MarkLogic Server

Reference Application Architecture Guide

MarkLogic 9
May, 2017

Last Revised: 9.0-1, May, 2017

Table of Contents

Reference Application Architecture Guide

1.0 Understanding the Reference Architecture

The MarkLogic Reference Application Architecture is a three-tier application template and set of best practices for architects, developers, and adminstrators designing, developing, and deploying applications that use MarkLogic Server.

This guide covers the following topics:

1.1 Purpose

MarkLogic is a flexible and powerful platform, capable of fitting into many different application solutions. The MarkLogic Reference Application Architecture provides an three-tier application template and set of best practices for architects, developers, and adminstrators designing, developing, and deploying applications that use MarkLogic Server.

The MarkLogic Reference Application Architecture consists of the following parts:

- A description of the application architecture, the responsibilities of each tier, and the relationship between the tiers.
- A set of recommended best practices for developing reliable, large-scale application on top of MarkLogic Server.
- An open source example application, `samplestack`, that implements the recommended three-tier model on modern tools and frameworks, using the best practices.

1.2 Why a Three Tier Architecture

Historically, much of what end users see of a web application is rendered by a back-end server that manages application and session state, and constructs HTML for the browser. Today, browsers and browser frameworks are more capable, so the balance has shifted towards browser applications managing their own views, session state, and some of the business logic. It is also now common for applications to expose business services through REST-style APIs that are easy to integrate with, making it easier to swap out parts of the application.

There are many ways to structure a MarkLogic application, including the traditional 2-tier model. For example, MarkLogic Server includes an application server, so you can easily create two-tier MarkLogic applications that generate HTML for the browser from within MarkLogic Server using XQuery or JavaScript.

However, we've chosen a three-tier model that leverages JSON, REST over HTTP, and Java/JavaScript as the reference architecture because we believe it enables a team new to MarkLogic to become productive as quickly and easily as possible. The MarkLogic Reference Application Architecture offers the following advantages:

- The team can work exclusively with industry standard frameworks and programming languages (Java, Spring, JavaScript, AngularJS, JSON).
- Little knowledge of MarkLogic internals is required to get started.
- Separation of business logic from the data services layer makes it easier to integrate MarkLogic with existing enterprise infrastructure.

A three-tier architecture offers additional benefits such as the ability to scale and optimize the tiers independently and separation of security concerns.

1.3 Reference Architecture Overview

The MarkLogic Reference Application Architecture is a three-tier model containing database, middle, and browser tiers. As shown in the following diagram, JSON over RESTful HTTP is the transport mechanism between all tiers.

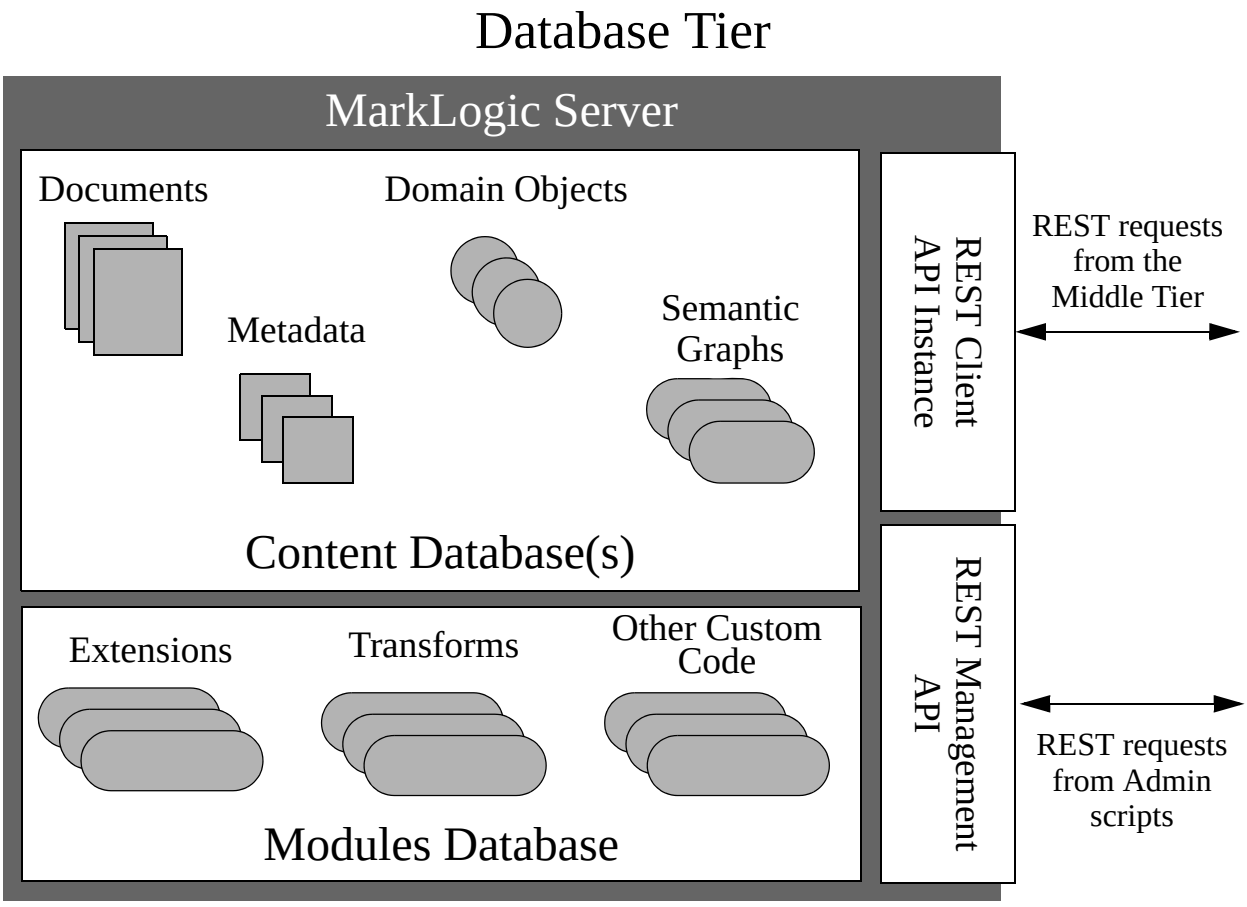
The database tier provides high availability, long-running data services to the middle tier. All the data required by the application is managed by MarkLogic Server in the database tier. Persistent application state is managed here for purposes of scalability and simplicity. This tier can include code that needs to run close to the data for performance reason or to enforce the data model. MarkLogic provides services and data to the middle tier through one of the powerful and extensible MarkLogic client APIs. For details, see "Database Tier" on page 5.

The middle tier provides data to and shares session state with the browser tier. In a MarkLogic application, it brokers exchanges between the browser and database tiers using one of the MarkLogic client APIs. The middle tier implements business logic verification and can have additional responsibilities such as rate limiting and integrations with non-MarkLogic external services. For details, see "Middle Tier" on page 7.

The browser tier contains the web application front-end that faces end users. The application includes code that runs in the browser, markup, and styles that tailor the user experience. In the MarkLogic Reference Application Architecture, the browser application is a rich client. That is, the browser tier fully owns the rendering of the UI, including decisions about how views are organized and most of transitions within a flow. The browser tier shares awareness of business logic with the middle tier. For details, see "Browser Tier" on page 8.

1.4 Database Tier

The database tier provides data and application services to the middle tier. The following diagram outlines the major components of the database tier.



MarkLogic Server manages data and code required by the application, such as the following:

- JSON, XML, Binary and Text documents
- Metadata about documents, such as permissions, quality, collections, and application-specific document properties
- Persistent application state, such as domain objects realized in the middle tier as Java or JavaScript objects
- Semantic graphs
- Schemas
- Extensions, transformations, and other application-specific code that needs runs in the database tier for purposes of performance, encapsulation, or enforcement of data rules

Storing such assets in the database provides transactional integrity to the application. The MarkLogic transaction model includes multi-statement transactions that enable applications to interleave transactional operations in the database tier with business logic in the middle and browser tiers.

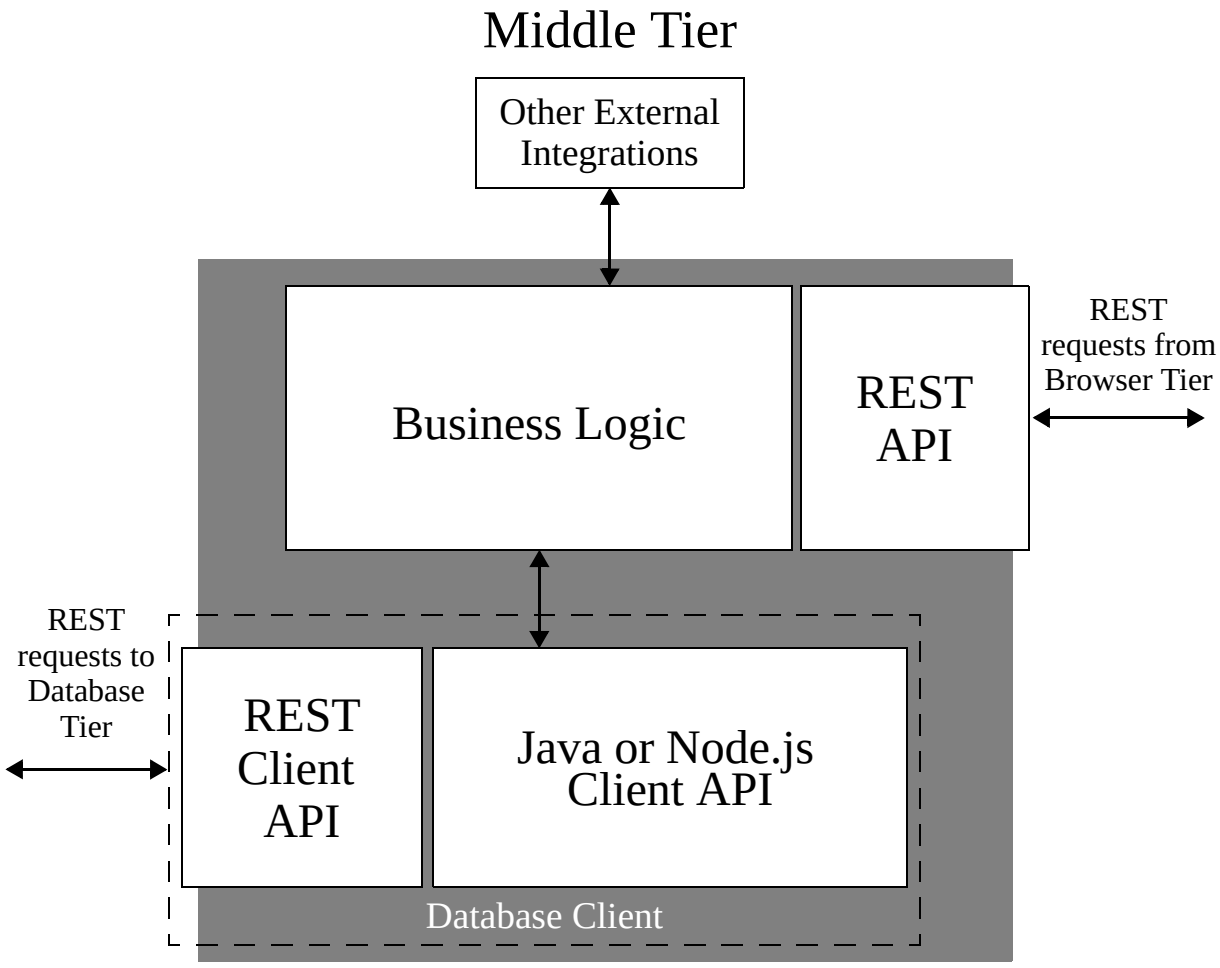
The middle tier communicates with MarkLogic Server through the REST Client API and the REST Management API. Incoming REST requests are handled by an HTTP App Server embedded in MarkLogic Server.

The REST Client API is the foundation of a family of MarkLogic client APIs that enable applications to create, read, update, delete, and search database content. The Java Client API and the Node.js Client API use this foundation. All the MarkLogic client APIs are extensible, so you can easily install and use content transformations, search customizations, REST resource services extensions, and other database tier library modules. All such database tier code is stored in a modules database and can be managed through the MarkLogic client APIs.

The REST Management API enables developers and administrators to manage, monitor, and review their MarkLogic Server configuration and status remotely through REST requests. Though MarkLogic Server has user interfaces for interactive administration and monitoring, the REST Management API makes it easy to script such tasks.

1.5 Middle Tier

The middle tier brokers inter-server communications between the browser tier and the database tier and other external services. The following diagram shows the major components of the middle tier of a MarkLogic application:



Even in a thick client model where the majority of the business logic may be in the browser tier and data flows between the browser tier and the database tier with little or no modification, the middle tier provides critical services such as the following:

- Coordination of inter-server communications for the browser tier, such as between the browser tier and the database tier or the browser tier and other external services.
- Security services. The middle tier can provide user authentication services and verify that data coming from the insecure browser tier adheres to the business rules.
- Network traffic optimization for the browser tier. Communication between the browser and the middle and database tier is typically a relatively long hop, so minimizing requests from the browser improves performance. The browser tier can make a single request to the middle tier that requires multiple requests between the middle tier and the database tier.

- Transactional integrity. The middle tier can coordinate application logically discrete operations that should be part of a single transaction, such as updating an account balance at the same time the user authorizes a withdrawal or deposit.

A rich browser client is usually the first line implementer of business logic, but the middle tier injects business logic as necessary, to support the services listed above.

The middle tier communicates with MarkLogic Server through a database client using one of the MarkLogic client APIs, such as the Java Client API or the Node.js Client API. These APIs access MarkLogic through REST requests over HTTP, while providing a fluent interface natural to your application. Using these APIs means developers do not need to learn a new programming or query language and injection risks via unsafe evaluation of strings of code can be eliminated.

The MarkLogic client APIs enable you to do the following:

- Insert, read, update, and delete documents, metadata, semantic triples, and domain objects, singly or in batches.
- Query documents and other data in the database using MarkLogic's powerful search features. Choose from among several query styles, depending on what best fits your needs.
- Extend the built-in services through several extension points. You can use and manage your extensions through the MarkLogic client APIs.

The data structures that are returned by MarkLogic through the client APIs can be passed through directly to the browser tier as JSON or XML. Similarly, domain objects stored in the database can be retrieved as native Java and JavaScript objects.

The business logic in the middle tier validates data delivered by the browser tier, as well as handling integration with other systems. The middle tier may publish a REST interface with which to communicate with the browser tier in a way natural to the application. This API should be flexible enough to support swapping in a different front-end implementation. Typically, data flows between the middle tier and the browser tier as JSON.

1.6 Browser Tier

MarkLogic does not require any particular web application architecture. The reference architecture promotes a thick client such as a Single Page Application (SPA), where the browser tier is responsible for all view rendering and transient application state. However, other browser tier solutions can be plugged into the architecture.

The browser tier uses JavaScript to interact with the middle tier through JSON services provided by an application-specific REST API. The services provided by this REST API model the tier interactions in a way that is natural to the application. The existence of this API does not necessarily require data transformations.

In the thick client, SPA solution, the browser is an Model-View-Controller (MVC) application that includes business logic and shares a high degree of fidelity with the data model exposed by the database tier. This enables the browser tier to consume data from the database tier (by way of the middle tier) with minimal transformation.

Though the browser tier implements much of the business logic, the browser is inherently more vulnerable to risks like injection and denial of service attacks. The middle and database tiers provide protection in the form of security, verification of the business and data rules, rate limiting.

The model can perform JavaScript object validation against a JSON schema. The view is HTML and CSS based.

1.7 Next Steps

Refer to the following table for suggestions of further reading and activities to continue learning about MarkLogic Server application development.

If you want to	Then see
Learn more about the Reference Architecture	"Samplestack: A Reference Architecture Instantiation" on page 12
Explore a full-feature MarkLogic Application based on the Reference Architecture	The `samplestack` application at http://github.com/marklogic/marklogic-samplestack
Learn about specific MarkLogic capabilities such as document operations, search, or semantics.	Tutorials on the MarkLogic developer community site, http://developer.marklogic.com/learn
Learn more about the MarkLogic client APIs	*Java Application Developer's Guide* *Node.js Application Developer's Guide* *REST Application Developer's Guide*
Learn about MarkLogic Server internals	The "Inside MarkLogic Server" paper on the MarkLogic developer community site, http://developer.marklogic.com/inside-marklogic

2.0 Recommended Best Practices

When designing your MarkLogic application, you should consider incorporate at least the following best practices into your development process:

- Organize your project around logical components
- Use source control for code, tests, and automation drivers
- Make project development and deployment easy to configure
- Automate development, testing, and deployment tasks
- Incorporate automated testing into all phases of development

2.1 Organize your project around logical components

You should organize your project in a way that preserves the logical separation of concerns in your application. This makes it easier to share the project among teams working in the different tiers and makes it easier to find source files and other assets.

2.2 Use source control for code, tests, and automation drivers

Your source code, tests, automation scripts, and other application assets should all be under source control. Source control preserves historical changes to your application and enables multiple people to work on a project concurrently. Putting all your application development and deployment components under source control also makes it easy to know goes into each release of your product. Even a the smallest team benefits from source control.

2.3 Make project development and deployment easy to configure

The building, testing, and deployment of your application should be configuration driven.

For example, if you move your testing infrastructure to a new host, you should only need to update a single configuration file, not every test. Similarly, you should be able to change product build dependencies with minimal configuration file changes.

Consolidating such variables into configuration files makes it easier to track and less prone to error when change is required.

2.4 Automate development, testing, and deployment tasks

Project development involves many repetitive tasks. For example, a developer goes through the edit-test-debug cycle many times in a single day. Automating common tasks improves productivity and reproducibility.

Similarly, using automation to simplify testing removes barriers to frequent developer testing. Frequent testing improves product stability because problems are easiest to diagnose and fix close to the point at which they’re introduced. Test automation also helps improve the reliability of test results.

Automating the setup of your application development and deployment environment makes it easier to add new developers, manage dependencies, and provide a stable development, testing, and release platform.

2.5 Incorporate automated testing into all phases of development

It is important to have several levels of tests for your product. For example:

- Unit tests verify the functionality of specific sections of code, usually at the function/class/interface level. Unit tests are usually created by the developers. Unit tests can easily be run during a developer's edit-test-debug cycle and before each checkin.
- Integration tests verify the interfaces and integration points between components, such as between the browser tier and the middle tier.
- Regression tests detect when previously working features break as an unintended consequence of a code change.
- System tests verify the end-to-end behavior of your application.
- Smoke tests, sometimes called sanity tests, are a small test set that check for some minimal level of operability. For example, you can use smoke tests to test a project integration branch before merging changes into a code line shared with a larger set of developers.

3.0 Samplestack: A Reference Architecture Instantiation

This chapter provides a high level overview of the `Samplestack` application. `Samplestack` is demo application based the MarkLogic Reference Application Architecture.

The following topics are covered:

3.1 What is Samplestack?

Samplestack is a full-featured example web application that implements the MarkLogic Reference Application Architecture and the best practices described in this guide. You can deploy Samplestack in your development environment as a learning tool; it is maintained as an open source project.

The technologies selected for use in Samplestack reflect a forward-looking approach to developing enterprise web applications. However, nothing about the architecture, the application, or the best practices requires you to use these specific technologies.

Samplestack models a Question and Answer site using freely available data from stackoverflow.com. A Samplestack user can perform the following tasks:

- Browse or search questions and answers
- Submit a question
- Submit an answer
- Vote on a question or an answer
- Accept an answer (submitter only)

Users can browse and search questions and answers anonymously. Submitting questions, answers, or comments; accepting answers; and voting require the user to log into the site. Voting affects the ordering of search results because questions with more votes appear in the search results before questions with fewer votes. Accepting a “best” answer to a question affects the reputation of the contributor of the accepted answer.

MarkLogic Server provides the database tier of Samplestack. The middle tier is a Java stack, based on Spring Boot and other standard Java libraries. The browser tier is a Single Page Application (SPA) implemented on AngularJS. For details on the libraries and frameworks used by Samplestack, see “Technologies Used in Samplestack” on page 23.

The Samplestack project includes robust project infrastructure that demonstrates the best practices outlined in “Recommended Best Practices” on page 10. For details, see “Best Practices Demonstrated by Samplestack” on page 21.

Samplestack is an open source project maintained on GitHub. You can review, download, and contribute to the project at http://github.com/marklogic/marklogic-samplestack.

3.2 Samplestack Implementation Overview

This section discusses how some key features of Samplestack are enabled by MarkLogic Server. Samplestack uses additional MarkLogic features not discussed here. For more details, see “Exploring Samplestack in Detail” on page 25.

The following topics are covered:

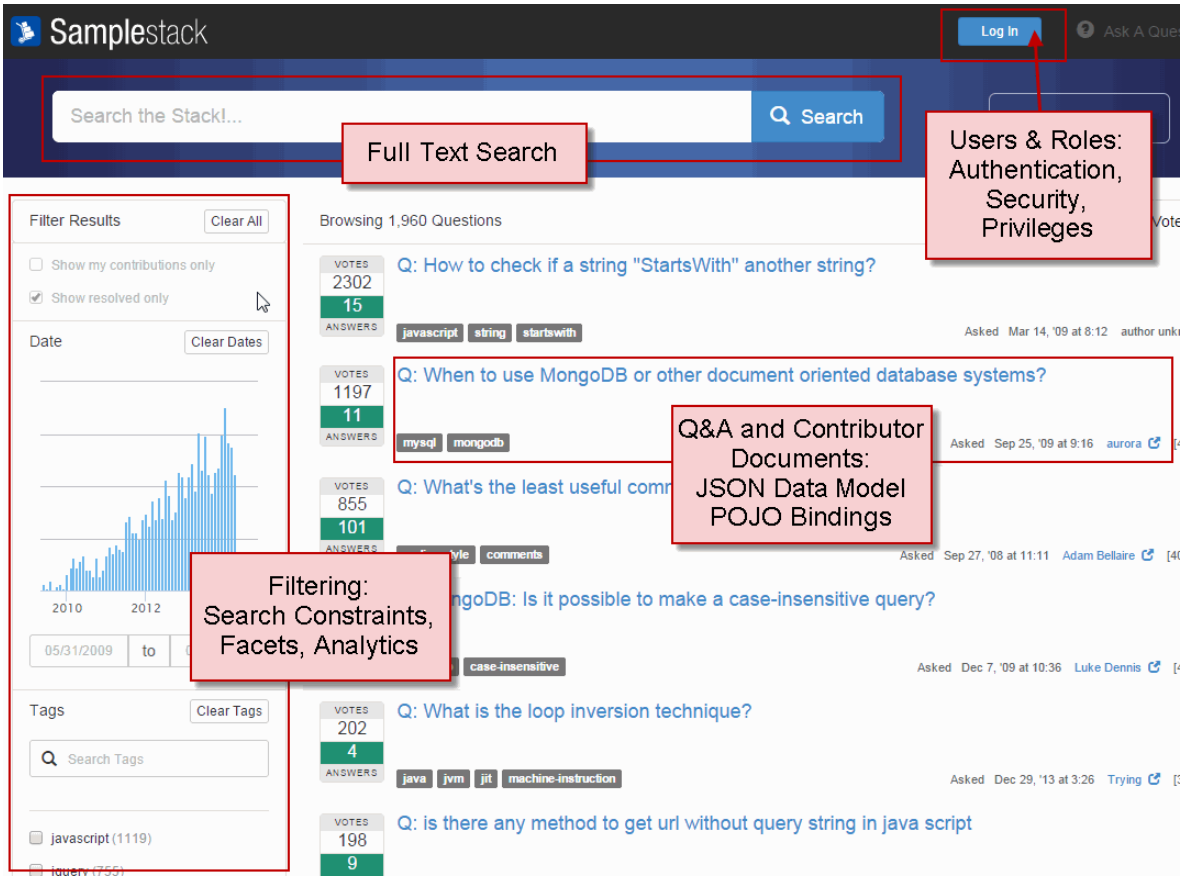
3.2.1 UI to MarkLogic Feature Summary

The following table maps visual elements of Samplestack to MarkLogic capabilities and points to a brief related discussion. The graphics after this table tie the same visually tie the same mapping to the Samplestack UI.

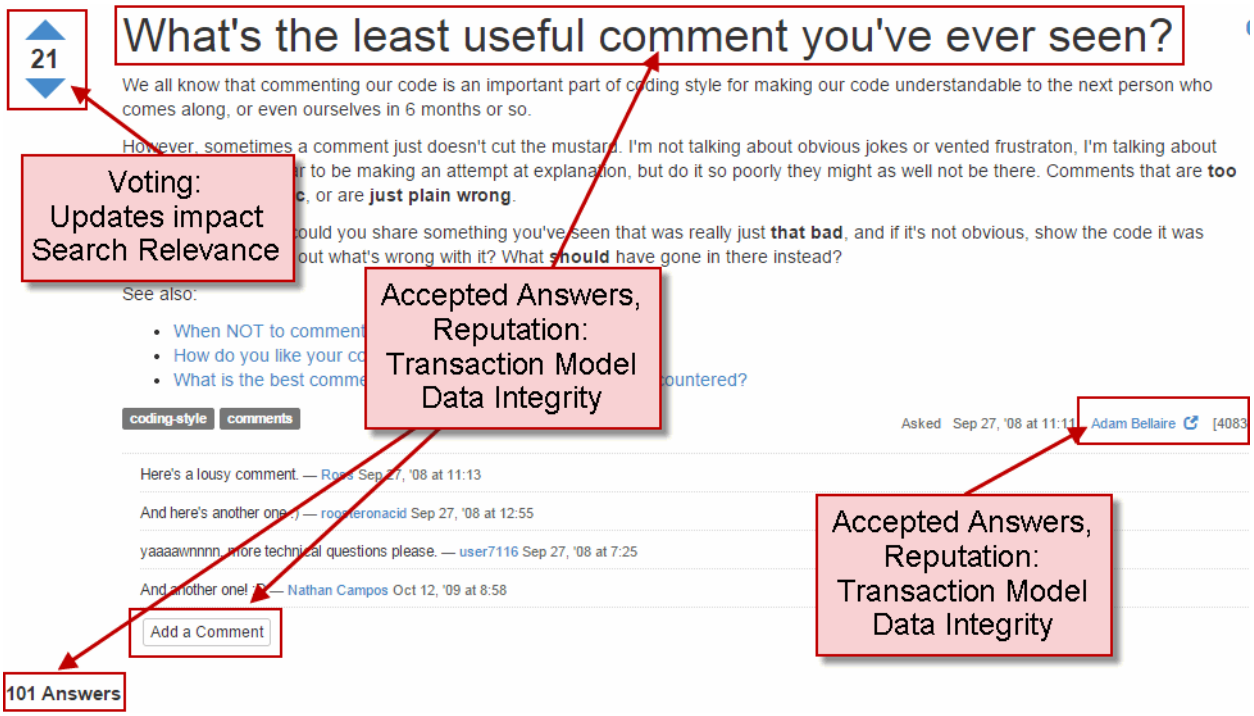
Application Feature	MarkLogic Capability	Where to go for more information
Full-text Search	Full-text search, different query styles, indexing	“Full Text Search” on page 15
Facets	Search constraints, analytics	“Search Result Filtering” on page 16

Application Feature	MarkLogic Capability	Where to go for more information
Users and Roles	Authentication, security, privileges	“Users and Roles” on page 16
User Records and Q&A Documents	JSON data model, POJO data binding	“Document Model” on page 18
Submitting Questions, Answers, and Comments	Inserting and updating documents	“Document Insertion and Update” on page 19
Accepting Answers, Reputation	Transaction model, data integrity	“Transactions and Data Integrity” on page 20

The following image highlights MarkLogic capabilities that enable key elements of the Samplestack search view. From this view, users can search (with or without filters), review search results, log in, and ask questions (if logged in).



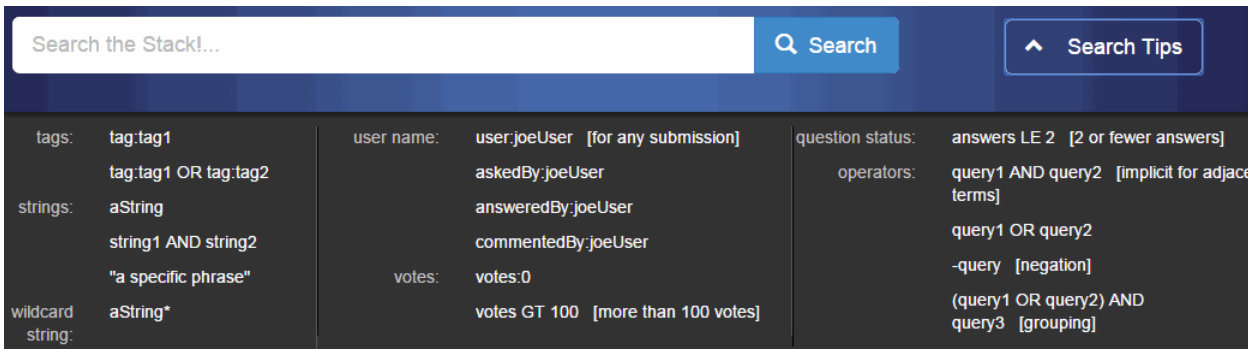
The following image highlights MarkLogic capabilities that enable key elements of a Samplestack Q&A view. From this view, users can review questions, answers, and comments. Logged in users can post answers, post comments, and vote on questions and answers. The submitter of a question can also accept a best answer.



3.2.2 Full Text Search

The full-text search box at the top of each page uses MarkLogic's built-in string search grammar, augmented by the definition of some custom bindings between a search term qualifier and specific data properties such as username or tags.

If you click on Search Tips, you see a summary of the search grammar supported by Samplestack:

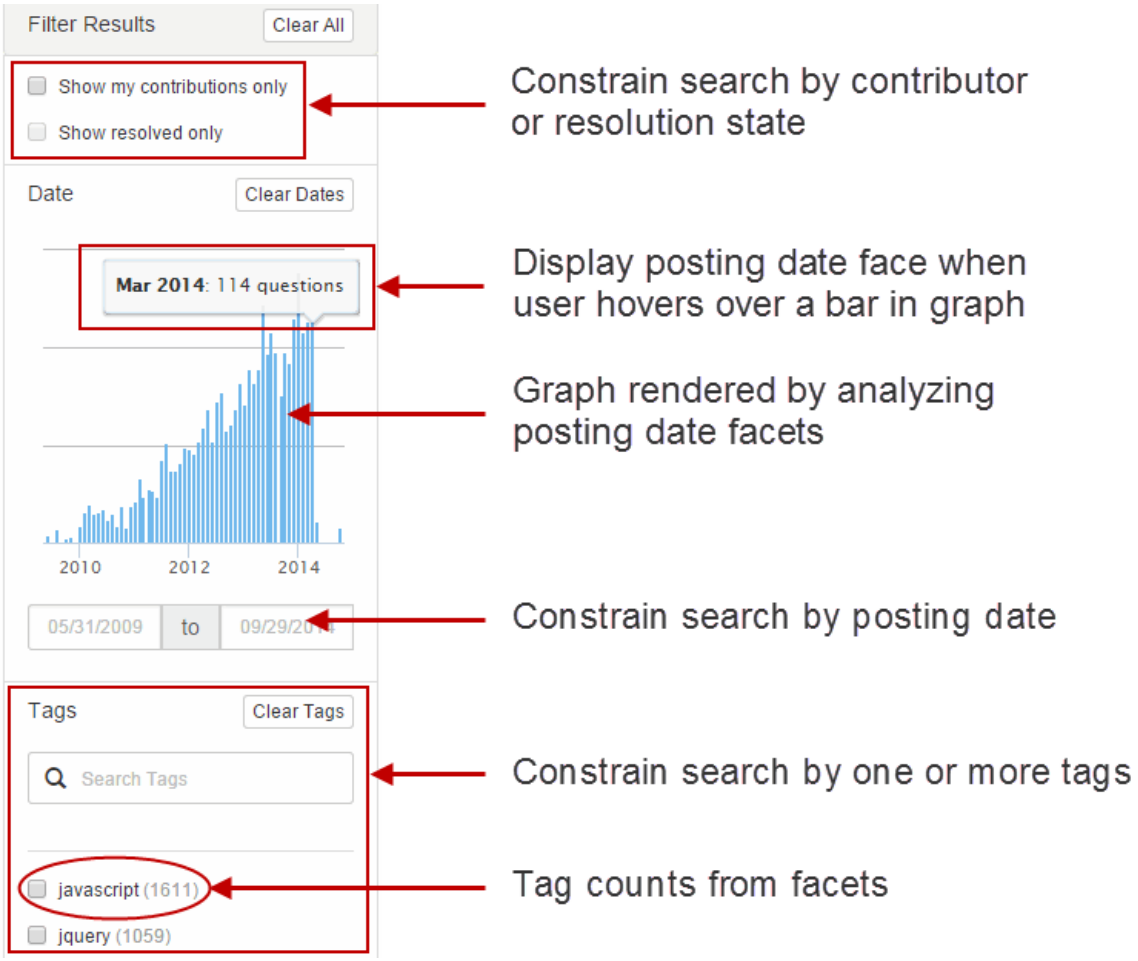


All the expressiveness in this summary is built into the MarkLogic default string search grammar, including logical operators, relational operators, grouping, and qualified terms ("tag:value").

The application-specific prefixes on qualified terms (`tag`, `user`, `askedBy`, `answeredBy`, `commentedBy`, `votes`) represent a binding between a name ("answeredBy") and a slice of the database content backed by a range index. The MarkLogic client APIs include simple hooks for defining such bindings.

3.2.3 Search Result Filtering

The "Filter Results" widget on the left side of the page enables users to narrow a search in several ways. The following picture highlights the types of filtering. The search constraint and faceting features that back these filters are built into MarkLogic. The application simply configures the details, such as which content properties to constrain by or generate facets from.



3.2.4 Users and Roles

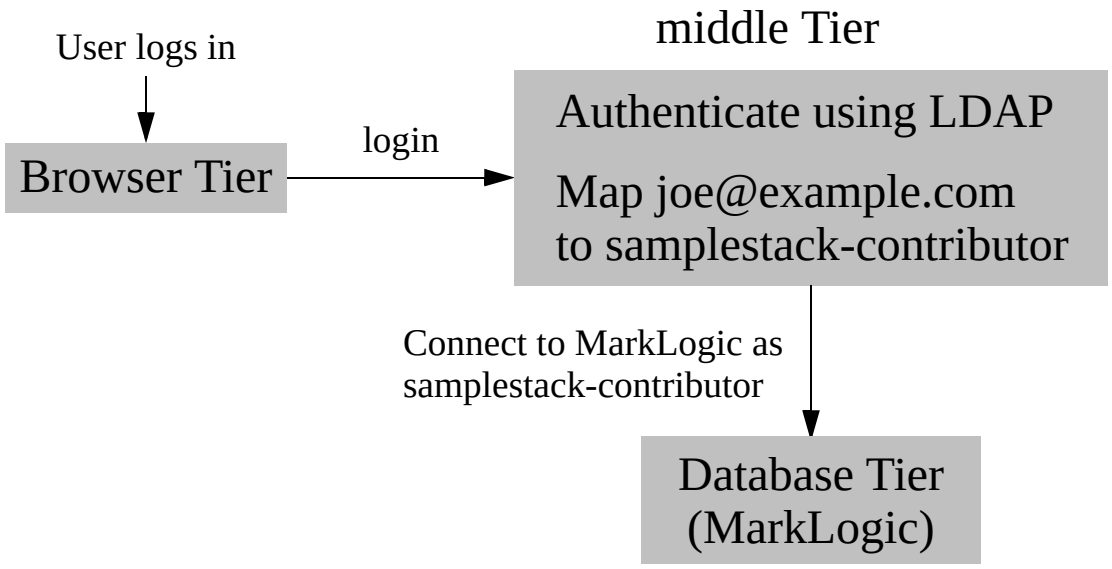
The user interactions in Samplestack support the following conceptual roles:

- Guest: Users who are not logged in can explore questions and answers, but they cannot vote or submit questions, comments, or answers. A guest user cannot see questions that do not yet have accepted answers.

- Contributor: Users who log in gain the ability to vote and to submit questions, comments, and answers. An authenticated contributor can also accept a best answer to his or her submitted question.
- Administrator: A user with administrative privileges has all the contributor privileges, plus can make configuration changes and manage the application. These capabilities are not exposed through the Samplestack UI. However, the project infrastructure uses this role to configure MarkLogic Server and deploy the application.

When you set up Samplestack, one Contributor user, 'joe@example.com", and one Administrative user, "mary@example.com", are pre-defined. For credential details, see the `marklogic-samplestack` project on GitHub.

When a user logs into Samplestack, the middle tier authenticates the credentials using LDAP and maps the user to a shared MarkLogic user with appropriate privileges. The following diagram illustrates this flow for "joe@example.com", a user with Contributor rights.



The Samplestack setup process creates MarkLogic users and security roles that support the Samplestack user model. For example, setup creates the following roles.

Role	Compartment	Description	Roles
samplestack-guest		Role for access to limited set of questions and answers, read-only	rest-extension-user
samplestack-writer		role for contributing to samplestack corpus	rest-writer, samplestack-guest

Only users with the `samplestack-writer` role (or equivalent privileges) can use write operations of the MarkLogic client APIs because only this role includes the `rest-writer` role. When you install MarkLogic, the `rest-writer`, `rest-reader`, `rest-admin`, and `rest-extension-user` roles are available for managing access through the MarkLogic client APIs.

The Samplestack setup also creates the following MarkLogic users and assigns them roles appropriate for the access granted to each type of user. The `samplestack-guest` user has only the `samplestack-guest` role, while the `samplestack-contributor` user has both the `samplestack-guest` and `samplestack-writer` roles.

User	Description	Roles
samplestack-admin	user that can administer REST server	rest-reader, rest-extension-user, manage-user, rest-admin, ...
samplestack-contributor	user for write unrestricted access to samplestack data	rest-reader, rest-extension-user, samplestack-guest, samplestack-writer, ...
samplestack-guest	user for read-only, restricted access to samplestack data	rest-extension-user, samplestack-guest

The `samplestack-admin` user has roles that allow use of the REST Management API, as well as the `rest-admin` role that allows use of the administrative endpoints of the MarkLogic client APIs. This enables the `samplestack-admin` user to perform operations such as database configuration and installation of any application code that runs inside MarkLogic.

3.2.5 Document Model

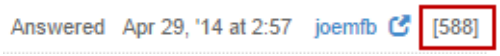
The Samplestack data consists of two types of JSON documents:

- Q&A documents. Each document contains a question, its answers, and any comments.
- Contributor documents. Each user known to the application has a corresponding contributor document that tracks information such as display name, votes, and reputation.

The contributor documents represent Samplestack domain objects. That is, the middle tier uses the POJO Binding feature of the MarkLogic Java Client API to manipulate contributor data directly as the Java class `com.marklogic.samplestack.domain.Contributor`.

The Q&A documents are larger and more complex than the contributor documents. Therefore, changes to the Q&A documents are made through the document patch feature of the MarkLogic Java Client API. For details, see “Document Insertion and Update” on page 19.

The reputation for each contributor is stored in the contributor document. However, when displaying a question with all its answers and comments, contributor reputation is included in the information display about each user. For example, the following contributor information, which is displayed with an answer, reflects a user with a reputation of 588:



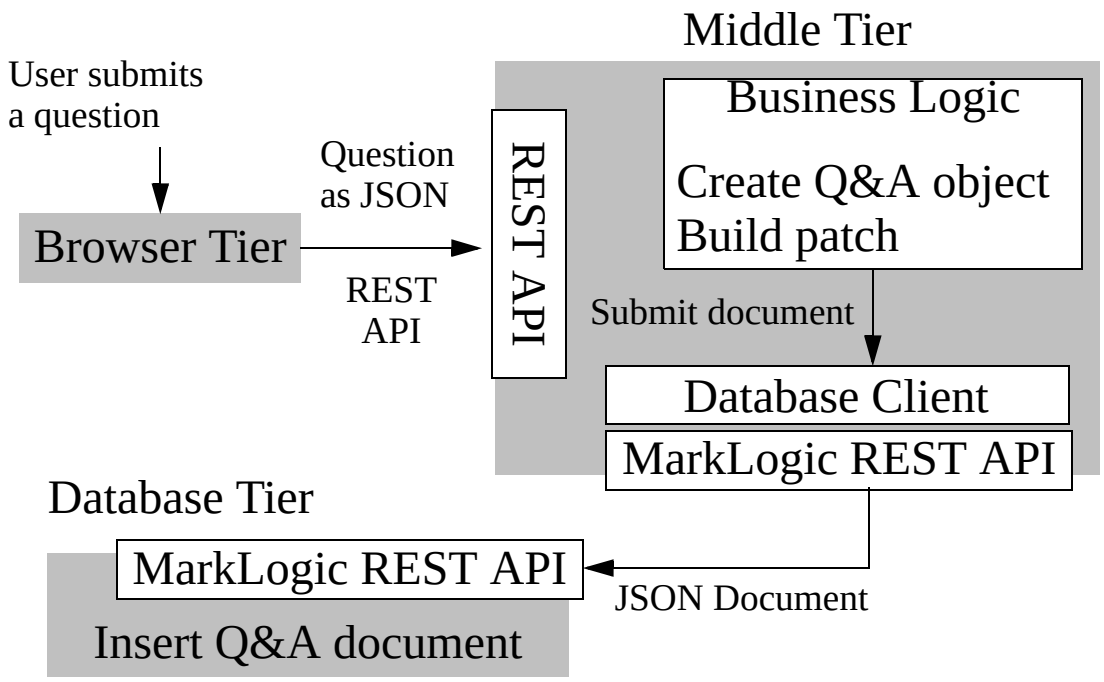
Joins should be performed close to the data, so an application-specific transformation installed on MarkLogic joins reputation data from contributor documents to Q&A data when a request is received for a specific Q&A document. A similar join is performed when retrieving search results.

The JSON received from the browser tier when users submit questions, answers, and comments is minimally modified by the middle tier. Those modifications that are made are consumable by the browser tier in subsequent fetches, so the data can be fetched from the database and passed back to the browser tier very efficiently.

3.2.6 Document Insertion and Update

When a contributor submits a question, a Q&A document is inserted into the database. The Browser Tier submits the contributor and question to the middle tier as JSON, using the application-specific REST API. The middle tier adds additional information, generates a database URI for the document, and then submits the complete Q&A structure to the database tier through the MarkLogic client API.

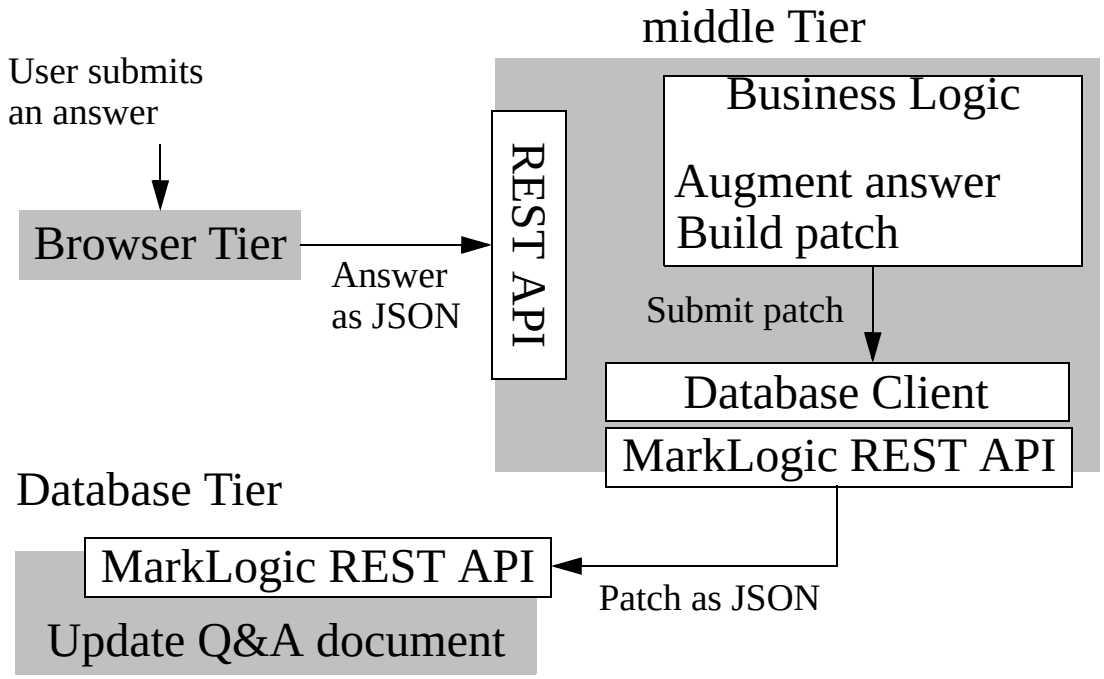
The following diagram illustrates the question submission flow.



When a user answers a question, the answer is added to the associated Q&A document in the database. Recall that a question, its answers, and related comments are stored in a single document.

In this flow, the browser tier submits the answerer, question id, and answer to the middle tier as JSON, using the application-specific REST API. The middle tier adds additional information to the answer, and then inserts the answer into the Q&A document using the patch feature of the MarkLogic Client API. The Q&A document need not be fetched from the database tier to perform this update. Submitting comments follows a similar flow.

The following diagram illustrates the answer submission flow.



3.2.7 Transactions and Data Integrity

Accepting an answer requires updating both a Q&A document and a contributor document. When a submitter accepts an answer to his or her question as the “best” answer, the Q&A document is updated to reflect the acceptance. At the same time, the reputation of the user who contributed the answer is incremented.

The update of the Q&A document and the contributor reputation should be an atomic operation. The answer should not be accepted without updating the reputation, and the reputation should not be incremented without accepting the answer.

Data integrity is preserved by using a MarkLogic multi-statement transaction. When the middle tier receives the answer acceptance from the browser tier, it requests the database client to create a multi-statement transaction on MarkLogic Server. The database client returns a transaction id. By including this transaction id in the update operations for the question and the contributor, the middle tier ensures both operations are part of the same transaction.

When both update operations complete successfully, the middle tier commits the transaction. If either operation fails, the middle tier rolls back the transaction.

3.3 Best Practices Demonstrated by Samplestack

The Samplestack project infrastructure demonstrates the following best practices recommended by the MarkLogic Reference Application Architecture:

- Project Organization
- Source Control and Issue Tracking
- Configuration and Dependency Management
- Task Automation
- Testing

3.3.1 Project Organization

If you explore the organization of the `marklogic-samplestack` project on github, you will see that the top level of the project is divided into a folder for each tier, plus a folder for shared assets:

```
marklogic-samplestack/
  appserver/
  browser/
  database/
  shared/
```

The folder for each tier contains the source code, tests, configuration files, and automation drivers required to build, test, and deploy that tier. To explore the project organization in detail, go to the following URL:

http://github.com/marklogic/marklogic-samplestack

3.3.2 Source Control and Issue Tracking

The Samplestack project uses GitHub and `git` for source control management. All the assets required to build, test, and deploy the application are under source control, except the large sample data set.

GitHub includes an issue tracking system, and the Samplestack development team uses it to track bugs, tasks, and open design issues.

3.3.3 Configuration and Dependency Management

The Samplestack project makes extensive use of configuration properties files to drive build, testing, and deployment.

For example, `marklogic-samplestack/appserver/java-spring/gradle.properties` contains configuration properties required for the Java middle tier to connect to MarkLogic Server, including the MarkLogic usernames, and the host and port information used to connect to MarkLogic. These same properties are used for setting up the database tier and testing. Thus, changing this one file and re-running the setup scripts is all that's needed to use a different MarkLogic instance or connect as different users.

To view this file on GitHub, go to the following URL:

> http://github.com/marklogic/marklogic-samplestack/blob/master/appserver/java-spring/gradle.properties

Similarly, `marklogic-samplestack/appserver/java-spring/build.gradle` contains all the external dependencies required by the Samplestack middle tier, and the browser tier maintains a similar list of dependencies in `marklogic-samplestack/browser/bowser.json`.

In the database tier, the MarkLogic database configuration details are maintained in `marklogic-samplestack/database/database-properties.json`, in a format suitable for passing directly to MarkLogic using the MarkLogic REST Management API.

3.3.4 Task Automation

The setup, build, deployment, and testing of Samplestack is fully automated. The tiers do not all use the same automation tools because they have different requirements.

The database tier and the Java implementation of the middle tier use the build automation tool `gradle`. The Samplestack `gradle` configuration defines a wide variety of task necessary to bring up and test the middle tier, including the following:

- Creating the MarkLogic roles, users, and REST API instance (App Server and database) required by Samplestack.
- Configuring database indexes and other properties.
- Loading the database tier Samplestack assets into the modules database on MarkLogic Server. This includes resource service extensions, content transformations, and persistent query options.
- Loading seed data into the database.
- Building the Java implementation of the middle tier.
- Starting the middle tier.
- Testing the middle tier.
- MarkLogic teardown - removing the REST API instance and database.
- An umbrella task that combines all of these steps into a single step that takes you from a freshly installed MarkLogic to a running, smoke-tested Samplestack middle tier.

For more details, see the following page:

http://github.com/marklogic/marklogic-samplestack/tree/master/appserver/java-spring

Since the browser tier is implemented in JavaScript and Node.js, it uses a different set of tools for similar purposes: `npm`, `bower`, and `gulp`. For details, see the following page:

http://github.com/marklogic/marklogic-samplestack/tree/master/browser

3.3.5 Testing

The browser and middle tier each have a set of unit tests that can be run independent of the other tiers. These tests include scaffolding that mocks up adjacent tiers.

For example, the middle tier includes JUnit-driven unit tests for the Java implementation that are run as part of the middle tier start up task (the `appserver` and `assemble` gradle tasks). This provides a sanity check each time you deploy a change.

The middle tier also includes integration tests that test interactions with each tier independently. That is, interactions between the middle tier and the database tier, and interactions between the middle tier and the browser tier.

The browser tier tests includes end-to-end tests that exercise the whole application stack. These tests use Selenium to drive the browser, and test tools such as `gherkin`, `cucumber`, and `protractor` to define the behaviors and expected results. For details, see the section on testing on the following page:

http://github.com/marklogic/marklogic-samplestack/tree/master/browser

To further explore the testing in Samplestack, see the following parts of the project:

- `marklogic-samplestack/appserver/java-spring/src/test`
- `marklogic-samplestack/browser/test`

3.4 Technologies Used in Samplestack

This section summarizes some of the technologies used to implement, build, test, and deploy Samplestack. Using these technologies illustrates how easily you can implement the recommended best pratices and integrate MarkLogic into your development process using industry standard technologies.

Your application does not have to use the same tools and frameworks in order to use MarkLogic.

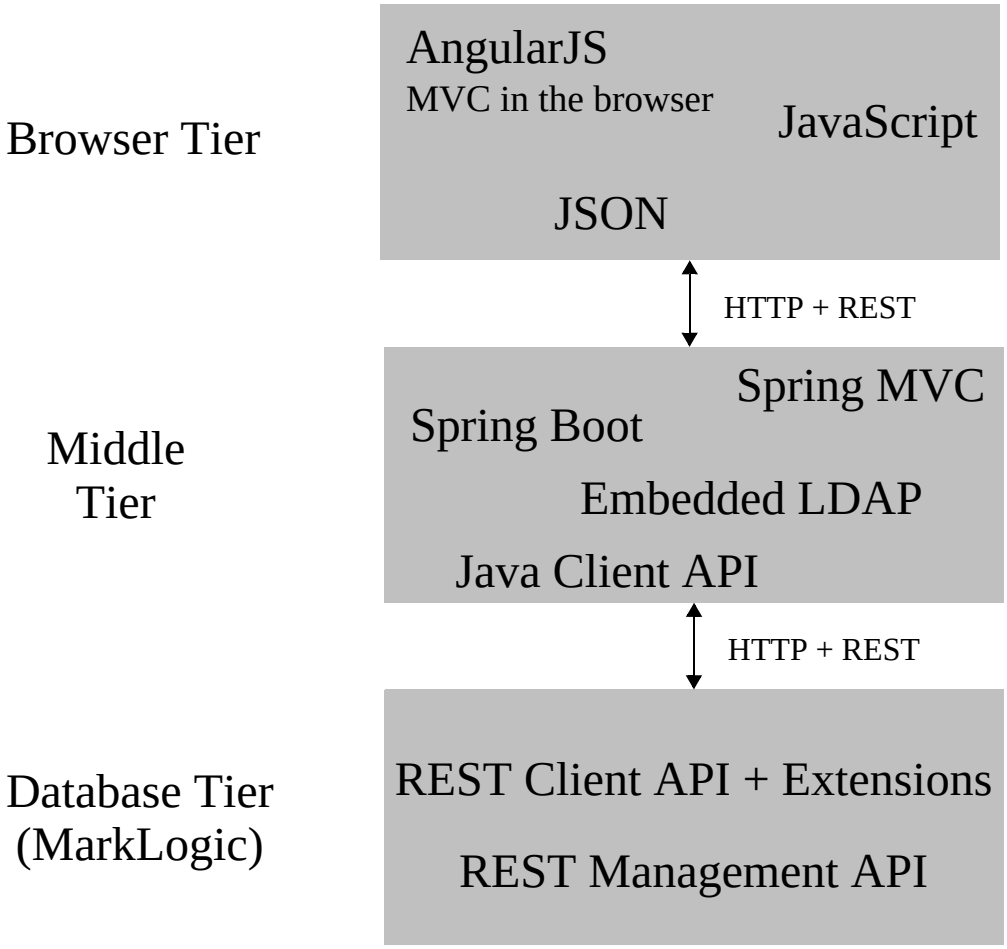
This section covers the following topics:

- Samplestack Implementation

- Samplestack Build, Test and Deployment Automation

3.4.1 Samplestack Implementation

The implementation of Samplestack uses key MarkLogic features alongside technologies such as the ones shown in the following diagram::



3.4.2 Samplestack Build, Test and Deployment Automation

The Samplestack project infrastructure demonstrate development best practices such as source code management, automated builds, automated unit, integration, and system testing, and configuration-driven application deployment.

The following tools and technologies are used to configure, build, test, and deploy Samplestack:

- Source control and issue tracking: GitHub
- Build, test, and deployment process automation: gradle, gulp, bower, npm
- Testing: JUnit, Selenium, mocha, PhantomJS, cucumber, protractor, gherkin,

- Application configuration and deployment: MarkLogic REST Management API, mlcp (MarkLogic content pump)

3.5 Exploring Samplestack in Detail

To try Samplestack or explore its implementation in more detail, see the `marklogic-samplestack` project on GitHub:

http://github.com/marklogic/marklogic-samplestack

The project page includes directions for building and setting up the application. The project wiki includes more information on the implementation, tests, and tooling.

4.0 Technical Support

MarkLogic provides technical support according to the terms detailed in your Software License Agreement or End User License Agreement.

We invite you to visit our support website at http://help.marklogic.com to access information on known and fixed issues, knowledge base articles, and more. For licensed customers with an active maintenance contract, see the Support Handbook for instructions on registering support contacts and on working with the MarkLogic Technical Support team.

Complete product documentation, the latest product release downloads, and other useful information is available for all developers at http://developer.marklogic.com. For general questions, join the general discussion mailing list, open to all MarkLogic developers.

5.0 Copyright

MarkLogic Server 9.0 and supporting products.
Last updated: April 28, 2017

COPYRIGHT

This technology is protected by U.S. Patent No. 7,127,469B2, U.S. Patent No. 7,171,404B2, U.S. Patent No. 7,756,858 B2, and U.S. Patent No 7,962,474 B2, US 8,892,599, and US 8,935,267.

The MarkLogic software is protected by United States and international copyright laws, and incorporates certain third party libraries and components which are subject to the attributions, terms, conditions and disclaimers set forth below.

For all copyright notices, including third-party copyright notices, see the Combined Product Notices.