Testing

Unit Testing

Database

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Test case | Expected Results | Actual Results | Notes |
| 1 | parsePayload([]byte) | The byte array parsed using |  |  |
| 2 | IsDeviceStolen(string) |  |  |  |
| 3 | VerifyAccountInfo(string, string) |  |  |  |
| 4 | GetUserDevice(string) |  |  |  |

Blackbox testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Action | Expected Result | Actual Result | Notes |
| 1 | User enters URL | Home page | As expected | URL: localhost:8080/home |
| 2 | User fills Sign Up form and submit | New account and customer created on database | As expected |  |
| 3 | User access control: before log-in | Error page: not logged in | As expected |  |
| 4 | User fills login form and submit | Rerouted to user map page | As expected |  |
| 5 | User map receives updates | Markers on map updating | As expected |  |
| 6 | User logs out | User redirected to home page and logged out of session | As expected |  |

Testing Screen Captures



Password: hello

Figure : Test #2 form filled before submit



Figure 1.1: Test #2 customer table after sign up submitted



Hashed password

Relation

Figure 1.2: Test #2 account table after sign up submitted



Figure : Test #3 Error page



Figure .1: Test #4 login form before submit



Figure 3.2: Test #4 login after submit, redirected to map page

Go server

3-tier architecture – Server, database, and client (Web)

* Server components –
  + Central Server
    - The central server is the entry point of the server. It initializes and starts the modules of the server, and initializes the communication channels used by the modules.
    - The central server connects all of the server modules together. All components send and received Requests through the central server. When a request is received, the central server redirects the request to the proper module.
  + RequestProtocol
    - The modules of the server communicate with a requesting protocol. Using a requesting protocol allows the server to be organized. It also allows for adding and removing server modules with less effort, because it allows each module to have a high cohesion.
    - The protocol contains opcodes for modules to use when creating requests.
    - A requests contains an id, the destination of the request, the source of the request, an opcode, a payload, and a response channel that the response to the request will be sent over.
  + Web Server
    - Http handler – handles http requests received and sends back an http response to the request. The response will contain all of the files needed for the client view. A typical response would contain files such as images, style sheets, scripts, and the html file.
    - Websocket handler – handles websocket requests made to the server. When a websocket request is received, the handler creates a new connection to the client and registers the connection in the Web hub.
    - WebClientConnection – Middleman between websocket and the web hub. Reads in messages from the websocket and pass it on to the hub. Also receives messages from the hub and passes it to the websocket.
    - Web hub – Maintains a list of connected clients as WebClientConnections; creates a channel for communication to and from those clients
    - Client Web Sessions – when a web client makes an http request to the server, a cookie-based session is created. A copy of the session is stored in the server. When a client logs in, the contents of the cookie are changed and represent a session for the client. The sessions are used for access control and to present the clients with relevant data.
    - RequestProtocol handler – handles incoming and outgoing requests made from the other server components.