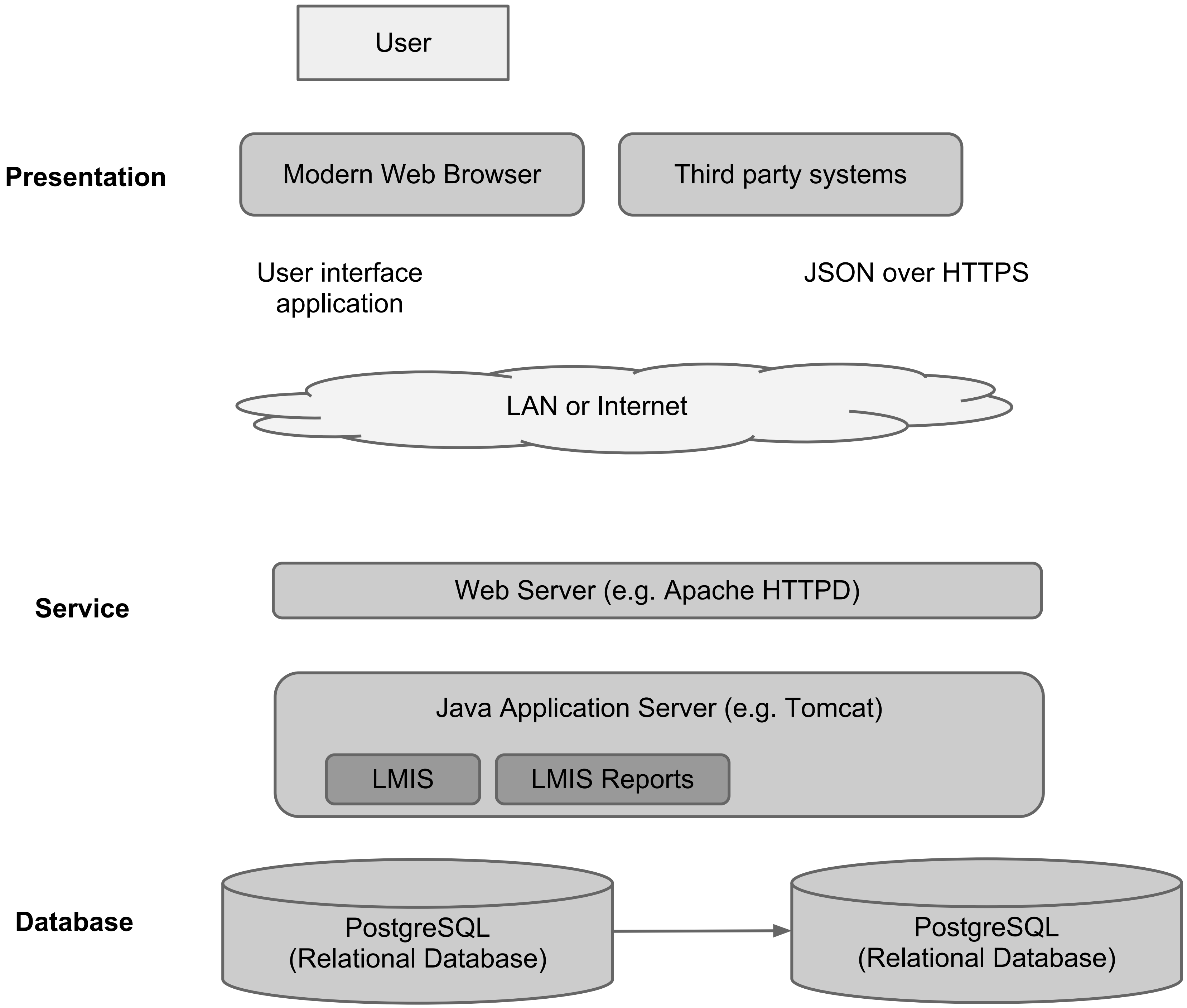
OpenLMIS Architecture


User
Presentation
Modern Web Browser
Third party systems
User interface application
JSON over HTTPS
LAN or Internet
Service
Web Server (e.g. Apache HTTPD)
Java Application Server (e.g. Tomcat)
LMIS
LMIS Reports
Database
PostgreSQL (Relational Database)
PostgreSQL (Relational Database)

Presentation layer (Web Browser)

Technology

HTML5, JavaScript, CSS

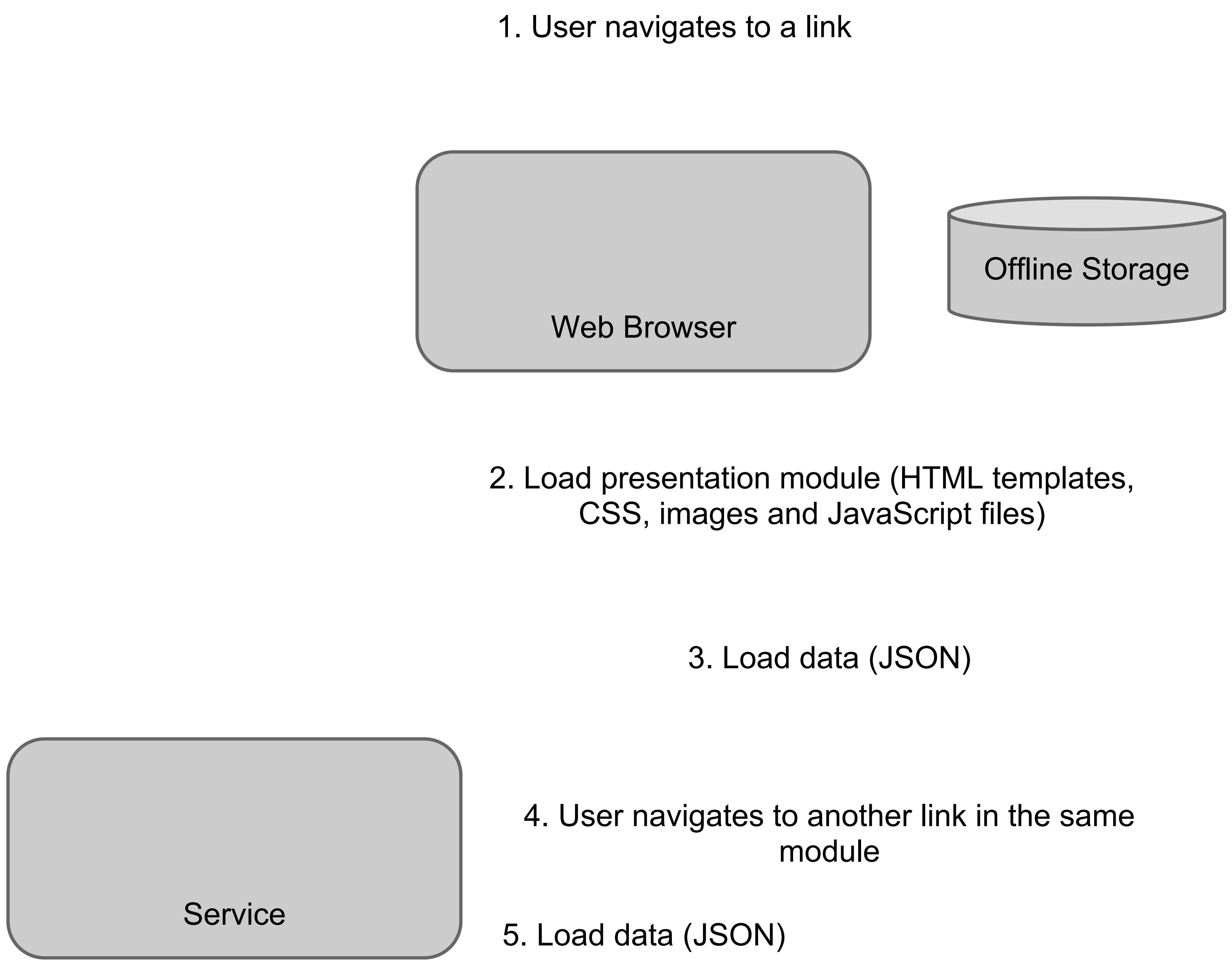
Tools/Libraries

AngularJS, jQuery & Bootstrap

LMIS modules

Facility, RnR, RnR-Template, Role, Schedule, Upload, Shared

- each module has all infrastructure layers - controller, model, routes


1. User navigates to a link
Web Browser
Offline Storage
2. Load presentation module (HTML templates, CSS, images and JavaScript files)
3. Load data (JSON)
4. User navigates to another link in the same module
Service
5. Load data (JSON)

Presentation layer - Architecture considerations

HTML rendering - in browser over server side

Use the some system functionality in offline mode

Local storage instead of cookies

Reduced upload traffic to server

Service layer (LMIS)

Technology

Java, J2EE

Tools/Libraries

Spring framework, Spring MVC, MyBatis

LMIS modules

Authentication, Core, Requisition, Upload & Web

- Core has all domain abstractions for setup/reference data like Product, Program, Facility etc.

- each module has all infrastructure layers - service, domain, repository, etc.

Service layer (LMIS) - Architecture considerations

Spring
Most widely used (almost a default choice)
Reduces boilerplate code
Unit testing driven development friendly

MyBatis over Hibernate *(Object relational mapping tools)*
Explicit SQL easier to review for performance issues
Easier to learn

JSON over XML
Easier to map serialize to/from objects
Relatively easier for non-developers to understand

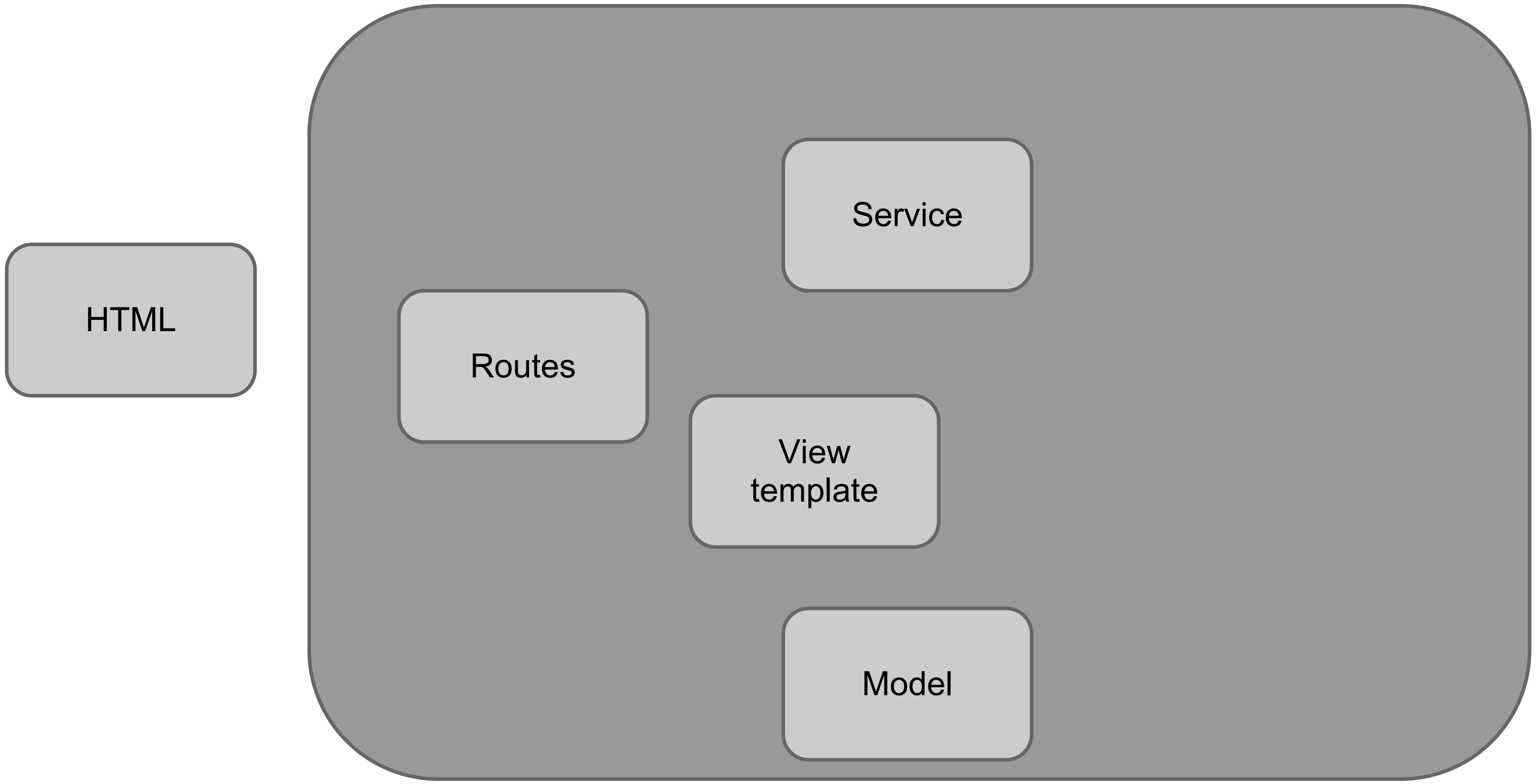
Reporting application

Platform
Jasper Reports (Java, J2EE)
Presentation in web browser

Out of box capabilities
User management
Excel, PDF, CSV, HTML and XML-over-WebService

Development effort is mostly in defining SQL queries

1. Navigate to a link



Database

PostgreSQL over MySQL

Suitable for large and agile databases

Custom data types like tree-node (ltree), documents, etc.

No stored procedures & triggers (unless a must)

Languages supported by database not as powerful

Debugging, refactoring, testing is difficult

Database becomes bottleneck at large scale

Database server setup

Continuous replication over batch job

Minimal loss of data active node failure

Reports generated from passive near real-time

Reporting from passive server

Non-performant queries not affecting transactional operations

Test for the correctness of backup process