CS 7325 Project

Supplementary Documentation

Msc SAI

Course: CS 7325

Basic requirements

The goal of the project is to create an internet based notification system between professors and students. The professors can use a web based service to send messages to the smartphones of selected registered students. The initial requirements do not cover sending notifications from students to professors.

Simple explanation from user perspective

The web service.

- A web service will be created, intended to be used by the professors. By logging in the service (using a predefined user name and password), the professor will be presented by a control panel showing the currently registered students of the service. By selecting some students from a list and writing a message in the appropriate field, this message is sent to the smartphone devices of these students. Using the same web service, the professor will also be able to delete students from the service.

The smartphone applications.

Smartphone applications will be created for the students who own an android and / or iOS device. For the initial prototype of the service, only an iOS application will be delivered.

The iOS application will be capable of:

- Registering the student (with the device he/she uses) to the web service as a student who wants to receive notifications. **NOTE**: e-mail verification is required as an intermediate step, see the registration system section for more details.
- Receiving notifications sent by the users of the web service.
- Displaying the messages that have already been received.
- Receiving push notifications (notifications that can be seen when the application is not in the foreground).

The registration procedure:

To verify student authenticity, a registration procedure will be created, which requires the student's name, lastname, and a valid e-mail address. During the first run of the application the client application will present the user with a signup form asking for all these information. When this information is given, an e-mail is sent to the user's e-mail address which will include a link to be pressed to finalize the registration procedure. When the user clicks the link in the e-mail, the registration procedure is completed, and the user can return to the application and start actually using it.

Technologies and frameworks used:

iOS:

- **iOS 5.1+ compatible**. Although for applications such as this version 4.0 + can be used, it is now obsolete and debugging tools for such an old version cannot be found.
- **AFNetworking** (https://github.com/AFNetworking/AFNetworking) . A widely used framework which serves as an Objective C block-based wrapper on Apple's NSURLConnection class, that facilitates multithreaded networking implementation. The reasoning behind this choice is:
  - Widely used: AFNetworking is used in many commercial applications, so technical support and examples are easy to find.
  - Block-based operations. Completion blocks can be called efficiently.
  - NSOperation based. Because the basic networking class is a subclass of NSOperation, implementing multithreading paradigms becomes easier
  - Has support for different types of error codes caused by network failure.
  - Prior experience by the developer of the project.
- **ARC** (http://en.wikipedia.org/wiki/Automatic_Reference_Counting). Because it speeds up development by a great factor, without compromising performance and flexibility.
- **Expecta Framework** (https://github.com/petejkim/expecta) for iOS unit testing
- **XCode 4.4+.** For support for automatic property synthesis (http://useyourloaf.com/blog/2012/08/01/property-synthesis-with-xcode-4-dot-4.html) and support for Objective C literals (http://joris.kluivers.nl/blog/2012/03/13/new-objectivec-literal-syntax/) which speed up development and make code less cluttered.
- **Core Data** for handling offline saving of data to the client. Core Data is a relational mapping database protocol specific to Apple devices. Core data has some limitations when compared to using a plain and barebones SQLite database, but those limitations are not enough to make it unusable, or incapable of making the project scalable. In addition, it provides a concrete wrapper for database operations, like saving and querying, facilitating asynchronous access to the saved data.

Web Service

- **Java Servlet technology (Servlet v 2.3+, Java 1.6+)**, because of its flexibility, scalability and knowledge of Java by the developer (which is better than the alternative - PHP). The servlet version was chosen because of the inclusion of Filters, which in our case facilitate access control to pages (among having other useful capabilities).
- **MySQL** as a relational database, because of its simplicity to set up, documentation, and existing supporting sites. However, MySQL is not a requirement, as the code is portable enough to work in other relational databases with little or no changes.
- **Tomcat** as a server testing environment. Tomcat is used for performing the tests locally, because of its simplicity to set up, its price (free), and a plethora of development resources on the Internet. However, no Tomcat-specific features have been used, to make the web application as portable as possible.
- **Plain JDBC Connector** as a framework to make SQL queries. The barebones framework is being used, and wrappers (who may have sped up development) are avoided (i.e. Hibernate). The decision was made because of the reluctance of the developer to add an abstraction layer upon plain MySQL queries, which would make it difficult to debug the web application in case problems would arise.
- **BoneCP** as a DataPool for reusing JDBC connections. The choice of having a datapool in the first place is still being examined. The choice of the actual framework was made because out of all the

alternatives examined (C3p0 and DBCP), BoneCP was simpler to implement and use, faster, and was actively developed (which is not the case with C3p0).

- **Maven**. The decision to use it was made halfway through the project, when the need for automatic dependency management was high. Also, because Maven expects files to exist in predefined and appropriate places, the need for an Ant file for building the final .war file is diminished (although may not gone in the end product).
- **Eclipse**. Because of 3rd party support, numerous plugins, community adoption (= documentation, internet resources) and prior experience of the developer.
- **JSON** (Google GSON library): JSON was chosen because of its facility to produce readable and easily parseable feeds. It is also very easy for consumption in iOS devices, as Objective C's NSdictionary class is interchangeable with JSON.

Deployment: Getting Started

A comprehensive read-me is included at the root folder of the project, in MarkDown format. The latest version of the read-me can be read at

https://bitbucket.org/csotiriou/sai-csc7325-project

A few architectural explanations

In this section, some architectural decisions and methods are explained

CC implementation

The service implements CC support. Each time a user of the web service sends a message to multiple recipients, the full list of recipients will be visible to the users of the mobile devices. **The service does not implement Bcc support** for two reasons:

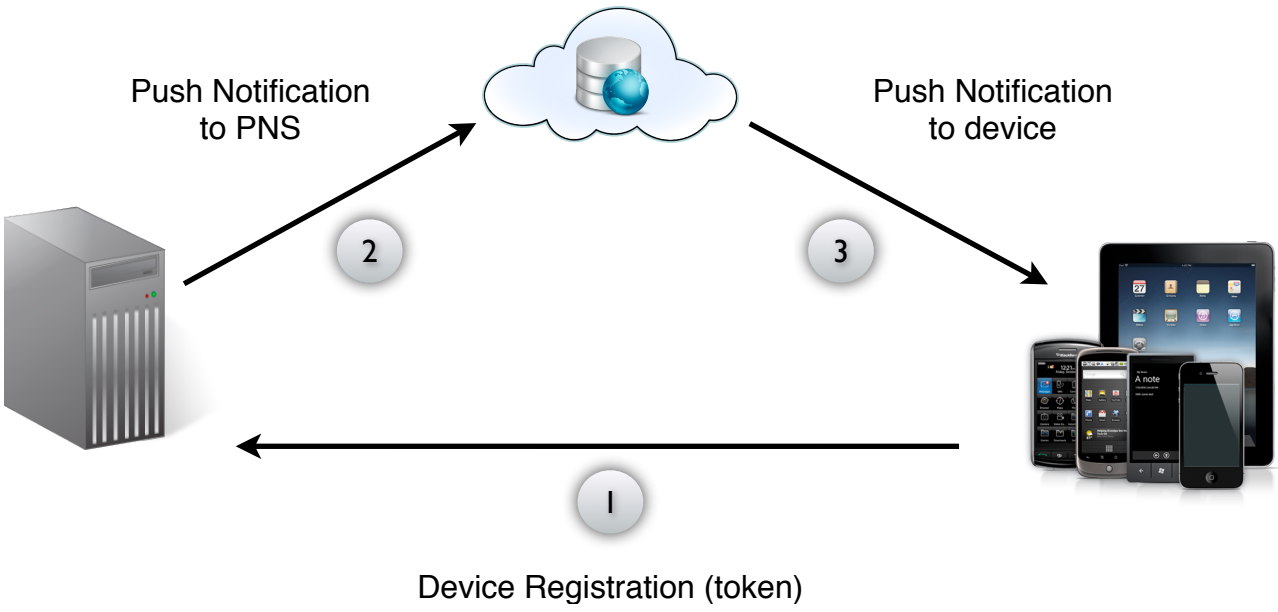
• It is out of the scope of the project. This project is about implementing a notification system between the students and the professors. It does not intent to provide e-mail - like support.
• Time constraints.

System limitations / Disclaimers

- A mobile device cannot be registered to two students. For registration to succeed, the device must be deleted from the database, so that it can be re-added and associated with another student.
- CC is implemented as an obligatory field. Bcc is not implemented. Any message that has multiple recipients is going to be shown as CC in the mobile device of the client.
- The system is not yet tested in a real environment. So far, development is being done in a local server.
- The mail confirmation system will have a high chance of failing if deployed in a web server. Google will suppress mail messages sent by other not known servers.
- There is no “un-registration” support on the client side yet.

Push Notification System Internals

A web server cannot send a push message to a mobile device directly. All push requests are sent to a push notification server, owned by Apple or Google (depending the target device), and are then forwarded to each device.



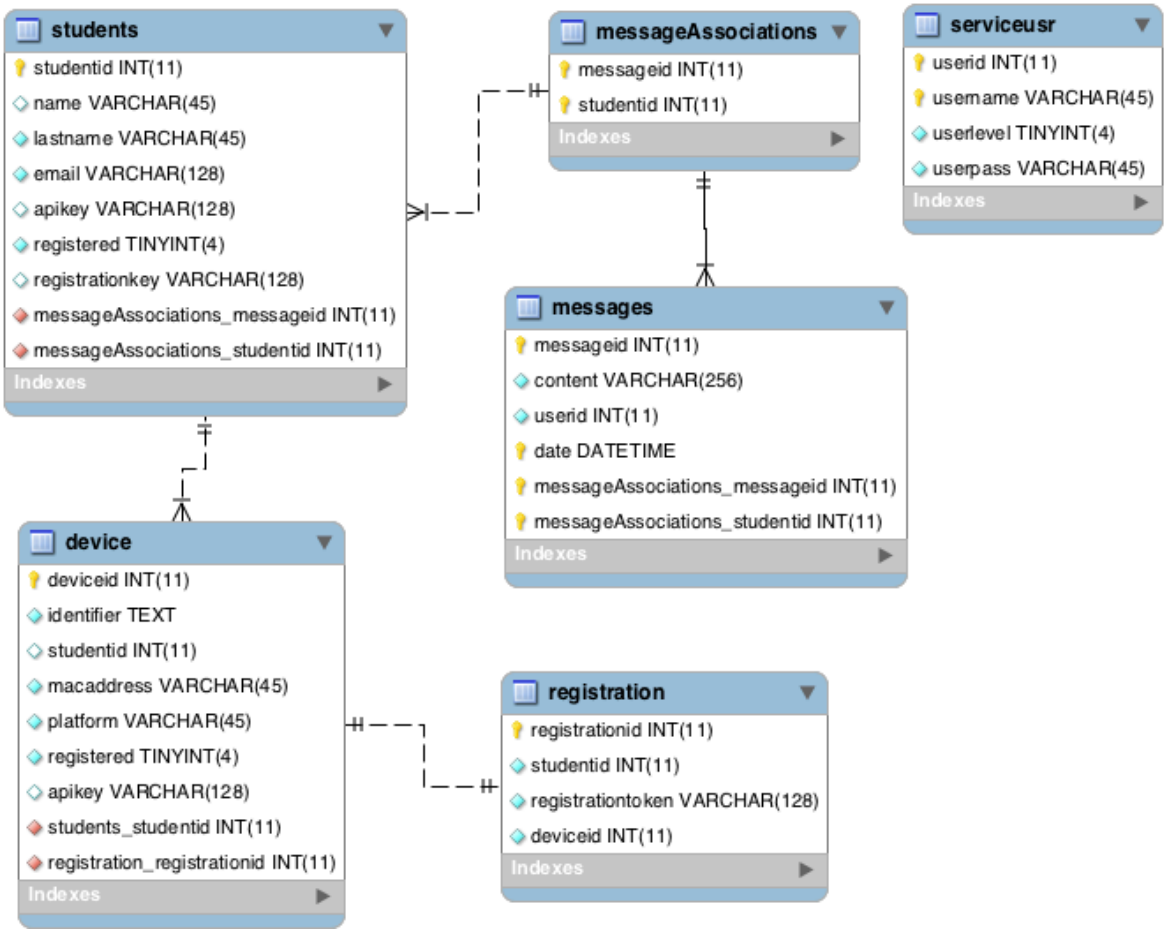
In order to do that, a web server must have a unique push notification identifier, associated with a device, when the latter registers itself with the push notification service.

The procedure in order to set up a push notification system is as follows:

• The application is sending its unique push notification to the server.
• When the server wants to send a push notification, it contacts Apple's or Google's services with one or more device tokens
• The push notifications service sends the messages to the corresponding devices.

The application is then responsible for sending this token to the web service, so that the latter can use this token as an argument when sending push notification messages, so that the messages can be delivered to the intended destinations.

Database Structure



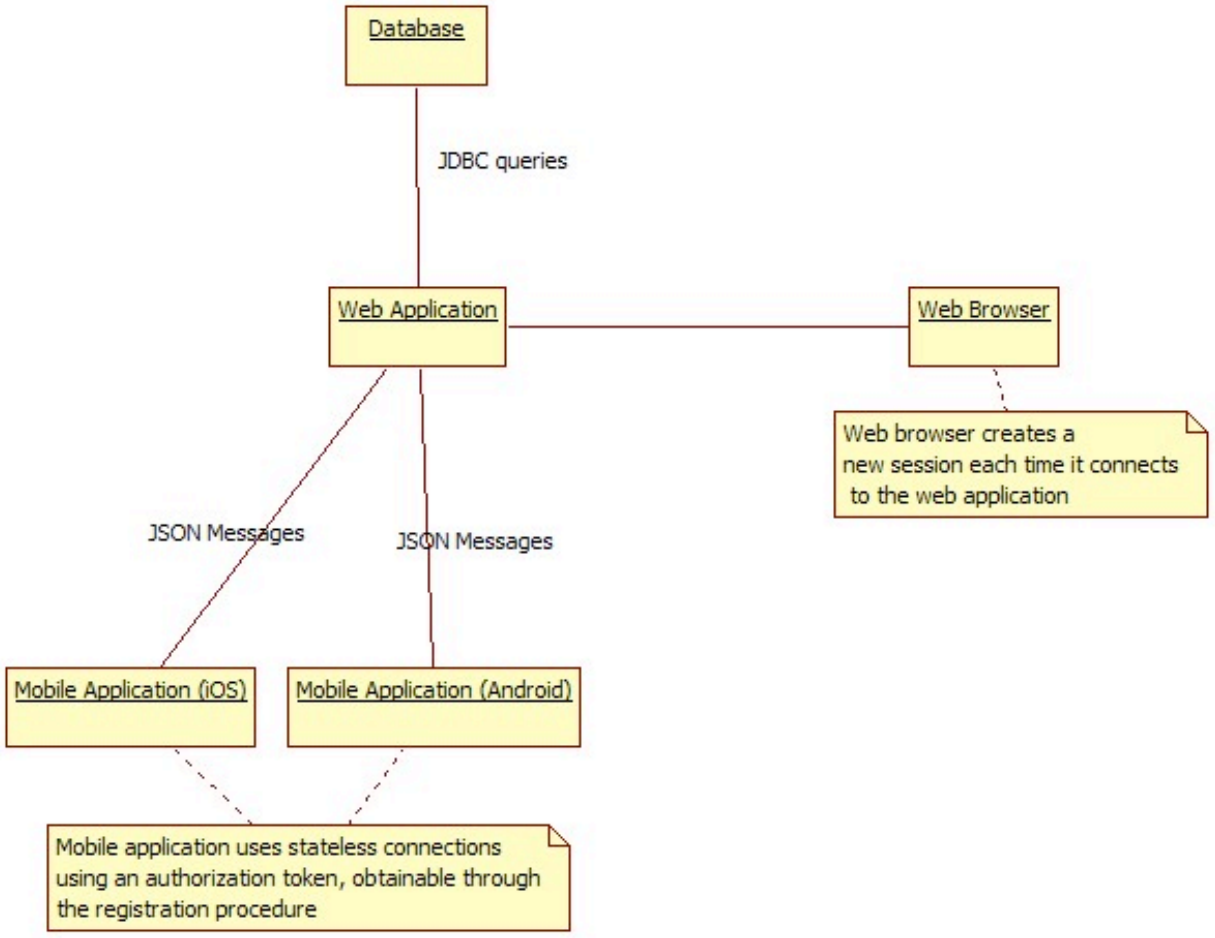
Parts of the database that are obsolete or deprecated are not included in the diagram.

- **students**: Represents the entry of a student in the database. A student is being recognized by its e-mail, which must be unique in the database. <<Registration key and apiKey are obsolete>>
- **device**: Represents each mobile device registered in the system. A device must belong to a registered student, hence the `studentid` entry. A student can have zero or more devices in his/her name. **Each device has an API key which it uses to make requests to the web application**. Therefore, an API key is specific to a student's device, and not to a student.
- **registration**: Entries in the table are being created when a mobile device tries to login for the first time in the system.
- **messages**: Entries in this table represent a message send to a student.
- **messageassociations**: An intermediate table for implementing the an e-mail-like CC feature. A single message can be sent to multiple students. For each student, a messageAssociations entry is created, associating the message with the student that has received it. This methodology is necessary for knowing all the recipients of a specific message.
- **serviceusr**: represents a user that can login to the web application.

Service Specifications

Service Internals

A UML collaboration diagram is shown below.

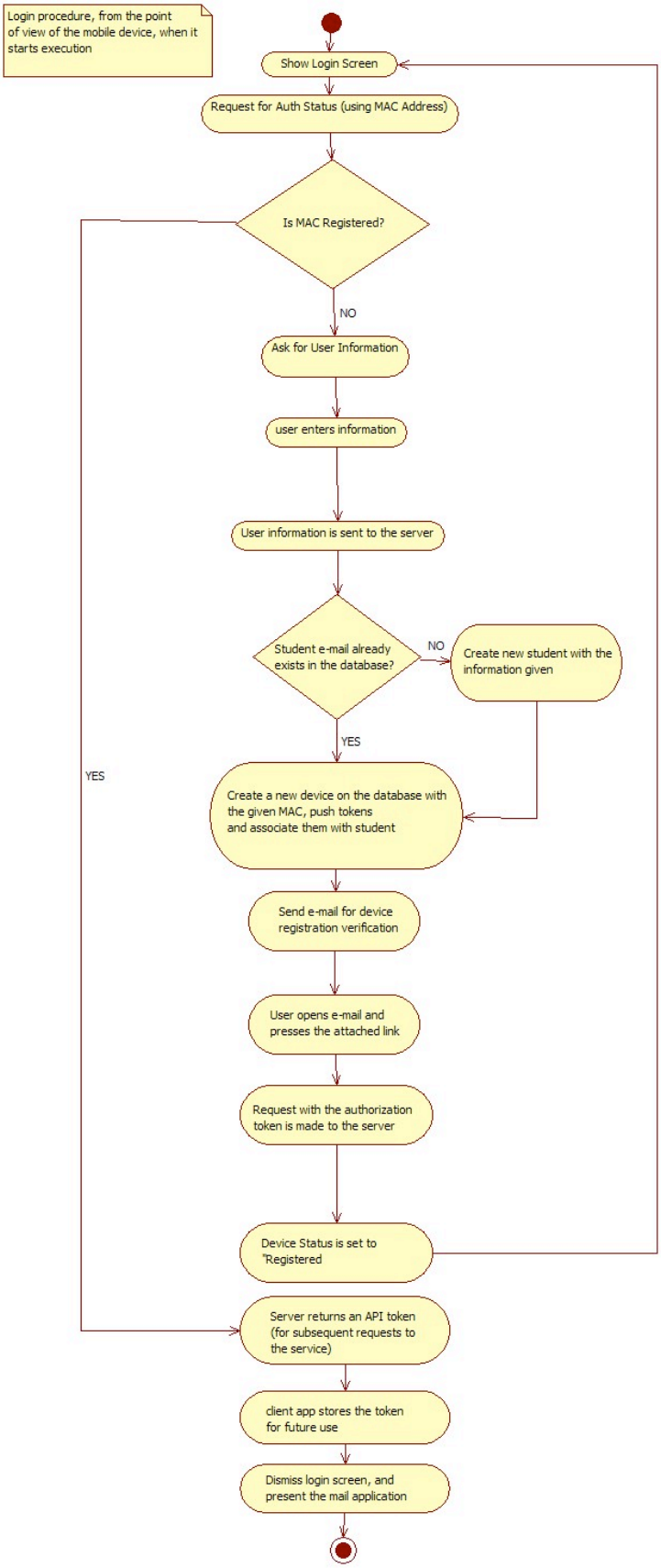


The Database is a MySQL database. Each time data need to get retrieved the web application opens JDBC connections to the server, or reuses an existing one that has been already opened (in the case where data pooling is enabled). The web application is responsible for formatting the response appropriately and sending the result to the end point which has requested the information.

The connections made by the mobile client are stateless. No session is opened. When logging in through a web browser, a session is opened, and is closed when the user logs out the application, or when the session times out.

Login procedure by the mobile client

For every request to the service, the mobile client must have an API key. The API key is obtainable by the procedure below:



At this prototyping stage, the API key is not saved in the device, but is being requested by the application during each startup.

If this device is already registered, this device's API key is returned, and the registration procedure is skipped. If this is the first time this device is used to login to the service, the registration procedure is being followed: After the student has entered the necessary information, an e-mail is being sent to the given e-mail, containing a link to be pressed. After this linked is clicked, the registration is complete, and the next time the mobile application enters the foreground, it will check for the registration status again, therefore obtaining an API key, and being able to make calls to the service.

Mail Confirmation Possible Problems

Under ideal circumstances, a confirmation e-mail is sent to the user of the mobile device to avoid spam registrations with the service. This e-mail contains a registration token which is used by the service to complete the registration.

For sending the confirmation e-mail, Google's SMTP services are used, and a TLS (optionally an SSL) connection is being created by the service to google, and sends the message using Google's server. Although this procedure works using a local server, **it will not work when the application is deployed into a production environment** (i.e. a global dedicated server), because Google's services forbid messages coming from unknown external public web servers to avoid spamming.

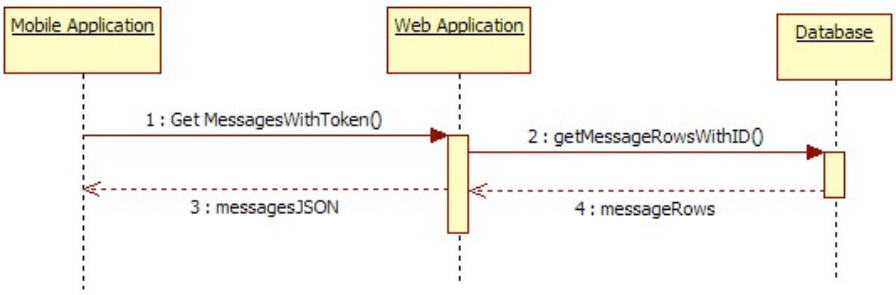
In that case, the mail will not be delivered, and the user will not be able to complete the registration procedure.

For testing purposes, a property has been created into the server's configuration file, called "allowRegistrationWithoutConfirmationEmail". When set to true, the server will automatically complete the registration procedure, in case where the mail fails to be delivered using Google's services.

This option is for testing purposes only, and leaves a very serious security opening in the service. The ideal would be to have an implementation of a mail server service to the dedicated server where the web application runs. However, due to limited resources and time constraints, it was impossible to deliver the project with these features.

Getting available messages (mobile client)

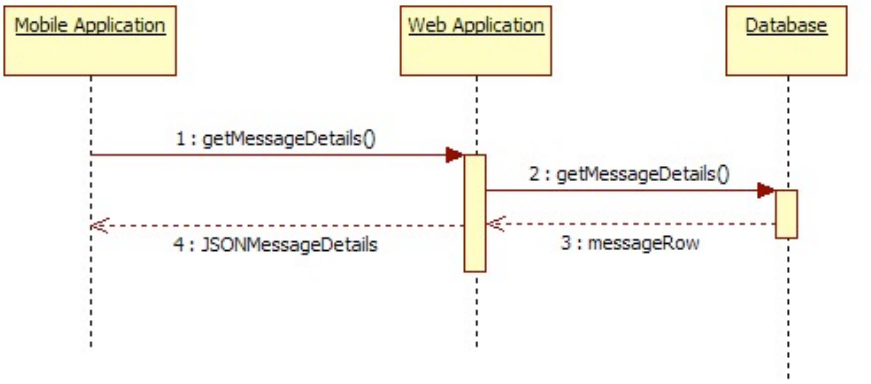
Getting the messages using the mobile client involves making a request to the web application, with an API key, which in turn will make a JDBC request to the MySQL database. The database will return the selected rows to the web application, which in turn will format the response to a JSON text, and sends it to the mobile device.



The response will not include all students that were send each message. For this information, a second procedure is needed.

Getting a message's CC status (mobile client)

The students to whom a message was sent are obtainable is obtained by a request to the web service, with this message's ID (apart from the required API key).



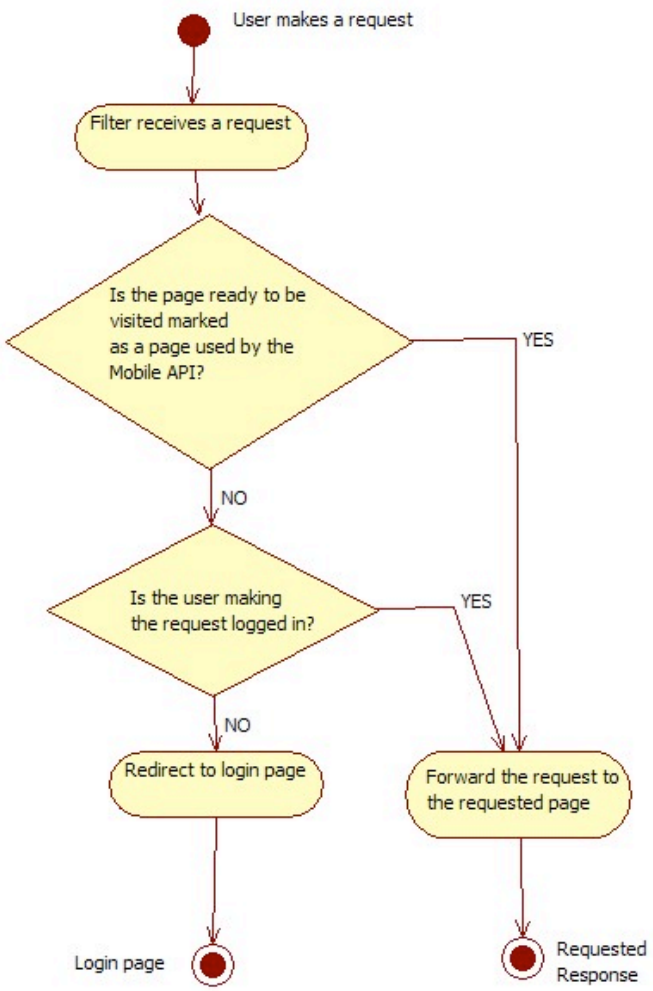
The procedure is similar to getting the available messages for a student. The JSON message returned to the client contains a JSON array with students information.

Access Control (web app)

Access control is achieved by using a Servlet Filter.

Inside the filter a list of pages is defined, where the filter will not be applied. These are considered to be the mobile API pages, where we do not want to restrict access using the filter (as a safety measure, an API token is needed when interacting with these pages). For pages and servlets visited using the web application site, the filter is applied.

A diagram showing the flow and the possible outcomes is shown below:



Service Multithreading:

For this project, a Tomcat server is used. According to the specifications, each request made by a client is ran by default in its own separate thread. However, for Servlets, each time a Servlet is being requested, it is being created only once, and is being reused for subsequent calls to the same servlet by any user and session. That means that any Servlet's instance variables must be thread-save, as they may be accessed by multiple threads at the same time.

Thread safety in the web application is achieved by making all servlets as stateless as possible. Each servlet instantiated has no instance variables, and connections to the database are done through a ServiceDao object, which is created for each request.

Future Expansion for the project

Improvements in Multiple User Support

Messages that will be shown in the message list section in the web application, could be separated according to senders, so that each user can see only the messages sent by his account. However, this is not only a technical consideration, but also a managerial one.

Security Considerations

-- Support for password hashing could be added
-- For the API used by the mobile device, a login system like OAuth can be used, to improve security.
-- A rotating token can be used. So far, the login token used by the device to use the service is constant, and is assigned during the student's registration. A more sophisticated login system could be used, to create a new token each time the student logs in, and will be valid for a certain amount of time.