Bitgray Practical Test - Jeyson Molina (Sept. 2016)

Instalación

(Demo corriendo en: http://45.55.138.14:9900/ (http://45.55.138.14:9900/))

- Prerrequisitos: Django (1.9), python MySQLdb, MySQL server
- El proyecto django se encuentra en project/
- Configurar entorno MySQL en env.py
- Realizar migración: python migrate.py
- Realizar migración de modelos Django: python manage.py makemigrations . python manage.py migrate
- Crear superusuario: python manage.py createsuperuser
- Iniciar: python manage.py runserver 8000

Soluciones

1. Crear y compartir un repositorio con esta prueba

   https://github.com/jeysonmc/test_bitgray (https://github.com/jeysonmc/test_bitgray)

2. Realizar la migración de dicha base de datos y realizar su respectivo CRUD para cada tabla

   Implementado en project/migrate.py para crear la DB y migrar los datos del archivo SQL.

   Para los CRUDS se usa el ORM de Django, los modelos se definen en project/app/models.py
   Los administradores que implementan una interfaz para el CRUD de cada modelo se definen en project/app/admin.py
   Para acceder a estos: Administradores (/admin) (user: admin , pass: testbitgray o el superusuario definido)

3. Crear el API necesaria para que otra aplicación pueda realizar transacciones. (Consultar y crear)

   API (tipo RESTful, retorna json) Implementada en project/app/views.py APIListCreate . Es una vista genérica que se configura con un modelo (ver project/urls.py)
   Ej API:
   (Se recomienda instalar jsonview (https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc) para Google chrome para visualizar)
   - Listado de compras: /api/compras/ (/api/compras/)
   - Detalle de compra: /api/compras/19/ (/api/compras/19)
   - Crear compra: `curl --data "precio=5000&cliente=1&producto=1" localhost:8000/api/compras/` (usando POST request, cambiar localhost:8000 por URL del servidor)
   Otras entradas: /api/clientes/, /api/productos/, /api/sedes/

4. Crear una interfaz para realizar compras

   Implementado en project/app/views.py (ComprasView) y la lógica front-end en project/app/templates/compras.html
   Interfaz compras (/compras/)

5. Crear un pequeño formulario donde se escriba el documento del cliente y traer en forma de factura sus compras

   Implementado en app/views.py(FacturaView) y la logica front-end en project/app/templates/factura.html

   Interfaz facturas (/facturas/)

6. Cada vez que se realice select, update o delete; se debe ingresar el registro en la tabla log, describiendo el tipo de movimiento y algún dato que crea necesario o de gran ayuda para una bitácora

   Implementado el log de transacciones SELECT en project/project/sql_logger.py Y el UPDATE, DELETE, INSERT de los modelos usando django signals en project/app/models.py (signal_update, signal_delete)
   - Para acciones UPDATE se almacena el valor anterior y nuevo de la fila alterada.
   - Para acciones DELETE se almacena el valor de la fila eliminada.

   Ver logs (/admin/app/log/)

7. Bajo la estructura de negocio actual, ¿Que propondría usted para realizar una numeración en la facturación, sin afectar o crear nuevos campos?

   `CODIGO = [fechaAño+fechaMes+fechaDia+fechaHora+fechaMinuto+fechaSegundo+id_sede+id_producto+id_cliente]`
   Ej: fecha: 2015,01,01,12,00
   id_sede: 1
   id_produdcto: 2
   id_cliente: 3
   Codigo: 201501011200123

8. Plantear herramientas y tecnologías necesarias para su publicación, seguridad y mantenimiento de esta aplicación, bajo el supuesto de una concurrencia diaria de 500.000 a 1.000.000 usuarios.

   Publicación
   - Caching: Mecanismos para el cacheo de respuestas del servidor para reducir el uso de peticiones a la BD y costo de procesamiento en general en el servidor. Tecnologías: django cache framework, redis y/o memcached para cachear en la memoria RAM del servidor para disminuir el tiempo de respuesta.
   - Mover contenidos estaticos a hardware/infraestructuras dedicadas a estos. Tecnologías: Amazon S3, Cloudflare.
   - Optimización de queries a la BD, estrategias de indexación en las tablas según el tipo de consulta (Ej: indexación de clientes.documento para la búsqueda de factura por documento).
   - Despliegue en instancias elásticas Amazon EC2, uso de servidor Nginx + UWSGI + docker

   Mantenimiento
   - Manejo de versiones (GIT), manejo de branches separados para desarrollo y para corrección de errores (hotfixes)
   - Mecanismos de automatización de despliegue de cambios en el codebase (Fabric)
   - Manejo de al menos dos instancias de la solución para mantener el uptime del servicio al 100% mientras se actualizan las instancias.
   - Implementar logs de todos los servicios/aplicaciones en el servidor para efectos de debug y propuesta de soluciones

   Seguridad
   - Sistema de logging en la bd de todas las transacciones realizadas
   - Replicación y/o backup incremental de la base de datos de forma periódica.
   - Almacenamiento de las claves/contraseñas en el sistema keyring del servidor
   - Protección del servidor, sistema de permisos adecuado, deshabilitar logueo SSH por contraseña, deshabilitar

logueo root. Sistemas auditores de autenticación (ej: fail2ban)
- Fragmentación del stack (Servidor http, BD, app) en varios contenedores docker
- Implementar capa de cifrado SSL (HTTPS)
- Mecanismos de seguridad contra DDOS

9. Crear un plan de escabilidad al momento de tener un alto volumen de datos, escrituras y lecturas en disco

    - Habilitar todos los sistemas de caching que se tengan disponibles
    - Escalamiento vertical del servidor. Aumentar su capacidad de memoria y procesamiento
    - Escalamiento horizontal. Clonar instancias y utilizar una aproximación de balanceo de carga para dividir el esfuerzo entre los servidores.

10. Automatizar un reporte semanal, que sea enviado a un correo electrónico y contenga los siguientes cálculos...

    Implementado en project/weekly_report.py (Ver cuenta gmail de prueba: testbgray.jmolina@gmail.com pass: testbgray2016) Enviar reporte de prueba (/enviar-reporte/)

Contacto: Jeyson Molina - jeyson.mco@gmail.com