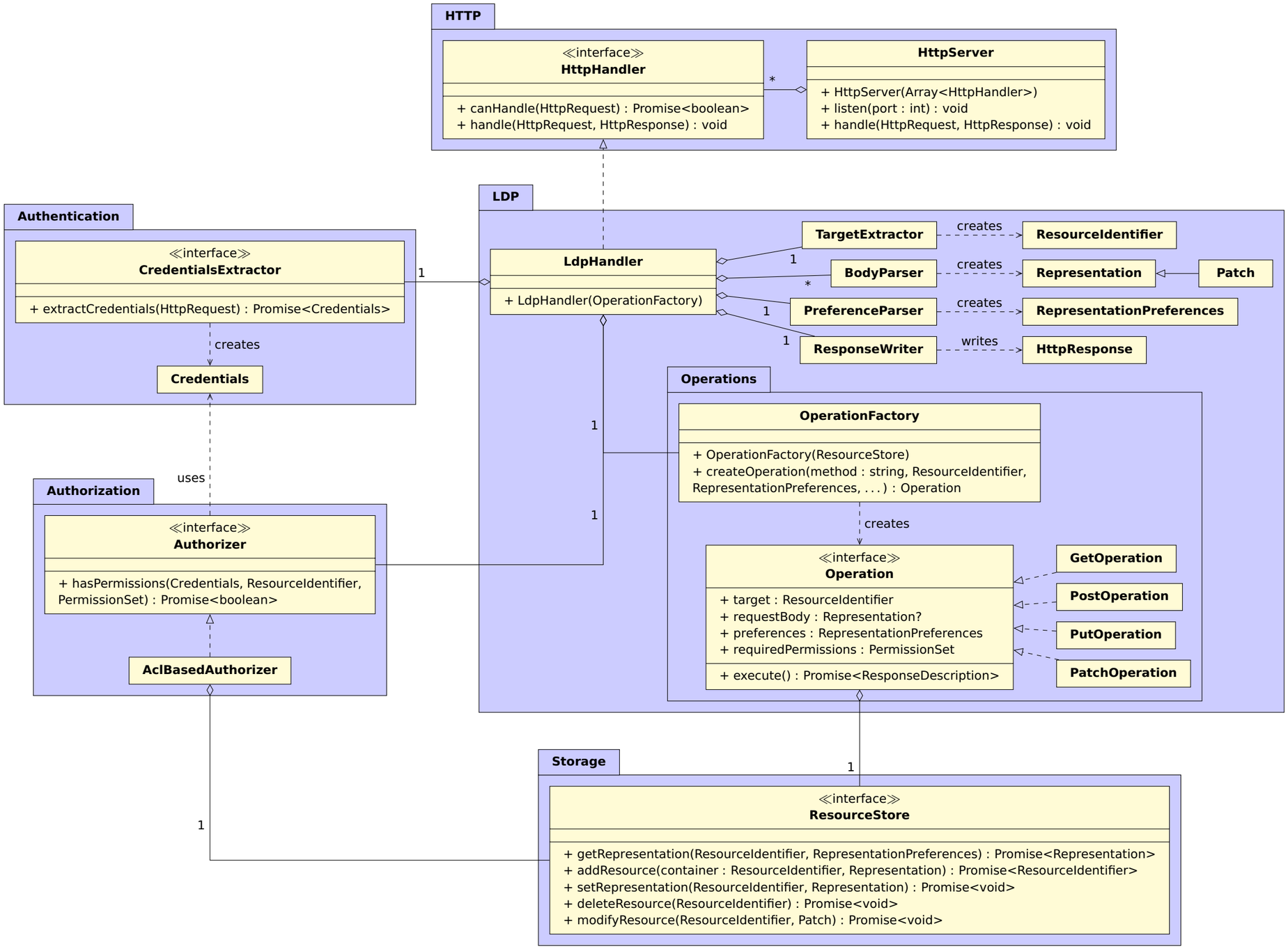
Solid server – Proposed architecture v1.1.0 *(status: draft)*

Ruben Verborgh – July 2, 2019

Purpose

This document conveys a personal view on important architectural considerations for a Solid server.
It is intended as a tool for discussion, to raise questions, and to highlight concerns.
It does not have any official standing whatsoever.

Overview of LDP and Access Control

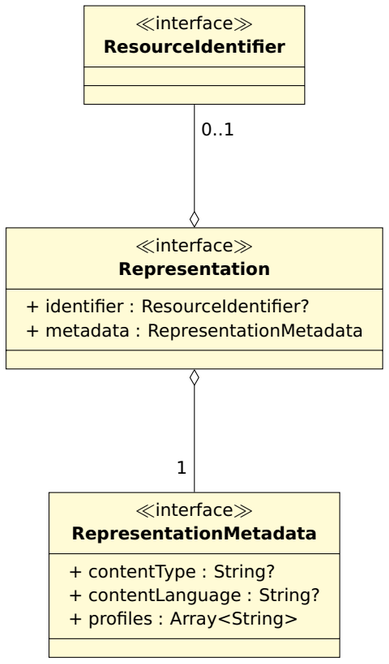


Resources and Representations

The intention of **ResourceIdentifier** and **Representation** is to capture the REST notion of a resource and its representation. In the case of a photograph, the resource is the photograph itself, whereas a representation is a concrete manifestation of that photograph with a certain resolution and file type. In the case of an RDF document, the resource is the RDF graph, and concrete representations serialize that graph into Turtle or specific framings of JSON-LD.

For all practical purposes, **ResourceIdentifier** can just be a **URL**; the terminology is mainly used to emphasize the resource/representation notion of REST. Also, there is no **Resource** class, because resources are always manipulated through representations in REST, so we only need to *identify* resources, and only deal with them through their representations.

Crucially, as the diagram below shows, the **Representation** interface can have vastly different underlying in-memory structures, such as strings, binary streams, RDF streams, etc. So they can be photographs as well as RDF streams, and most other classes handling them do not need to care. This enables backends to be RDF-aware when they need to, and RDF-oblivious when they do not.



Example classes and interfaces deriving from ResourceStore