1. EventGuest solicita cancelar su asistencia al evento. 2. Sistema da de baja a EventGuest de la lista de invitados que han confirmado sus asistencia. 3. Sistema aumenta la cantidad de vacantes del evento en 1. 4. Sistema informa a EventGuest que su pedido ha sido procesado exitosamente. Flujo alternativo A1: (EventGuest no ha confirmado su asistencia al evento) A1.1. Sistema informa a EventGuest que no es necesario cancelar su asistencia porque nunca la había confirmado.
Post-condiciones	EventGuest eliminado de la lista de invitados que había confirmado su asistencia al evento. Vacantes del evento incrementadas en 1.

Caso de uso	CheckConfirmationStatus
Actores	Primario: EventGuest o EventAdministrator
Pre-condiciones	Evento creado
Descripción	Flujo principal: 1. EventGuest solicita información del estado de su asistencia al evento. 2. Sistema responde a EventGuest si ha confirmado su asistencia al evento o no.
Post-condiciones	EventGuest recibe la información solicitada. No hay cambios al sistema.

Vista lógica: Diagrama de paquetes

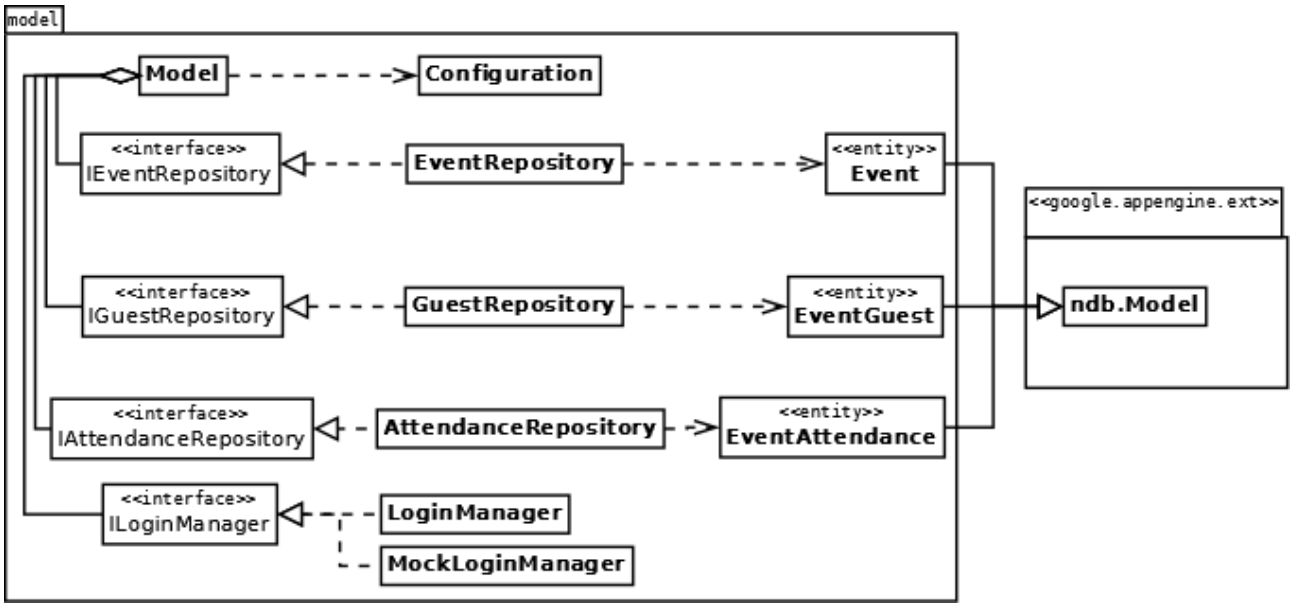
Diagrama de paquetes general



Se implementó una arquitectura MVC manualmente, a fin de aislar las dependencias con las diversas tecnologías usadas. Por ejemplo, sólo el modelo usa la API de Google para acceder a la Data Store, sólo la vista usa jinja2 y sólo controlador y driver usan webapp2. Esta separación es esencial para facilitar el mantenimiento de la aplicación.

Diagramas de paquetes y clases

Paquete model:



El punto de entrada, o fachada en el sentido del patrón Facade, es la clase Model. La misma provee acceso a las diferentes interfaces del sistema: los repositorios, y el login manager. Los repositorios abstraen el acceso a datos. Las entidades también se usan fuera del paquete.

Paquetes controller y view: Ambos son colecciones de clases o páginas independientes entre sí, por lo que diagramarlos no aporta mucho. Los controladores y sus páginas asociadas son:

- MainController -> index.html
- ConfirmAttendanceController -> confirm.html
- AdminLoginController -> admin_login.html
- CreateEventController -> create.html
- FailedLoginController -> failed_login.html
- SelectEventController -> select_event.html
- CancelAttendanceController -> cancel.html
- CheckAttendanceController -> check.html

La excepción a este caso es el módulo environment del paquete controller. El mismo contiene una instancia del entorno Jinja2, y una instancia de la clase Model. Esto evita usar el patrón Singleton (que el autor de este informe considera un anti patrón).

Descripción de clases

Descripción de clases más importantes

model.Model
Responsabilidad: Funcionar como Facade, proveyendo interfaces a los repositorios y demás funcionalidades del modelo.
Atributos principales: Una instancia de cada repositorio, y una instancia de LoginManager.
Métodos principales: Los gets de los repos y de loginManager.

driver (módulo)
Responsabilidad: Punto de entrada de la aplicación, mapea rutas a controladores.
Atributos principales: (como variable del script): una instancia de webapp2.app
Métodos principales: n/a. Como script, instancia app y configura las rutas.

EventRepository[1]
Responsabilidad: Abstraer el acceso a instancias de Event de la tecnología de persistencia adoptada.
Atributos principales: Stateless
Métodos principales: **def create(self, aName, aDate, aVacancies)**: Crea y persiste una instancia de Event con el estado dado por los parámetros. Si ya existe, actualiza sus campos. **def getAll(self)**: Devuelve todas las instancias persistidas. **def getByName(self, eventID)**: Busca un evento por nombre. **def update(self, event)**: Actualiza el estado de un evento en la base de datos.

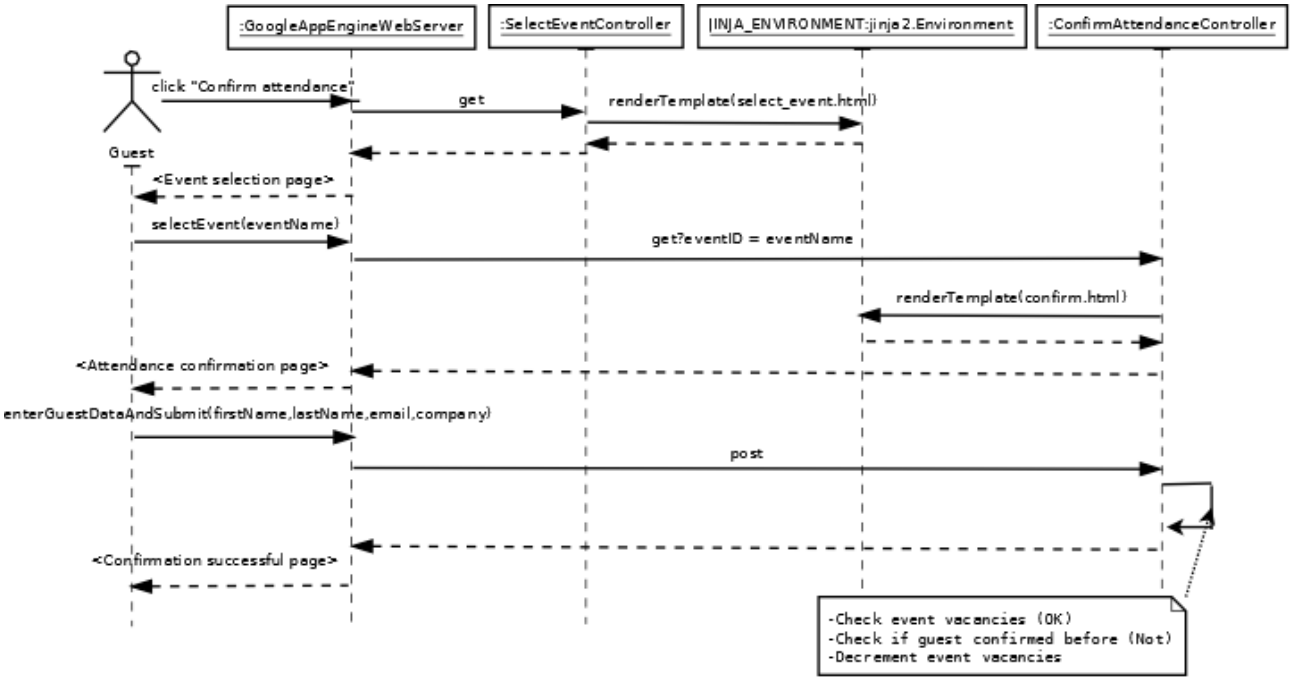
controller.ConfirmAttendanceController
Responsabilidad: Permitir a un Guest confirmar la asistencia a un Evento particular.
Atributos principales: Stateless
Métodos principales: **def get(self)**: Renderiza la página de confirmación (confirm.html), lo cual permite al Guest ingresar sus datos. ***Requiere como parámetro en el query string el nombre del evento***. def post(self): Se fija si hay vacantes para el evento. Si no hay, informa al Guest y termina. Si hay,

1 Las otras clases de repositorios son análogas y por ende son omitidas

usa los datos ingresados por el Guest para instanciar un EventGuest, y persistirlo (si ya existía, se actualizan sus campos). Luego, se fija si el Guest ya se había inscripto o no al evento. Si ya lo había hecho, lo informa y concluye. Si no, decrementa las vacantes del evento y notifica al Guest.

controller.environment (módulo)
Responsabilidad: Contener instancias de objetos necesarios para los controladores, permitiendo que los mismos carezcan de estado propio, y evitando usar el patrón Singleton.
Atributos principales: (como variables del script): una instancia del entorno de Jinja, y una instancia de model.Model.
Métodos principales: n/a

Vista de procesos: Diagrama de secuencia



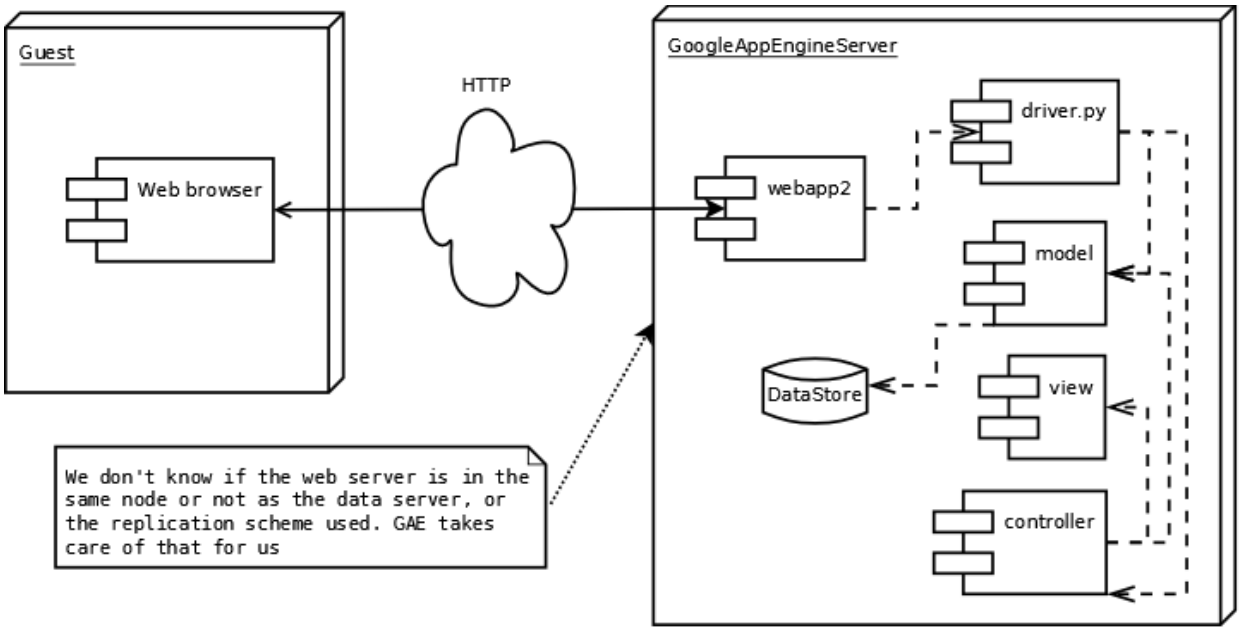
Este diagrama corresponde al escenario más común: un cliente confirma su asistencia a un evento, habiendo vacantes para el mismo en ese momento, y asumiendo que no confirmó su asistencia previamente.

El Guest hace click en el enlace de "Confirm attendance", lo cual es derivado por el web server de GAE a SelectEventController como un HTTP get (porque así se configuró en driver.py). Dicho controlador, en su handler para el método get, renderiza el template de la página select_event.html vía Jinja, lo cual le permite insertar dinámicamente la lista de eventos disponibles. La página resultante es enviada al cliente y renderizada en su browser.

A continuación, el cliente selecciona el evento para el cual desea confirmar su asistencia. Un proceso análogo al anterior, se produce, sólo que esta vez el get es atendido por ConfirmAttendanceController::get.

El cliente recibe la página de confirmación de asistencia, en la cual debe ingresar los datos. Al terminar, hace click en "Confirm Attendance", lo cual envía sus datos al método post de ConfirmAttendanceController. Allí, se verifica que haya vacantes para el evento (en este escenario, hay), luego que el Guest no esté inscripto (en este escenario no lo está), y finalmente se decrementan las vacantes del evento en uno, se registra al Guest como inscripto y se lo notifica enviándole una página pertinente.

Vista de despliegue: Diagrama de robustez



El cliente se comunica con el servidor de Google vía HTTP. Dentro de la nube de Google, no hay forma de saber dónde está el web server, dónde la base de datos y qué replicación se usa. Es opaco a nuestra aplicación.

Load Testing

Herramienta seleccionada: JMeter

Criterios para su elección:

- Recomendación de colegas/compañeros.
- No requiere instalación.
- Open source y gratuito.

Configuración del Test Plan 1:

- Thread Group: 50 users, ramp-up time de 10 segundos, 1 loop
  - HTTP Request sampler
    - Server name: http://event-management-1113.appspot.com
    - Path: /confirm
    - Method: POST
    - Parameters
      - event_guest_first_name: ${user_counter}
      - event_guest_last_name: Testman
      - event_guest_email: ${user_counter}@gmail.com
      - event_guest_company: Testers United
      - event_id: Induction4
  - User Counter:
    - Name: User counter
    - Start 1
    - Increment: 1
    - Maximum: 50
    - Reference name: user_counter
- HTTP Cookie Manager
- HTTP Request Defaults
  - Server name: http://event-management-1113.appspot.com
  - Path: confirm

Resultados: 138 ms promedio de latencia, para 2 corridas (ambas con la configuración detallada)

Planificación

(Veáse sprint_backlog.pdf)