**Sign up using Phone Number**

In today’s world, a user interacts with various apps/websites which require them to login or sign up via using their email address. There are a high chances of a person forgetting the username and the associated password to login or just completely giving up the sign up process as it asks a lot of information. Furthermore, from the app developer point of view, they will need to perform extra steps to verify the user’s email address.

This leads to another solution of using phone number as a way to login to the service. It is easy to remember the phone number and we can implement a mechanism to verify that the user is in possession of this number via text messages. In order to sign up with phone number the user Birth Date, Name, Gender ,Email this all need to be collected first.

* Birth Date: The user Date Of Birth must be collected that should be in dd/mm/yyyy format.
* Name: The complete name of the user is collected.
* Family Name: The complete Family details is necessary and that to be stored.
* Gender: The user gender to be collected as male, Female.
* Email: The user’s associated email address to be collected.

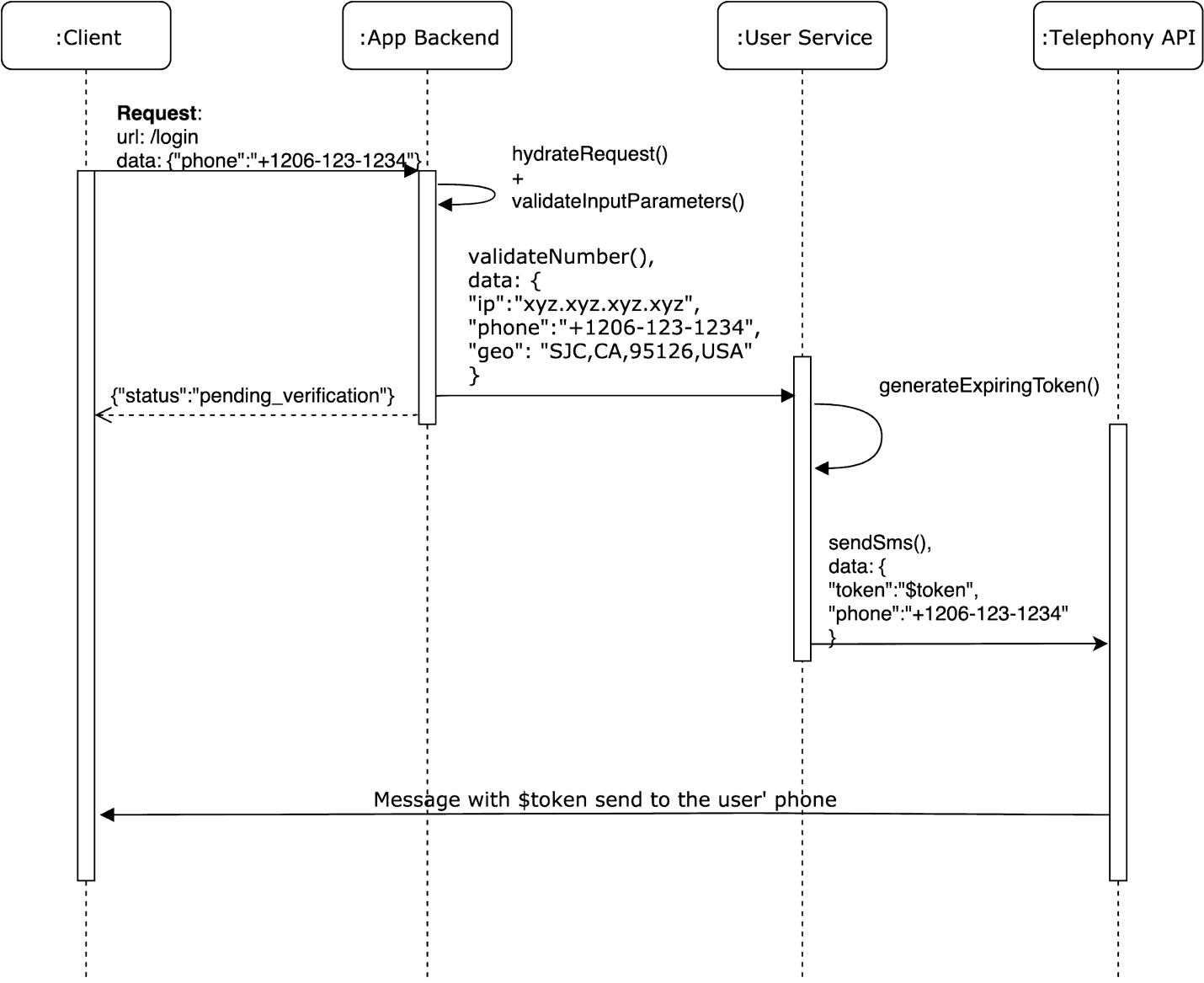
**sign up process is divided into two parts:**

1. Login Flow : In this phase, user submits the phone number to app backend and receives a token.
2. Verify Flow : In this phase, user submits the received token in the previous phase and sends to backend. On validation, user is logged in and a session cookie is issued.
3. **Login Flow:** In order to authenticate the user account, user submits the phone number to the App Backend . On submission, user is shown a screen where a token can be submitted. App backend validates the input parameter and enrichies/ hydrates the request with extra information like (IP address, geo information, device information etc). All this information can be used to generate a fingerprint of a user request which can be used for later security measures.This enriched and validated request is submitted to user service which generates a token and associate with these request. Token generation is an important process here.

**Token Generation**

The aim of the token is to make sure we can

* Expire the token so that same token cannot be used multiple times.
* Invalidate a token so that if a user enters the phone number multiple times then we can invalidate the previously generated token issue a new one.
* as the key and a randomly generated 6 digit token as the value which expires in 5 minutes.{“Key” : “+1206-123-1234”, “Value”:”123456”}



**Send SMS:** User service then calls a telephony API to send this token to the user’s phone number. There are various companies which provides this service (ex: Twilio, Plivo etc).

1. **Verify Phone Number Flow**

user enters the token on the form. This form consists of a hidden phone number field which is submitted to an endpoint (for ex: /verify\_phone) with the the token. Once again the app backend hydrates the request and validates the input parameters. The user service does a look up with the phone number as the key from the memcache data store. A session cookie is issued If the token exists and matches with the input token. This session cookie is sent as response header to the client which can persist it for later requests.

* **Sign up vs Login:** On successful token match, user service should do a look up in the user database to see if there is an already existing account with the same phone number, if that phone number is present then the user should be logged into that account. Another issue is to consider what happens if a user changes the phone number. This involves implementing a similar mechanism of verifying the new phone number and updating the user account if that is successful.
* **Advantage:** This definitely removes the mental load on the user to remember the email and passwords across various services. This is the best way.

**Signup using email:**

To sign up in any site we get more options like we can login with mail id, Phone number, Google, Face book, Amazon. I worked on Email.

Email was actually invented before the publicly accessible internet as we know it. An email address is unique identifier for an email account. It is used to both send and receive email messages over the internet. Similar to physical mail, an email message requires an address for both the sender and recipient in order to be sent successfully. Every email address has two main parts: a username and domain name. The username comes first, followed by an at (@) symbol, followed by the domain name.

When a message is sent SMTP the sending mail server checks for another mail server on the internet that corresponds with the domain name of the recipient address. If someone sends a message to user, it will check with the mail server to see if the username is valid. If the user exists, the message will be delivered.

While a basic email address consists of only a username and domain name, most email clients and webmail system include with names with email address. An email address contains a name is formatted with the first name, followed by the email address. While creating an email, it consists of Name, Birth date, Family name, Gender, Phone number and User name. Whenever we are signing up with email we have to enter all the details.

Birth Date: The user date of birth must be collected that should be in dd/mm/yyyy format.

Family Name: The complete family details is necessary and that to be stored.

Name: The complete name of the user is collected.

Gender: The user gender to be collected as male, female.

Phone number: The user’s phone number to be collected.

Email is based on these tools:

Qmail (http://www.qmail.org/), a robust mail transfer agent written by Dan Bernstein. It was designed to be simple, modular, secure, and reliable. It is easy to understand, and easy to integrate with other programs.

Oracle RDBMS (http://www.oracle.com).

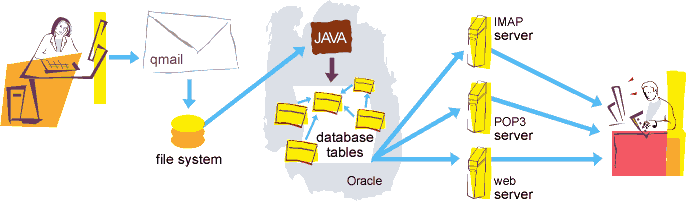
JavaSoft's JavaMail (http://java.sun.com/products/javamail/) and Oracle's SQLJ (http://www.oracle.com/java/index.html?jdbc\_sqlj.html) APIs, running within Oracle.

A presentation layer implemented in Tcl using AOLserver.

Qmail handles the traditional duties of a mail transfer agent: receipt of incoming email via SMTP, bouncing of incorrectly addressed or formatted messages, notification of errors during reception, and reliable delivery of the email into the file system. Aside from being configured in a somewhat idiosyncratic fashion, qmail is used without modification to do what it was designed to do. Non-interference with qmail means we can trust it to handle its MTA duties (to the extent that it is reliable).

After delivery to the file system, the message must be moved into the database as rows in tables. This job falls to Java code running in the database, relying on the JavaMail and SQLJ APIs to parse and insert the message. The headers are parsed and stored separately to facilitate searching and filtering, and any MIME attachments are decoded and stored separately. Once this processing is complete, the message is removed from the file system. This polling process is triggered once a minute by the DBMS\_JOB PL/SQL package.

Once the message is in the database, it is a simple matter to serve it up again through an IMAP server, or present it through Web pages produced by Tcl scripts running within AOL server. In the latter case, the flexibility of SQL make it easy to implement user interfaces that allow readers to view and manipulate their email in many powerful ways.



**Design Goals**

In writing email, I had the following goals:

Avoid writing code unnecessarily: Mail transfer agents are responsible for the reliable sending, receipt, and delivery of email in the face of network outages, full disks, and hardware failures. This is a solved problem; other people have solved it and solved it well, with MTAs that go to heroic lengths to preserve email. Similarly, parsing RFC822 and MIME messages correctly is something other people have written libraries for.

Make it easy for a human to handle large amounts of email:I receive more mail than most people. Much of it is from automated email alerts and monitors. I don't want to stop receiving them, but it is rare that I actually care to read them. Most web-based email interfaces seem to be designed with the notion that you actually care to read the email you receive. Webmail was designed to facilitate dealing with email in bulk.