Services

Jump to bottom

Jóakim von Kistowski edited this page on 29 Mar 2018 · 28 revisions

The **TeaStore** consists of 5 replicatable services and a single **Registry** instance. The **WebUI** service retrieves images from the **ImageProvider**. Users are authenticated by the **Auth** service. Data is retrieved from the **PersistenceProvider** and product recommendations from the **Recommender** service.

WebUI

The WebUI provides the TeaStore front-end using Servlets in combination with JSP files. It contains logic to save and retireve values from cookies. Images (with few exceptions) are not provides by the WebUi, but are retrieved from the Image Provider Service instead.

The UI provides a status page at `http://$host$:$port$/tools.descartes.teastore.webui/status` indicating the current state of the TeaStore. The status view lists the instance count and hosts for all registered service instances.

Image Provider

The Image Provider delivers images to the WebUI as base64 encoded strings to embed them in the final HTML. It matches the provided product ID or UI name (the filename for images not representing a product and therefore without product ID) and the image size to a unique image identifier.

- If the product ID or UI name is not available to the Image Provider, a standard "*not found*" image will be delivered in the correct size.
- If the product ID or UI name is found but not in the requested size, the largest image will be loaded and scaled. The scaled image is stored for later use.
- If the product ID or UI name and size is found, the image will be loaded and delivered.

To speed up image delivery, an in-memory cache with Least Frequently Used (LFU) replacemenent strategy is in place. Before loading an image from the physical drive, a cache lookup will be performed and if the image is present, delivered directly from cache or from the physical drive otherwise.

Auth

The Auth service handles user and session authentication. Passwords are hashed using BCrypt. To validate sessions, the SessionBlob is salted and hashed using SHA512 and stored in a cookie. When a SessionBlob is received, the cookie content is re-salted and checked against the hash to check for session tampering.

Persistence Provider

The Persistence service provides access to the data persisted in the relational database back-end. It maps the relational entities to the JSON entity objects passed between services using the EclipseLink JPA ORM mapper. It features endpoints for general CRUD-Operations (Create, Read, Update, Delete) for the persistent entities. The persistence provider uses a second level entity cache provided by the JPA implementation. As such, it also acts as a caching layer.

Recommender

The Recommender is used to generate individual product recommendations for each user. It is trained using all existing orders. Recommendations are generated based on the users current shopping cart, the user's previous orders and/or the item the user is currently looking at. The item rating of the users are based on their purchases.

If the current user is unknown (e.g., if the user is not logged in or did not purchase anything yet), a fallback algorithm based on the general item popularity is chosen. If the user is known, Slope One as item-based collaborative filtering is applied to calculate the recommendations. We implemented two versions of the algorithm: One CPU-intensive, calculating the item-rankings per user on-the-go and one memory-intensive, calculating the total user rating prediction matrix during the training phase. Furthermore, one order-based nearest-neighbor approach is available. Its recommendation time is dependent on the number of items in the current cart as well as the total number of stored orders.

Registry

The Registry provides information about how many service instances are online for each service and where they are located. Service instance register themselves at the registry on startup. Services are also required to send a heartbeat signal by re-registering periodically. Services missing their heartbeat for more than 10 s are assumed to be offline until the register function is called again.

Every running instance of the TeaStore uses one single registry. The TeaStore is a test application. By limiting it to a single registry instance, it enables easy configuration of multiple parallel TeaStores with minimal configuration overhead.

▸ Pages 6

TeaStore Home

1. **Getting Started**
   i. Run using Docker or k8s
   ii. Deploy on Java Application Server
2. **Testing and Benchmarking**
   i. Load Generation
   ii. Instrumentation
3. **Customizing the TeaStore**
4. **Architecture and Services**
5. **Troubleshooting**
6. **Cite Us**

Clone this wiki locally

```
https://github.com/DescartesResearch/TeaStore.wiki.git
```