Example Project: MediaStore

The MediaStore is a web-based store for purchasing audio and video files, whose functionality is modeled after Apple's iTunes music store. It is a three-tier architecture assembled from a number of independently usable software components (see Figure 1). It consisted of a user interface (here, the `WebGUI` components responsibility), business logic (encapsulated in the `MediaStore` and `Digital-Watermarking` components) and a database (the `AudioDB` component).

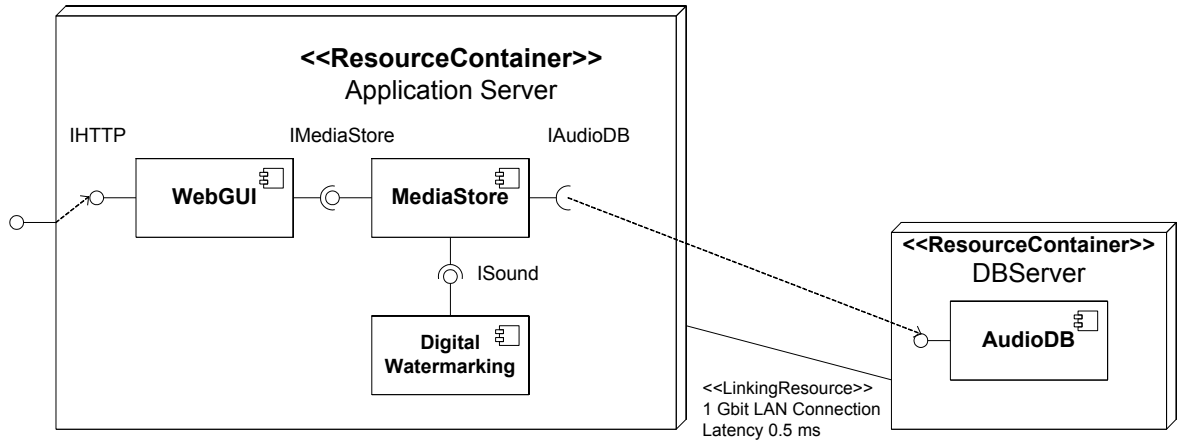

Figure 1: MediaStore Architecture

Users interact with the store via web browsers, and may purchase and download different kinds of media files, which are stored in a database connected to the store's application server. Figure 2 shows a dynamic view of the `MediaStore` architecture where a user downloads a set of files from the store. Via the web browser, the user provides the `WebGUI` component with a query that results in a set of media files from the database. After forwarding the query to the `MediaStore` component, the store searches the `AudioDB` component for the requested files. The matching files are then transferred back to the `MediaStore` component. For copy protection, the `MediaStore` adds a digital watermark to each file. With the watermark, which is unrecognisable by the user, the store can for example add the current user's ID to the files. If the respecting files would appear elsewhere on the Internet, for example in file sharing services, the user who downloaded the files from the store, could be tracked down with the water mark. The component `DigitalWatermarking` carries out the actual watermarking of the media files. It can be configured to include additional texts, such as lyrics or subtitles, into the files as digital watermarks. This component processes single files, therefore the `MediaStore` component calls it for each file that was requested by the user in a loop. After completing the watermarking, the system sends all files to the user.

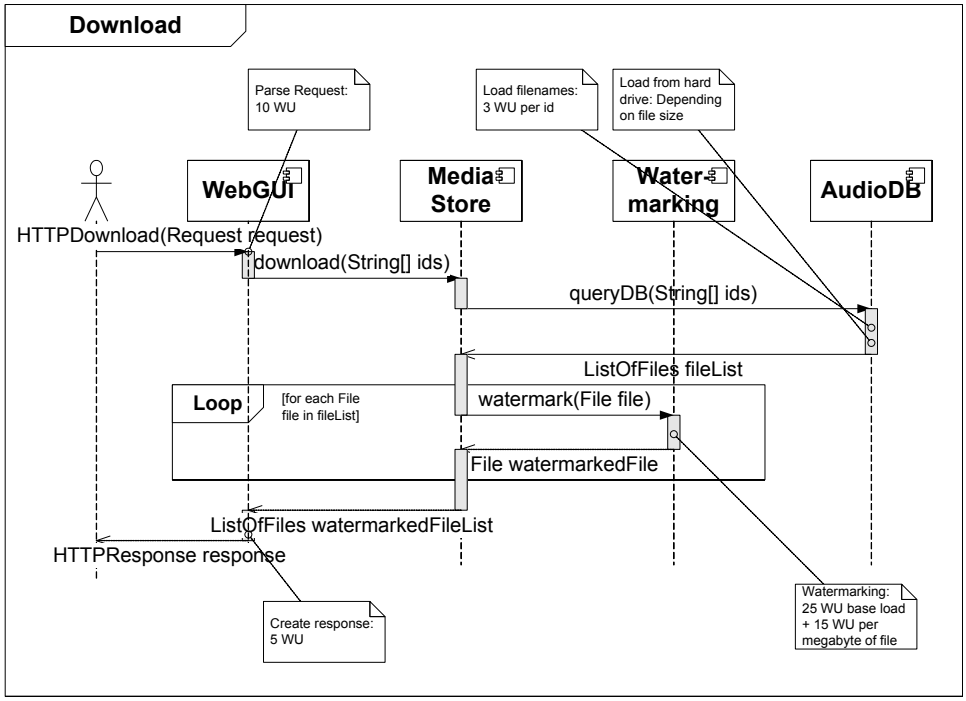

Figure 2: Sequence Diagram Download

Starting from this base model, it is possible to evaluate different design alternatives:

- **Introduction of a cache component**:
  For this option, a new `Cache` component was introduced and chained between the `MediaStore` and the `AudioDB` component. The cache kept a certain amount of mp3 files in memory and thus reduced the number of hard disk drive accesses. However, the check of the availability of a file in the cache needed some calculation done by the CPU. The cache hit ratio was given in the task description.
- **Use of a database connection pool:**
  For this option, the `AudioDB` component was replaced with the `PoolingAudioDB` component, that used a database connection pool to access the internal database. Thus, the accesses to the database needed less computations and the number of concurrent accesses to the hard disk drive was reduced, thereby reducing contention effects. However, only a certain amount of transactions could be executed concurrently, other users had to wait.
- **Re-encoding and reduction of bit rate:**
  For this option, a new `Encoding` component was introduced and the `MediaStore` component was replaced by a `Encoding-MediaStore` component calling the `Encoding` component before sending files to the `AudioDB`. The `Encoding` component reduced the bit rate of mp3 files with high bit rates (e.g. CD quality) by re-encoding them and thus reduced the file size of files in the database. The drawback was a computational effort for the encoding.