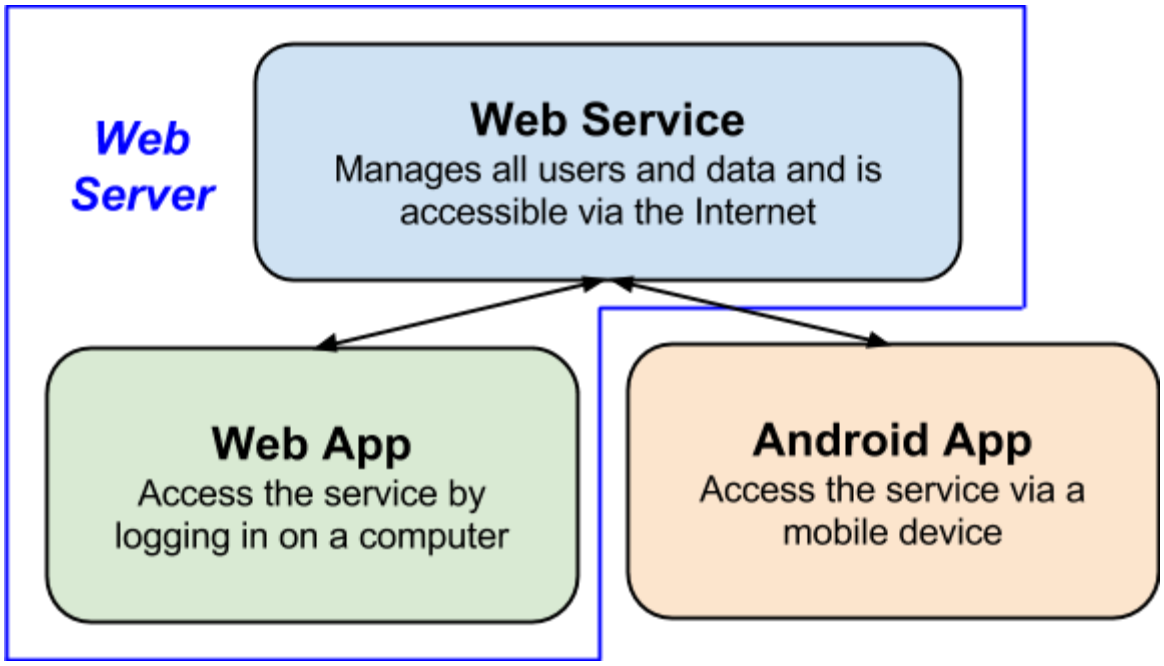
Incremental and Regression Testing

Classification of Components

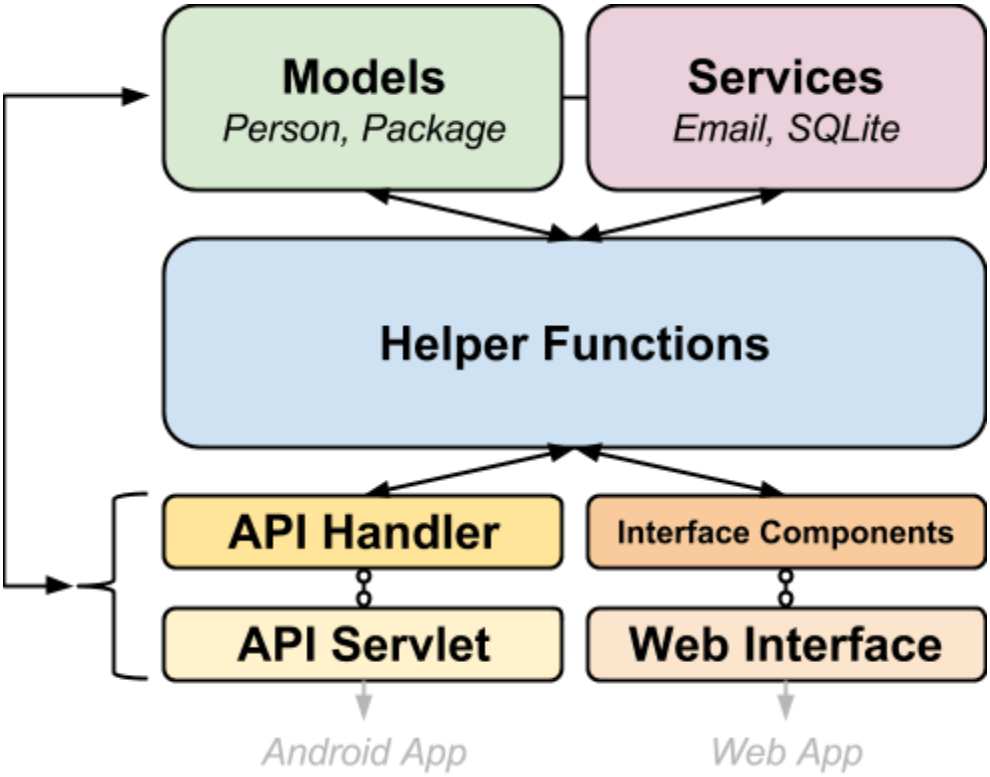
Overall Architecture and Major System Components



The overall system consists of three major components. A central web service is responsible for managing all of the user and package data as well as performing other main duties of the system. As part of this service, a web app allows users to access the functionality of the service by logging in via a web browser. Additionally, an API allows users to access the service by logging in via an Android app (or, potentially, other apps capable of using the API).

- Web Service
    - Input: Actions from the web app and messages from the API
    - Output: Data to display in web app and API response messages
    - Dependencies: See section below
    - Dependents: Web app and Android app
- Web App and Android App
    - Input: User commands via a GUI
    - Output: Displaying data and taking actions based on commands given
    - Dependencies: Vaadin Framework (for web interface)
    - Dependents: Users

Web Service and Web App (Web Server)



The web server provides the core services of the system as well as interfaces to access these services. From a user-facing perspective are two interfaces: a web-based interface for use in a web browser and an API for use by our Android app.

The API interface consists of two components. An API servlet reads requests from clients using the API and passes them to the API handler, which parses these messages, performs the appropriate tasks, and returns a response to the API servlet. This response is then returned to the client as the result of the client's request.

A web interface is also provided. The web interface is created in Java using the Vaadin web framework. Classes are created to generate a login UI, an account creation UI, a package management UI, and an administrator UI, all of which are bound to particular URLs in the web app. These UIs may consist of custom components, such as a login form.

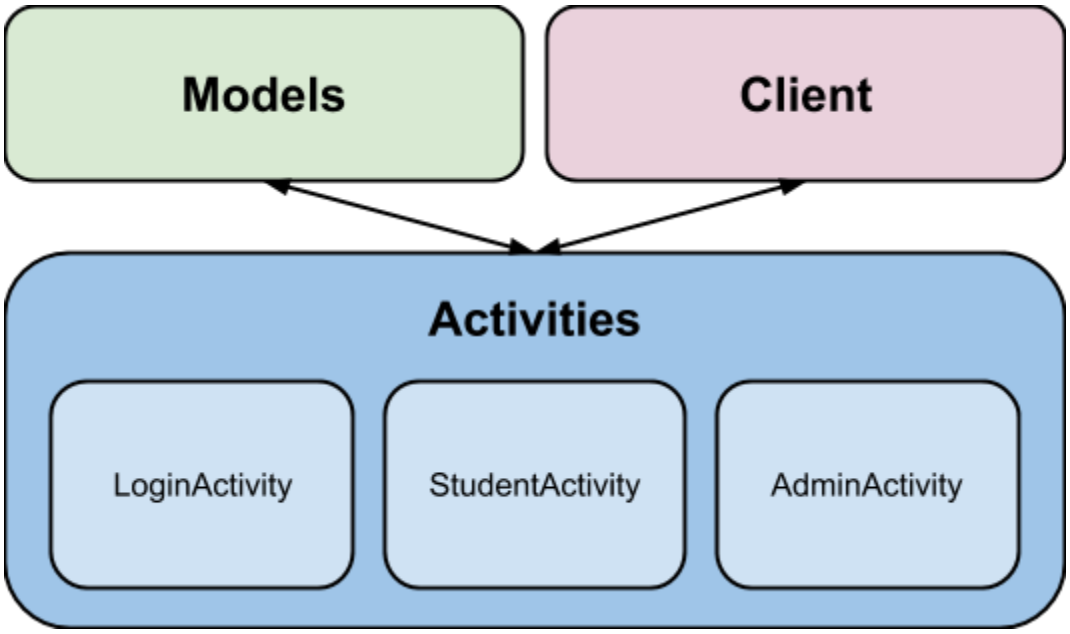
Helpers provide easy access to the core functionality of the system. Methods provide functionality to add users, check user credentials, add packages, change package status, and more. These methods perform appropriate database transactions as well as other functions such as sending emails. They help to ensure that consistent functionality is provided between the API and web app.

Finally, several services, such as an email service and SQLite helpers, are used by the helper functions in order to perform their tasks. Models for users and packages are used by the helpers, services, and other portions of this component of the system.

- API

    - Input: Command messages in a defined format
    - Output: Response messages in a defined format
    - Dependencies: Helper functions
    - Dependents: Android App
- Web Interface Input/Output: See overall architecture
- Helper Functions
    - Input: Method calls from API and web app classes
    - Output: Taking actions and returning instances of model classes to callers
    - Dependencies: Services, Models
    - Dependents: API and Web Interface
- Services
    - Input: Method calls from helper functions
    - Output: Performing actions and possibly returning data
    - Dependencies: Service providers (SQLite, MailX, etc.)
    - Dependents: Helper functions
- Models
    - Input: Construction of object
    - Output: Accessing properties
    - Dependencies: None
    - Dependents: Helper functions, API, web interface

Android App



The Android app consists of several activities, each allowing users to access and use different parts of the system. These include logging in, viewing packages, and managing packages as an administrator.

The activities make use of model classes for packages and users as well as a client class that handles communication with the server via HTTP.

- Activities
    - Input: User commands via GUI
    - Output: Updating interface with data based on commands
    - Dependencies: Models and HTTP client
    - Dependents: Users
- Models
    - Input: Construction of objects
    - Output: Accessing properties in activities
    - Dependencies: None
    - Dependents: Activities
- Client
    - Input: Method calls that cause HTTP requests to the server
    - Output: Response messages for activities resulting from requests
    - Dependencies: Apache HTTP Client library for Android
    - Dependents: Activities

Form of Incremental Testing Followed

Although we used both bottom-up and top-down testing methodologies, we primarily focused on top-down testing. This helped us ensure that the major components of our system (such as communication via the API) were working as soon as possible, while allowing us to focus on smaller details (such as the exact structure of models and the database) at a later time when they could be more easily tested as part of the system.

Incremental and Regression Testing

Incremental Testing

Android

Defect #	Description	Severity	How to Correct
1	Log in does not respond if client is not able to successfully communicate with web service	3	Add timeout to client and print an error Toast for user to see that timeout has occured
2	Log in crashed if given empty username or password to send to web service	2	When checking username and password check for empty length before attempting other operations
3	Valid logins that received bad messages from web service might not switch intents upon success	1	Make the default intent to assume that the user is a student and not an admin

Web Service

Defect #	Description	Severity	How to Correct
1	Client doesn't get a response from server	2	Ensure the Client always gets a message back no matter what happens in the server
2	Client doesn't get error notification from server that the login information is incorrect	2	Checking the sending username and password in database. If the mismatching happens, server will write error message to the Client.
3	If the Client sends garbage values to the server it could crash it	1	Check validity of message in Client before sending it

4	SQL injections could pose a security risk	1	Prepare statements and cleanse inputs

Web App

Defect #	Description	Severity	How to Correct
1	Web UI isn't responsive	2	When messages arrive or need to be sent to Java server do these tasks on a separate thread so other requests can be handled
2	SQL exceptions trigger improper error messages or pages in the client	2	Allow the checkLogin function to throw SQLException, and check for this when verifying logins in order to return a useful error message

Regression Testing

Android

Defect #	Description	Severity	How to Correct
1	Android Application should not crash when given an empty input field	2	Check size of Strings pulled from Android forms before performing other String operations on them
2	Android application would create a new account if the password field is empty	3	Throw an exception if the user attempts to use an empty username with empty password
3	Android application would not switch intents if the client class returned an ambiguous success	1	Make a default intent switch for any success message returned

Web Server

Defect #	Description	Severity	How to Correct
1	Malicious users could specify names to mimic certain commands	2	Prepend a constant String to each message to prevent user from being able to call any server methods by specifying the name to look like a different message
2	Handling requests on separate threads created race conditions	1	Use calls to database functions which has the effect of making any code that changes the database atomic
3	Adding Client timeout didn’t allow server time to perform longer operations	3	Increased timeout amount to give server more time

Web App

Defect #	Description	Severity	How to Correct
1	Registering the new user account with an existing username	2	Before adding the new entry to the database, check whether the username has already been stored
2	Failing to jump to a new webpage after successfully log in	1	Returning error message to the user. Request the server to set up a timer and resend the message
3	Failing to load the package tracking number or update the status on the webpage	1	Leave a reasonable amount of time to the server for checking and updating the data