Crowd/tidbits
COS 790
Architecture Specification

Collen Mphabantshi
U10404687

Sydney Chadwick
U29143595

Contents

Introduction

Purpose of document

This document aims to provide an architectural overview of the Crowd/tidbits system, and to describe the different aspects of the system with regard to architectural decisions made on the structure of the system.

Scope of project

The system needs to provide a mobile application allowing users to upload messages that are relevant to a specific location. To this end, an iOS/Android application, a back-end HTTP server, and a persistence unit.

Access channels

The system provides two points of access; both human access channels; namely the application proper and a web based CMS allowing administrators to interface with the database.
The application needs to run on mobile devices, and must have support for iOS version 7 and above, as well as Android version 3 and above. Although this is not strictly a requirement, adhering to the design guidelines of each operating system will be advantageous and is recommended.
The web front end, containing the CMS, will be a module that forms part of the back end server. This will run as a servlet, which is accessed through the server URL. Given that only system administrators will have access to this interface, having it run as part of the server will not have any negative impact.

Integration channels

Internal

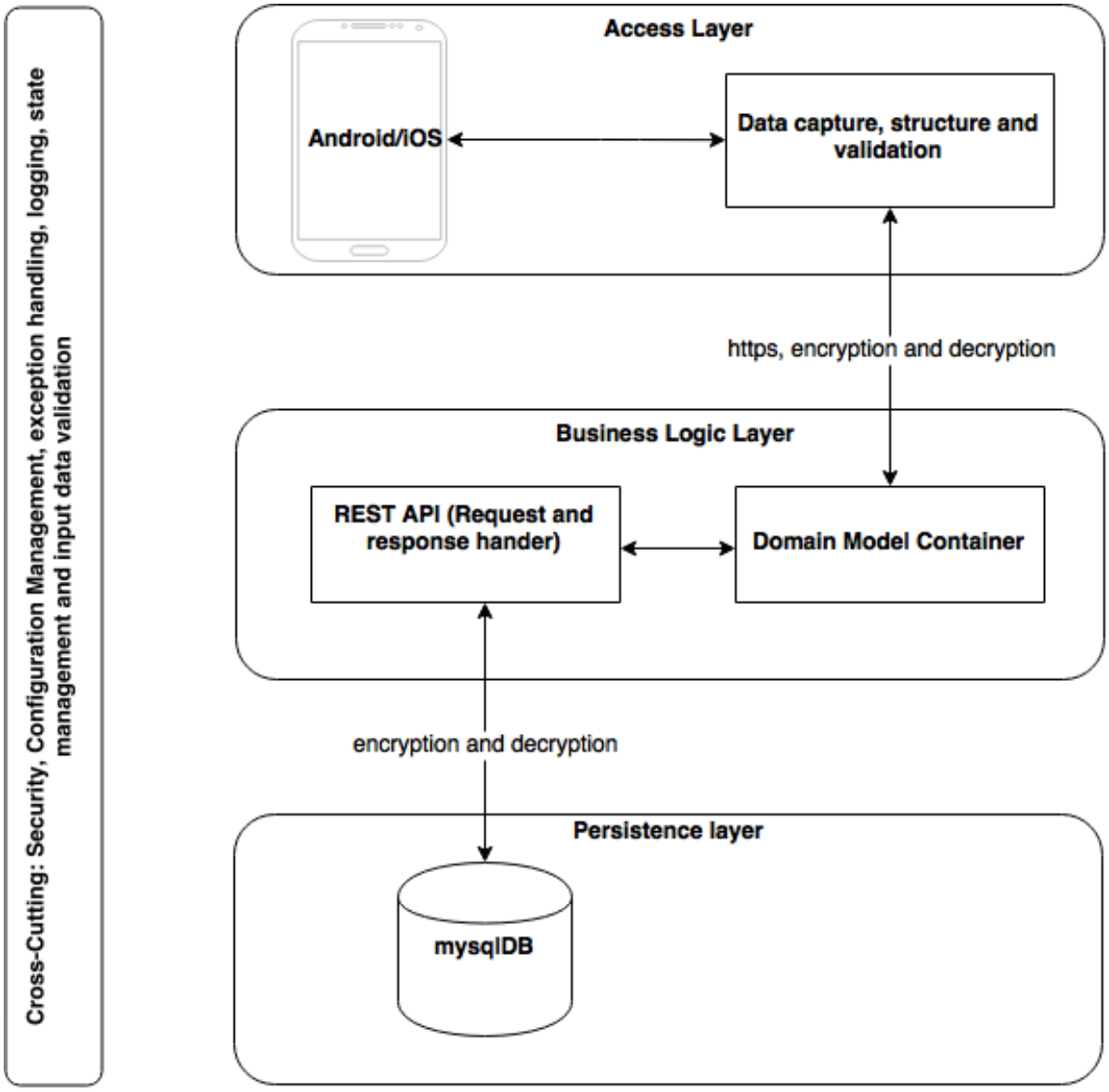
Two internal integration interfaces are required. The server needs to integrate with the DBMS, as well as with the application.
The server-application integration will take place over HTTP POST calls, and therefore security is paramount to the implementation of the communication. Making secure calls with HTTPS, along with hashing of secret data such as passwords, are both requirements that must be taken into consideration.
The server and the DBMS are likely to run on the same server, so the communication channels between the two need not be extremely secure; however, to prepare for future scaling secure channels should be considered. Integration will take place using the Hibernate entity management framework for Java, which also protects against SQL injection attacks.

External

The application will require integration with a Maps API platform, in order to display the “tidbits” as nearby points on a map, as well as provide an overview map containing all tidbits on the screen area.

Architectural responsibilities

- Log all requests to a separate log file, for ease of searching
- Persist information received from users
- Communicate with Maps API
- Keep transferred data to a minimum
- Secure transfer of information
- Cater for intermittent signal of mobile data services



Quality requirements

Scalability and performance

Since this is a proof of concept application, scalability is not really something that needs to be considered at this time. However, the server should provide concurrent HTTP connections and perform intensive actions asynchronously where possible to improve performance. The database should also have a connection pool, which is supplied by the application container. Allowing multiple connections makes the system more scalable, whilst improving performance at the same time. It is easier to develop with scalability in mind from the start, than to optimise at a later stage.

Maintainability

Although change is not expected to occur often, the code should be easily maintainable. Using a dependency management framework, such as Maven, will make keeping libraries up to date simple. Following what is generally accepted as good coding standards, along with the addition of code comments and Javadoc will improve maintainability should the developers working on the system change. Properties in the system that are likely to change often, such as URLs and system settings should not be hard coded; but should use a common configuration file that will allow values to be changed without having to modify the code and redeploy the application.

Security

Only application and database security are covered explicitly. The security involved for communication with the server is included under the heading of each.

Application

Access to the application should be restricted, and a user must have a valid account in order to use the application. Any private data stored on the device, such as credentials, need to be encrypted to protect user data should the device the compromised. Communication with the server needs to securely transport credentials by making use of hashing algorithms, and HTTPS calls for the actual data transfer.

Database

The POPI act forbids the storage of any information that can be used to identify an individual. To comply with these requirements, all personal information (e.g. email address, MSISDN, name) needs to be encrypted within the database. The use of the

Hibernate framework on the server reduces the risk of SQL injection attacks, and also the potential of developer error in SQL commands resulting in security risks.

Monitoring and auditability

Monitoring of the system should not be required, however auditing is of paramount importance. This will involve multiple logging files, at the very least an error log and a request log. The request log has to contain enough information to be able to trace those requests made by a certain number, however POPI compliance must still be taken into consideration. Use of a rolling log which maintains logs up to a certain age, and does automatic cleanup of old files will improve maintainability of the system.
The Hibernate framework supplies the capability of automatically generated database audit tables, which also maintain a history of database edits. Requests should not be logged in the database, as this will result in rapid growth in size; which is not recommended.

Testability

Unit tests are required to ensure proper functionality of the code, as well as integration tests which will determine if the database access and web server calls function correctly. Testing should not modify the production environment, so mocking the database and the servlet in unit tests are essential.

Usability

The application needs to be used on mobile devices, which have intermittent and unreliable internet connections. Thus, the application must cater for requests and responses getting lost. Implementing retries from the application, along with checks for duplicate requests on the server side. The application also needs to be implemented in such a way that it is easy to use with a touch screen.

Architecture constraints

The technologies that will be used are various, and the following should not be considered an exhaustive list.

Application

- Cordova
- AfriGIS Maps API

Server

- Apache Commons Libraries
- Glassfish application server

- Java
- Hibernate entity framework
- Slf4j + Logback
- Spring entity injection
- Vaadin

Database

- MySQL

Acronyms

API - Application Program Interface
CMS – Content Management Service
DBMS – Database Management Service
HTTP – Hypertext Transfer Protocol
HTTPS – Hypertext Transfer Protocol Secure
MSISDN – Mobile Station International Subscriber Directory Number
POPI act – Protection of Personal Information act
SQL – Structured Query Language
URL – Uniform Resource Locator