Answers

Roger González

December 9, 2019

Contents

1 Arquitectura **1**

2 Diseño **3**

3 Cosas que faltaron **3**

1 Arquitectura

1. Sabiendo que la escalabilidad y la performance son los dos atributos de calidad más importantes, proponga una arquitectura para resolver el problema, explicando cómo la misma tiene en cuenta los atributos de calidad mencionados.

    - Usaría Python (en el ejemplo, Django REST Framework) para servir todo via REST. Para aprovechar la escalabilidad, usaría un módulo en comun (en el ejemplo llamado "core") para guardar las funciones y/o modelos que pueden usar los demás posibles modelos de la solución. En el caso del performance, Python ha demostrado ser un lenguaje rápido en la ejecución de trabajos en multithread, el compilador es mas amigable que algunos otros y es un lenguaje multipropósito, lo que nos puede ayudar a agregar multiples funcionalidades (scripts) en el futuro

2. Explique cómo almacenaría los datos para resolver el problema. Tenga en cuenta que puede seleccionar más de un tipo de almacenamiento

para resolver lo mejor posible cada uno de los requerimientos funcionales.

- Sugiero utilizar una base de datos SQL ya que los datos son muy estructurados, eso ayudaría con las búsquedas por ID porque la data estaría relacionada via llaves primarias

3. Mencione cuál le parece la mejor opción para comunicar los eventos mencionados a otros servicios en las condiciones mencionadas.

- Websockets. Es robusto, es conocido y la comunidad lo soporta muy bien.

4. En base a la arquitectura propuesta en la pregunta anterior, seleccione qué tecnologías utilizaría para implementar la solución, tanto a nivel de programación, almacenamiento y comunicación entre servicios. Como dato adicional se sabe que la empresa favorece las tecnologías libres y de código abierto frente a las propietarias y con costo de licenciamiento.

- Mi respuesta seguramente va a estar dirigida por mi trabajo, pero yo usaría el siguiente stack:
  - Para el API usaría Python con algun framework de desarrollo rápido, como Django, Flask o Tornado.
  - Para la persistencia de datos, usaría Postgres o MySQL. Como mencioné antes, los datos son estructurados por eso considero mejor una base de datos SQL.
  - Para el almacenamiento iría a un PasS. No veo necesario correr una instancia de Amazon EC2 o cualquier otro VPS solo para este pequeño módulo. Puede tener infinitos datos, pero al final es un solo modelo. Un PasS como Heroku nos permitiría escalar cuando sea necesario, incluso se podría colocar en automático.

2 Diseño

Para instalar, revisar el README.md
El URL de la documentación de Redoc está en `https://pedidosya.rogs.me/redoc/`
El URL de Swagger está en `https://pedidosya.rogs.me/swagger`
El URL de admin está en `https://pedidosya.rogs.me/accounts/login/`
El usuario administrador (para eliminar comentarios) es:

- user: admin
- password: 12345

Para probar la app pueden usar `https://pedidosya.rogs.me/`
Para buscar por store_{id}: `https://pedidosya.rogs.me/opinions/?store_id=1`
Para buscar por user_{id}: `https://pedidosya.rogs.me/opinions/?user_id=1`
Para buscar por $\text{purchase}_{\text{id}}$: `https://pedidosya.rogs.me/opinions/?purchase_id=1`
Para buscar entre fechas: `https://pedidosya.rogs.me/opinions/?from_date=date&to_date=date`

3 Cosas que faltaron

- Pruebas unitarias. No tuve el tiempo necesario