

# Open-Source Report

Proof of knowing your stuff in CSE312

## Guidelines

Provided below is a template you must use to write your reports for your project.

Here are some things to note when working on your report, specifically about the **General Information & Licensing** section for each technology.

- **Code Repository:** Please link the code and not the documentation. If you'd like to refer to the documentation in the **Magic** section, you're more than welcome to, but we need to see the code you're referring to as well.
- **License Type:** Three letter acronym is fine.
- **License Description:** No need for the entire license here, just what separates it from the rest.
- **License Restrictions:** What can you *not* do as a result of using this technology in your project? Some licenses prevent you from using the project for commercial use, for example.

Also, feel free to extend the cell of any section if you feel you need more room.

If there's anything we can clarify, please don't hesitate to reach out! You can reach us using the methods outlined on the course website or see us during our office hours.

## Express.js - Parsing HTTP headers

### General Information & Licensing

Code Repository	<a href="https://github.com/expressjs/express">https://github.com/expressjs/express</a>
License Type	MIT License
License Description	<a href="https://github.com/expressjs/express/blob/master/LICENSE">https://github.com/expressjs/express/blob/master/LICENSE</a>  (The MIT License)  Copyright (c) 2009-2014 TJ Holowaychuk <tj@vision-media.ca>

	<p>Copyright (c) 2013-2014 Roman Shtylman &lt;shtylman+expressjs@gmail.com&gt; Copyright (c) 2014-2015 Douglas Christopher Wilson &lt;doug@somethingdoug.com&gt;</p> <p>Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the 'Software'), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:</p> <p>The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.</p> <p>THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.</p>
License Restrictions	<ul style="list-style-type: none"><li>• No Warranty</li><li>• No Liability</li></ul>

# Purpose

Express.js handles our HTTP requests and sends our responses. It is a layer on top of Node.js that is efficient in managing servers and routes. The express application is exported from the express module and handles top-level functions.

In our use, it can determine what HTTP requests were made, help us parse the headers, retrieve the information needed, and send our HTTP response to the client.

This functionality was used in server.js where:

- Express was imported [\[Line 1\]](#):
  - Syntax → **`const express = require('express')`**
- App object was created using `express()` in "auction.js" - a routing path file of the backend, where it is then exported to the main source file of the backend (server.js) to be the core/main usage of the Express framework, which is to let us add middleware and functionality to our server API.
  - The App object is a JavaScript Function designed to be passed to Node's HTTP Servers as a callback to handle request.
  - App object usage can be found on lines:
    - server.js → Lines [12-14](#), [26-28](#), [31-32](#), [36](#)
    - auction.js → Lines [12](#)
- Using **`app.use()`** we were able to put specified middleware\*\* functions at a specific path, organizing and simplifying our routings. (It differs from **`app.get()`** by being able to handle more than just **GET** requests of a specific path/route, such as **POST, PUT, DELETE, etc.**)
  - \*\*Definition of middleware → functions that run between a client request and response from server, have access to request object (req), response object (res), and next function.
  - General format of **`app.use()`** → **`app.use([path], callback, [callback])`**
  - Usage on Lines [12-14](#) of source file for backend (server.js):
    - Handled parsing the body of the HTTP request and cookies.
    - It then returns a function which could be passed into `app.use` as another function.
  - Usage on Lines [26-28](#) of source file for backend (server.js):
    - Handled routes of products, users, auction, purchasing, and selling.
    - **`app.use()`** lets us create route specific middleware, meaning that for a specific HTTP request, we are attaching a stack of operations uniquely for a request, such as a path to user accounts, bidding, etc.
- **`App.get()`** is a method of the Express app, which is used to handle HTTP GET requests that the server receives, which uses this function to determine what to do

when a get request at a given route is called.

- General format of **app.get()** → **app.get([path], callback)**
- Usage of this method can be found on [Line 36](#), which specifies the root URL of the server.
  - Serves the response status code **200**.
  - Sends the raw file to the root directory, in our example it would be `'index.html'`.

*Magic* ★★°°°° ☾ ◡ ↗ ° ★ ≡ ✨ 🌸

The Express.js library prepares and sends the given HTTP status code on request header in the function defined on [Line-369](#). It uses the [statuses](#) module to get the status codes and corresponding messages. The map of all the code and messages are defined in <https://github.com/jshttp/statuses/blob/master/src/node.json> and furthermore defined in the [node](#) library at: [https://github.com/nodejs/node/blob/main/lib/\\_http\\_server.js#L106](https://github.com/nodejs/node/blob/main/lib/_http_server.js#L106).

- The logic to check if the incoming request contains the "Content-Type" header field is defined in [Line-618](#), and it contains the given mime `type` ([Line-278](#)).
- The logic to set the headers is defined on [Line-579](#).
  - This method is used by the server in generating a response for the client. This method follows a similar approach to generating a response in **Homework 2**. For example, [Line 789](#), is shown to split the http header with (";").
- The logic to set the cookies is defined on [Line-831](#) and to clear the cookies is on [Line-824](#).
  - This method is used by the server to set a cookie to the client similar to that of **Homework 3**. For example ([Line 884](#)) appends to the response with a `'Set-Cookies'` header with the given parameters being a key/value pair.
- **app.get()** → The **req** object is inherited from (from the prototype of) **http.IncomingMessage** class ([Line-31](#)), which is defined in the **http** module. In the http library (http module for JS is not available on GitHub anymore, so we are attaching screenshots from the node modules), we have the class defined on **Line-839**.

```

832 * status, headers and data.
833 *
834 * Different from its 'socket' value which is a subclass of 'stream.Duplex', the 'IncomingMessage' is
835 * parse and emit the incoming HTTP headers and payload, as the underlying socket
836 * may be reused multiple times in case of keep-alive.
837 * @since v0.1.17
838 */
839 class IncomingMessage extends stream.Readable {
840   constructor(socket: Socket);
841   /**
842    * The 'message.aborted' property will be 'true' if the request has
843    * been aborted.
844    * @since v18.1.0
845    * @deprecated Since v17.0.0, v16.12.0 - Check 'message.destroyed' from <a href="stream.html#class-readable-stream-2">stream.Readable</a>
846    */
847   aborted: boolean;
848   /**
849    * In case of server request, the HTTP version sent by the client. In the case of
850    * client response, the HTTP version of the connected-to server.
851    * Probably either '1.1' or '1.0'.
852    *
853    * Also 'message.httpVersionMajor' is the first integer and 'message.httpVersionMinor' is the second integer.
854    * @since v0.1.1
855    */
856   httpVersion: string;
857   httpVersionMajor: number;
858   httpVersionMinor: number;
859   /**
860    * The 'message.complete' property will be 'true' if a complete HTTP message has
861    * been received and successfully parsed.
862    *
863    * This property is particularly useful as a means of determining if a client or
864    * server fully transmitted a message before a connection was terminated.
865    *
866    * ''js
867    * const req = http.request({
868    *   host: '127.0.0.1',
869    *
870    *   // ...
871    * });
872    *
873    * req.on('end', () => {
874    *   // message was received
875    * });
876    */
877   complete: boolean;
878   /**
879    * The raw request headers.
880    *
881    * Note that the keys of this object are the lowercased header names.
882    *
883    * ''js
884    * const req = http.request({
885    *   host: '127.0.0.1',
886    *   headers: {
887     *     'X-Custom-Header': 'tobi',
888     *     'user-agent': 'node-http'
889     *   },
890    *   // ...
891    * });
892    *
893    * req.on('end', () => {
894    *   // req.headers is {
895    *   //   'x-custom-header': 'tobi',
896    *   //   'user-agent': 'node-http',
897    *   // }
898    * });
899    */
900   headers: string | string[];
901   /**
902    * The request method.
903    *
904    * Note that for HTTP 200, the request method is lowercased.
905    *
906    * ''js
907    * const req = http.request({
908    *   host: '127.0.0.1',
909    *   method: 'GET',
910    *   // ...
911    * });
912    *
913    * req.on('end', () => {
914    *   // req.method is 'get'
915    * });
916    */
917   method: string;
918   /**
919    * The raw request headers as an array of key-value pairs.
920    *
921    * Note that the keys of this object are the lowercased header names.
922    *
923    * ''js
924    * const req = http.request({
925    *   host: '127.0.0.1',
926    *   headers: {
927     *     'X-Custom-Header': 'tobi',
928     *     'user-agent': 'node-http'
929     *   },
930    *   // ...
931    * });
932    *
933    * req.on('end', () => {
934    *   // req.rawHeaders is ['x-custom-header', 'tobi', 'user-agent', 'node-http']
935    * });
936    */
937   rawHeaders: string[];
938   /**
939    * The raw request trailers as an array of key-value pairs.
940    *
941    * Note that the keys of this object are the lowercased header names.
942    *
943    * ''js
944    * const req = http.request({
945    *   host: '127.0.0.1',
946    *   method: 'GET',
947    *   trailers: {
948     *     'x-custom-trailer': 'tobi',
949     *     'content-length': '10'
950     *   },
951    *   // ...
952    * });
953    *
954    * req.on('end', () => {
955    *   // req.rawTrailers is ['x-custom-trailer', 'tobi', 'content-length', '10']
956    * });
957    */
958   rawTrailers: string[];
959   /**
960    * Set to true when the request headers have been received and the request body
961    * is being read.
962    *
963    * Note that for HTTP 200, the request method is lowercased.
964    *
965    * ''js
966    * const req = http.request({
967    *   host: '127.0.0.1',
968    *   method: 'GET',
969    *   // ...
970    * });
971    *
972    * req.on('end', () => {
973    *   // req.isReadable is true
974    * });
975    */
976   isReadable: boolean;
977   /**
978    * Set to true when the request headers have been received and the request body
979    * is being read.
980    *
981    * Note that for HTTP 200, the request method is lowercased.
982    *
983    * ''js
984    * const req = http.request({
985    *   host: '127.0.0.1',
986    *   method: 'GET',
987    *   // ...
988    * });
989    *
990    * req.on('end', () => {
991    *   // req.isReadable is true
992    * });
993    */
994   isReadable: boolean;
995   /**
996    * Set to true when the request headers have been received and the request body
997    * is being read.
998    *
999    * Note that for HTTP 200, the request method is lowercased.
1000    *
1001    * ''js
1002    * const req = http.request({
1003    *   host: '127.0.0.1',
1004    *   method: 'GET',
1005    *   // ...
1006    * });
1007    *
1008    * req.on('end', () => {
1009    *   // req.isReadable is true
1010    * });
1011    */
1012   isReadable: boolean;
1013   /**
1014    * Set to true when the request headers have been received and the request body
1015    * is being read.
1016    *
1017    * Note that for HTTP 200, the request method is lowercased.
1018    *
1019    * ''js
1020    * const req = http.request({
1021    *   host: '127.0.0.1',
1022    *   method: 'GET',
1023    *   // ...
1024    * });
1025    *
1026    * req.on('end', () => {
1027    *   // req.isReadable is true
1028    * });
1029    */
1030   isReadable: boolean;
1031   /**
1032    * Set to true when the request headers have been received and the request body
1033    * is being read.
1034    *
1035    * Note that for HTTP 200, the request method is lowercased.
1036    *
1037    * ''js
1038    * const req = http.request({
1039    *   host: '127.0.0.1',
1040    *   method: 'GET',
1041    *   // ...
1042    * });
1043    *
1044    * req.on('end', () => {
1045    *   // req.isReadable is true
1046    * });
1047    */
1048   isReadable: boolean;
1049   /**
1050    * Set to true when the request headers have been received and the request body
1051    * is being read.
1052    *
1053    * Note that for HTTP 200, the request method is lowercased.
1054    *
1055    * ''js
1056    * const req = http.request({
1057    *   host: '127.0.0.1',
1058    *   method: 'GET',
1059    *   // ...
1060    * });
1061    *
1062    * req.on('end', () => {
1063    *   // req.isReadable is true
1064    * });
1065    */
1066   isReadable: boolean;
1067   /**
1068    * Set to true when the request headers have been received and the request body
1069    * is being read.
1070    *
1071    * Note that for HTTP 200, the request method is lowercased.
1072    *
1073    * ''js
1074    * const req = http.request({
1075    *   host: '127.0.0.1',
1076    *   method: 'GET',
1077    *   // ...
1078    * });
1079    *
1080    * req.on('end', () => {
1081    *   // req.isReadable is true
1082    * });
1083    */
1084   isReadable: boolean;
1085   /**
1086    * Set to true when the request headers have been received and the request body
1087    * is being read.
1088    *
1089    * Note that for HTTP 200, the request method is lowercased.
1090    *
1091    * ''js
1092    * const req = http.request({
1093    *   host: '127.0.0.1',
1094    *   method: 'GET',
1095    *   // ...
1096    * });
1097    *
1098    * req.on('end', () => {
1099    *   // req.isReadable is true
1100    * });
1101    */
1102   isReadable: boolean;
1103   /**
1104    * Set to true when the request headers have been received and the request body
1105    * is being read.
1106    *
1107    * Note that for HTTP 200, the request method is lowercased.
1108    *
1109    * ''js
1110    * const req = http.request({
1111    *   host: '127.0.0.1',
1112    *   method: 'GET',
1113    *   // ...
1114    * });
1115    *
1116    * req.on('end', () => {
1117    *   // req.isReadable is true
1118    * });
1119    */
1120   isReadable: boolean;
1121   /**
1122    * Set to true when the request headers have been received and the request body
1123    * is being read.
1124    *
1125    * Note that for HTTP 200, the request method is lowercased.
1126    *
1127    * ''js
1128    * const req = http.request({
1129    *   host: '127.0.0.1',
1130    *   method: 'GET',
1131    *   // ...
1132    * });
1133    *
1134    * req.on('end', () => {
1135    *   // req.isReadable is true
1136    * });
1137    */
1138   isReadable: boolean;
1139   /**
1140    * Set to true when the request headers have been received and the request body
1141    * is being read.
1142    *
1143    * Note that for HTTP 200, the request method is lowercased.
1144    *
1145    * ''js
1146    * const req = http.request({
1147    *   host: '127.0.0.1',
1148    *   method: 'GET',
1149    *   // ...
1150    * });
1151    *
1152    * req.on('end', () => {
1153    *   // req.isReadable is true
1154    * });
1155    */
1156   isReadable: boolean;
1157   /**
1158    * Set to true when the request headers have been received and the request body
1159    * is being read.
1160    *
1161    * Note that for HTTP 200, the request method is lowercased.
1162    *
1163    * ''js
1164    * const req = http.request({
1165    *   host: '127.0.0.1',
1166    *   method: 'GET',
1167    *   // ...
1168    * });
1169    *
1170    * req.on('end', () => {
1171    *   // req.isReadable is true
1172    * });
1173    */
1174   isReadable: boolean;
1175   /**
1176    * Set to true when the request headers have been received and the request body
1177    * is being read.
1178    *
1179    * Note that for HTTP 200, the request method is lowercased.
1180    *
1181    * ''js
1182    * const req = http.request({
1183    *   host: '127.0.0.1',
1184    *   method: 'GET',
1185    *   // ...
1186    * });
1187    *
1188    * req.on('end', () => {
1189    *   // req.isReadable is true
1190    * });
1191    */
1192   isReadable: boolean;
1193   /**
1194    * Set to true when the request headers have been received and the request body
1195    * is being read.
1196    *
1197    * Note that for HTTP 200, the request method is lowercased.
1198    *
1199    * ''js
1200    * const req = http.request({
1201    *   host: '127.0.0.1',
1202    *   method: 'GET',
1203    *   // ...
1204    * });
1205    *
1206    * req.on('end', () => {
1207    *   // req.isReadable is true
1208    * });
1209    */
1210   isReadable: boolean;
1211   /**
1212    * Set to true when the request headers have been received and the request body
1213    * is being read.
1214    *
1215    * Note that for HTTP 200, the request method is lowercased.
1216    *
1217    * ''js
1218    * const req = http.request({
1219    *   host: '127.0.0.1',
1220    *   method: 'GET',
1221    *   // ...
1222    * });
1223    *
1224    * req.on('end', () => {
1225    *   // req.isReadable is true
1226    * });
1227    */
1228   isReadable: boolean;
1229   /**
1230    * Set to true when the request headers have been received and the request body
1231    * is being read.
1232    *
1233    * Note that for HTTP 200, the request method is lowercased.
1234    *
1235    * ''js
1236    * const req = http.request({
1237    *   host: '127.0.0.1',
1238    *   method: 'GET',
1239    *   // ...
1240    * });
1241    *
1242    * req.on('end', () => {
1243    *   // req.isReadable is true
1244    * });
1245    */
1246   isReadable: boolean;
1247   /**
1248    * Set to true when the request headers have been received and the request body
1249    * is being read.
1250    *
1251    * Note that for HTTP 200, the request method is lowercased.
1252    *
1253    * ''js
1254    * const req = http.request({
1255    *   host: '127.0.0.1',
1256    *   method: 'GET',
1257    *   // ...
1258    * });
1259    *
1260    * req.on('end', () => {
1261    *   // req.isReadable is true
1262    * });
1263    */
1264   isReadable: boolean;
1265   /**
1266    * Set to true when the request headers have been received and the request body
1267    * is being read.
1268    *
1269    * Note that for HTTP 200, the request method is lowercased.
1270    *
1271    * ''js
1272    * const req = http.request({
1273    *   host: '127.0.0.1',
1274    *   method: 'GET',
1275    *   // ...
1276    * });
1277    *
1278    * req.on('end', () => {
1279    *   // req.isReadable is true
1280    * });
1281    */
1282   isReadable: boolean;
1283   /**
1284    * Set to true when the request headers have been received and the request body
1285    * is being read.
1286    *
1287    * Note that for HTTP 200, the request method is lowercased.
1288    *
1289    * ''js
1290    * const req = http.request({
1291    *   host: '127.0.0.1',
1292    *   method: 'GET',
1293    *   // ...
1294    * });
1295    *
1296    * req.on('end', () => {
1297    *   // req.isReadable is true
1298    * });
1299    */
1300   isReadable: boolean;
1301   /**
1302    * Set to true when the request headers have been received and the request body
1303    * is being read.
1304    *
1305    * Note that for HTTP 200, the request method is lowercased.
1306    *
1307    * ''js
1308    * const req = http.request({
1309    *   host: '127.0.0.1',
1310    *   method: 'GET',
1311    *   // ...
1312    * });
1313    *
1314    * req.on('end', () => {
1315    *   // req.isReadable is true
1316    * });
1317    */
1318   isReadable: boolean;
1319   /**
1320    * Set to true when the request headers have been received and the request body
1321    * is being read.
1322    *
1323    * Note that for HTTP 200, the request method is lowercased.
1324    *
1325    * ''js
1326    * const req = http.request({
1327    *   host: '127.0.0.1',
1328    *   method: 'GET',
1329    *   // ...
1330    * });
1331    *
1332    * req.on('end', () => {
1333    *   // req.isReadable is true
1334    * });
1335    */
1336   isReadable: boolean;
1337   /**
1338    * Set to true when the request headers have been received and the request body
1339    * is being read.
1340    *
1341    * Note that for HTTP 200, the request method is lowercased.
1342    *
1343    * ''js
1344    * const req = http.request({
1345    *   host: '127.0.0.1',
1346    *   method: 'GET',
1347    *   // ...
1348    * });
1349    *
1350    * req.on('end', () => {
1351    *   // req.isReadable is true
1352    * });
1353    */
1354   isReadable: boolean;
1355   /**
1356    * Set to true when the request headers have been received and the request body
1357    * is being read.
1358    *
1359    * Note that for HTTP 200, the request method is lowercased.
1360    *
1361    * ''js
1362    * const req = http.request({
1363    *   host: '127.0.0.1',
1364    *   method: 'GET',
1365    *   // ...
1366    * });
1367    *
1368    * req.on('end', () => {
1369    *   // req.isReadable is true
1370    * });
1371    */
1372   isReadable: boolean;
1373   /**
1374    * Set to true when the request headers have been received and the request body
1375    * is being read.
1376    *
1377    * Note that for HTTP 200, the request method is lowercased.
1378    *
1379    * ''js
1380    * const req = http.request({
1381    *   host: '127.0.0.1',
1382    *   method: 'GET',
1383    *   // ...
1384    * });
1385    *
1386    * req.on('end', () => {
1387    *   // req.isReadable is true
1388    * });
1389    */
1390   isReadable: boolean;
1391   /**
1392    * Set to true when the request headers have been received and the request body
1393    * is being read.
1394    *
1395    * Note that for HTTP 200, the request method is lowercased.
1396    *
1397    * ''js
1398    * const req = http.request({
1399    *   host: '127.0.0.1',
1400    *   method: 'GET',
1401    *   // ...
1402    * });
1403    *
1404    * req.on('end', () => {
1405    *   // req.isReadable is true
1406    * });
1407    */
1408   isReadable: boolean;
1409   /**
1410    * Set to true when the request headers have been received and the request body
1411    * is being read.
1412    *
1413    * Note that for HTTP 200, the request method is lowercased.
1414    *
1415    * ''js
1416    * const req = http.request({
1417    *   host: '127.0.0.1',
1418    *   method: 'GET',
1419    *   // ...
1420    * });
1421    *
1422    * req.on('end', () => {
1423    *   // req.isReadable is true
1424    * });
1425    */
1426   isReadable: boolean;
1427   /**
1428    * Set to true when the request headers have been received and the request body
1429    * is being read.
1430    *
1431    * Note that for HTTP 200, the request method is lowercased.
1432    *
1433    * ''js
1434    * const req = http.request({
1435    *   host: '127.0.0.1',
1436    *   method: 'GET',
1437    *   // ...
1438    * });
1439    *
1440    * req.on('end', () => {
1441    *   // req.isReadable is true
1442    * });
1443    */
1444   isReadable: boolean;
1445   /**
1446    * Set to true when the request headers have been received and the request body
1447    * is being read.
1448    *
1449    * Note that for HTTP 200, the request method is lowercased.
1450    *
1451    * ''js
1452    * const req = http.request({
1453    *   host: '127.0.0.1',
1454    *   method: 'GET',
1455    *   // ...
1456    * });
1457    *
1458    * req.on('end', () => {
1459    *   // req.isReadable is true
1460    * });
1461    */
1462   isReadable: boolean;
1463   /**
1464    * Set to true when the request headers have been received and the request body
1465    * is being read.
1466    *
1467    * Note that for HTTP 200, the request method is lowercased.
1468    *
1469    * ''js
1470    * const req = http.request({
1471    *   host: '127.0.0.1',
1472    *   method: 'GET',
1473    *   // ...
1474    * });
1475    *
1476    * req.on('end', () => {
1477    *   // req.isReadable is true
1478    * });
1479    */
1480   isReadable: boolean;
1481   /**
1482    * Set to true when the request headers have been received and the request body
1483    * is being read.
1484    *
1485    * Note that for HTTP 200, the request method is lowercased.
1486    *
1487    * ''js
1488    * const req = http.request({
1489    *   host: '127.0.0.1',
1490    *   method: 'GET',
1491    *   // ...
1492    * });
1493    *
1494    * req.on('end', () => {
1495    *   // req.isReadable is true
1496    * });
1497    */
1498   isReadable: boolean;
1499   /**
1500    * Set to true when the request headers have been received and the request body
1501    * is being read.
1502    *
1503    * Note that for HTTP 200, the request method is lowercased.
1504    *
1505    * ''js
1506    * const req = http.request({
1507    *   host: '127.0.0.1',
1508    *   method: 'GET',
1509    *   // ...
1510    * });
1511    *
1512    * req.on('end', () => {
1513    *   // req.isReadable is true
1514    * });
1515    */
1516   isReadable: boolean;
1517   /**
1518    * Set to true when the request headers have been received and the request body
1519    * is being read.
1520    *
1521    * Note that for HTTP 200, the request method is lowercased.
1522    *
1523    * ''js
1524    * const req = http.request({
1525    *   host: '127.0.0.1',
1526    *   method: 'GET',
1527    *   // ...
1528    * });
1529    *
1530    * req.on('end', () => {
1531    *   // req.isReadable is true
1532    * });
1533    */
1534   isReadable: boolean;
1535   /**
1536    * Set to true when the request headers have been received and the request body
1537    * is being read.
1538    *
1539    * Note that for HTTP 200, the request method is lowercased.
1540    *
1541    * ''js
1542    * const req = http.request({
1543    *   host: '127.0.0.1',
1544    *   method: 'GET',
1545    *   // ...
1546    * });
1547    *
1548    * req.on('end', () => {
1549    *   // req.isReadable is true
1550    * });
1551    */
1552   isReadable: boolean;
1553   /**
1554    * Set to true when the request headers have been received and the request body
1555    * is being read.
1556    *
1557    * Note that for HTTP 200, the request method is lowercased.
1558    *
1559    * ''js
1560    * const req = http.request({
1561    *   host: '127.0.0.1',
1562    *   method: 'GET',
1563    *   // ...
1564    * });
1565    *
1566    * req.on('end', () => {
1567    *   // req.isReadable is true
1568    * });
1569    */
1570   isReadable: boolean;
1571   /**
1572    * Set to true when the request headers have been received and the request body
1573    * is being read.
1574    *
1575    * Note that for HTTP 200, the request method is lowercased.
1576    *
1577    * ''js
1578    * const req = http.request({
1579    *   host: '127.0.0.1',
1580    *   method: 'GET',
1581    *   // ...
1582    * });
1583    *
1584    * req.on('end', () => {
1585    *   // req.isReadable is true
1586    * });
1587    */
1588   isReadable: boolean;
1589   /**
1590    * Set to true when the request headers have been received and the request body
1591    * is being read.
1592    *
1593    * Note that for HTTP 200, the request method is lowercased.
1594    *
1595    * ''js
1596    * const req = http.request({
1597    *   host: '127.0.0.1',
1598    *   method: 'GET',
1599    *   // ...
1600    * });
1601    *
1602    * req.on('end', () => {
1603    *   // req.isReadable is true
1604    * });
1605    */
1606   isReadable: boolean;
1607   /**
1608    * Set to true when the request headers have been received and the request body
1609    * is being read.
1610    *
1611    * Note that for HTTP 200, the request method is lowercased.
1612    *
1613    * ''js
1614    * const req = http.request({
1615    *   host: '127.0.0.1',
1616    *   method: 'GET',
1617    *   // ...
1618    * });
1619    *
1620    * req.on('end', () => {
1621    *   // req.isReadable is true
1622    * });
1623    */
1624   isReadable: boolean;
1625   /**
1626    * Set to true when the request headers have been received and the request body
1627    * is being read.
1628    *
1629    * Note that for HTTP 200, the request method is lowercased.
1630    *
1631    * ''js
1632    * const req = http.request({
1633    *   host: '127.0.0.1',
1634    *   method: 'GET',
1635    *   // ...
1636    * });
1637    *
1638    * req.on('end', () => {
1639    *   // req.isReadable is true
1640    * });
1641    */
1642   isReadable: boolean;
1643   /**
1644    * Set to true when the request headers have been received and the request body
1645    * is being read.
1646    *
1647    * Note that for HTTP 200, the request method is lowercased.
1648    *
1649    * ''js
1650    * const req = http.request({
1651    *   host: '127.0.0.1',
1652    *   method: 'GET',
1653    *   // ...
1654    * });
1655    *
1656    * req.on('end', () => {
1657    *   // req.isReadable is true
1658    * });
1659    */
1660   isReadable: boolean;
1661   /**
1662    * Set to true when the request headers have been received and the request body
1663    * is being read.
1664    *
1665    * Note that for HTTP 200, the request method is lowercased.
1666    *
1667    * ''js
1668    * const req = http.request({
1669    *   host: '127.0.0.1',
1670    *   method: 'GET',
1671    *   // ...
1672    * });
1673    *
1674    * req.on('end', () => {
1675    *   // req.isReadable is true
1676    * });
1677    */
1678   isReadable: boolean;
1679   /**
1680    * Set to true when the request headers have been received and the request body
1681    * is being read.
1682    *
1683    * Note that for HTTP 200, the request method is lowercased.
1684    *
1685    * ''js
1686    * const req = http.request({
1687    *   host: '127.0.0.1',
1688    *   method: 'GET',
1689    *   // ...
1690    * });
1691    *
1692    * req.on('end', () => {
1693    *   // req.isReadable is true
1694    * });
1695    */
1696   isReadable: boolean;
1697   /**
1698    * Set to true when the request headers have been received and the request body
1699    * is being read.
1700    *
1701    * Note that for HTTP 200, the request method is lowercased.
1702    *
1703    * ''js
1704    * const req = http.request({
1705    *   host: '127.0.0.1',
1706    *   method: 'GET',
1707    *   // ...
1708    * });
1709    *
1710    * req.on('end', () => {
1711    *   // req.isReadable is true
1712    * });
1713    */
1714   isReadable: boolean;
1715   /**
1716    * Set to true when the request headers have been received and the request body
1717    * is being read.
1718    *
1719    * Note that for HTTP 200, the request method is lowercased.
1720    *
1721    * ''js
1722    * const req = http.request({
1723    *   host: '127.0.0.1',
1724    *   method: 'GET',
1725    *   // ...
1726    * });
1727    *
1728    * req.on('end', () => {
1729    *   // req.isReadable is true
1730    * });
1731    */
1732   isReadable: boolean;
1733   /**
1734    * Set to true when the request headers have been received and the request body
1735    * is being read.
1736    *
1737    * Note that for HTTP 200, the request method is lowercased.
1738    *
1739    * ''js
1740    * const req = http.request({
1741    *   host: '127.0.0.1',
1742    *   method: 'GET',
1743    *   // ...
1744    * });
1745    *
1746    * req.on('end', () => {
1747    *   // req.isReadable is true
1748    * });
1749    */
1750   isReadable: boolean;
1751   /**
1752    * Set to true when the request headers have been received and the request body
1753    * is being read.
1754    *
1755    * Note that for HTTP 200, the request method is lowercased.
1756    *
1757    * ''js
1758    * const req = http.request({
1759    *   host: '127.0.0.1',
1760    *   method: 'GET',
1761    *   // ...
1762    * });
1763    *
1764    * req.on('end', () => {
1765    *   // req.isReadable is true
1766    * });
1767    */
1768   isReadable: boolean;
1769   /**
1770    * Set to true when the request headers have been received and the request body
1771    * is being read.
1772    *
1773    * Note that for HTTP 200, the request method is lowercased.
1774    *
1775    * ''js
1776    * const req = http.request({
1777    *   host: '127.0.0.1',
1778    *   method: 'GET',
1779    *   // ...
1780    * });
1781    *
1782    * req.on('end', () => {
1783    *   // req.isReadable is true
1784    * });
1785    */
1786   isReadable: boolean;
1787   /**
1788    * Set to true when the request headers have been received and the request body
1789    * is being read.
1790    *
1791    * Note that for HTTP 200, the request method is lowercased.
1792    *
1793    * ''js
1794    * const req = http.request({
1795    *   host: '127.0.0.1',
1796    *   method: 'GET',
1797    *   // ...
1798    * });
1799    *
1800    * req.on('end', () => {
1801    *   // req.isReadable is true
1802    * });
1803    */
1804   isReadable: boolean;
1805   /**
1806    * Set to true when the request headers have been received and the request body
1807    * is being read.
1808    *
1809    * Note that for HTTP 200, the request method is lowercased.
1810    *
1811    * ''js
1812    * const req = http.request({
1813    *   host: '127.0.0.1',
1814    *   method: 'GET',
1815    *   // ...
1816    * });
1817    *
1818    * req.on('end', () => {
1819    *   // req.isReadable is true
1820    * });
1821    */
1822   isReadable: boolean;
1823   /**
1824    * Set to true when the request headers have been received and the request body
1825    * is being read.
1826    *
1827    * Note that for HTTP 200, the request method is lowercased.
1828    *
1829    * ''js
1830    * const req = http.request({
1831    *   host: '127.0.0.1',
1832    *   method: 'GET',
1833    *   // ...
1834    * });
1835    *
1836    * req.on('end', () => {
1837    *   // req.isReadable is true
1838    * });
1839    */
1840   isReadable: boolean;
1841   /**
1842    * Set to true when the request headers have been received and the request body
1843    * is being read.
1844    *
1845    * Note that for HTTP 200, the request method is lowercased.
1846    *
1847    * ''js
1848    * const req = http.request({
1849    *   host: '127.0.0.1',
1850    *   method: 'GET',
1851    *   // ...
1852    * });
1853    *
1854    * req.on('end', () => {
1855    *   // req.isReadable is true
1856    * });
1857    */
1858   isReadable: boolean;
1859   /**
1860    * Set to true when the request headers have been received and the request body
1861    * is being read.
1862    *
1863    * Note that for HTTP 200, the request method is lowercased.
1864    *
1865    * ''js
1866    * const
```

- [Lines 200-201](#), first makes sure that when **app.use** is called the parameters passed matches that of the general format.
  - [Lines 203-212](#) gets the path from the first parameter.
  - [Line 214](#) the arguments object represents our parameters, it then makes the arguments object into an array without the path(first argument) for example our server called **app.use('/api/products', products)**; where fns = products and path = '/api/products'.
  - [Lines 216-218](#) checks if there is a function to be run, if not throw an error.
  - [Line 222](#) creates an instance of a router with properties inherited from the app.
  - [Lines 224-228](#) checks if each function that's getting added to the middleware is not already present/non-express if not, then it gets routed. Back to our example, our **'/api/products'** path will now have the properties of **products.js**.
  - [Lines 235-246](#) restores the properties of req and res by using **req.app** which gets the express app that is using the middleware and lets the new stack inherit its properties. Lastly the mounted app gets emitted.
- In the express.js library, **App.get()** is defined in [Line 811](#) of the express.js library.
  - [Line 812](#) returns the value of the header that it was given using:
   
**this.getHeader(field) -> [1]**
    - The **this** object refers to the object inherited from **http.ServerResponse** class (which was set as a prototype during the object instantiation using **Object.create(http.ServerResponse.prototype)**)
    - **getHeader()** is the inherited method from the class described above in node (**http module** in **Node.js**). It is a part of **Http2ServerResponse** class, and is defined on [Line-595](#). It is used to receive the header of the request from the client. This means it is not defined by **express.js**. The implementation of this function corresponds to the function we implemented in **Homework 2**.
      - This function takes a parameter of type string that specifies the header name and returns a string that contains the value of the requested header.
      - If there is no header matching the parameter, it will return **null**. (using the function **validateString** defined on [Line-161](#))
      - If there are multiple headers matching the parameter, it will return the value from the first header.
- Router → **express.Router()** is an instance of the App object of the Express framework, which is a piece of middleware used to define routes as well as handling incoming HTTP requests for a specific routes. It is used to break the App object into

smaller components that can be re-used through out the application.

- Defined in the express.js library at:  
<https://github.com/expressjs/express/blob/8368dc178af16b91b576c4c1d135f701a0007e5d/lib/router/index.js#L46>
  - Creates an another verison of the application being able to perform routing functions alongside `app.use()` with most of the functionalities as the app.
  - It uses a `setPrototypeOf()` function to inherit the class proto with similar functions as the one in app such as `proto.handle()`.
- Example of usage can be seen on these lines in our implementation:
  - `auction.js` → Lines [2](#), [137](#), [161](#)
  - `item.js` → Lines [2](#), [10](#), [54](#), [117](#), [162](#), [199](#), [290](#)
  - `user.js` → Lines [2](#), [10](#), [15](#), [66](#), [115](#), [142](#)
- Express.js imports an array of **lower-cased method names** that Node.js supports from [methods](#) module. It is used by the function on [Line-490](#) in the library, that adds methods like GET/POST/PUT/DELETE etc. to the router object.
- In an Express.js application, the functionality for handling HTTP **GET/POST/PUT/DELETE** requests is implemented in the **`express.urlencoded` middleware** (*built-in middleware function in Express*). This middleware is typically included in the application's middleware stack by calling the **`app.use(express.urlencoded({options}))`** method, where **`app`** is an instance of an Express application, and **`options`** is an optional object that can be used to configure the behavior of the `express.urlencoded` middleware.  
The chain of calls for finding the implementation is as follows:
  - [Line-83](#) exports the **`urlencoded`** method, which is a part of [body-parser module](#) imported on [Line-15](#).
  - In the body-parser module, there are several files to parse different kind of data types. All these files can be found here:  
<https://github.com/expressjs/body-parser/tree/master/lib/types>
  - The implementation of how the requests of these files are being parsed corresponds to the implementation in **Homework-2**.
- Since the **router is an instance of the app application** (it inherits its methods from app object prototype), it have access to the methods defined in the app object. We have described above how these methods are being imported in express.js and implemented in node and http module. Here are more details about the methods specific to the routes object:
  - **`Router.get()`** → Since the router is an instance of the app application, it have access to the methods defined in the app object. Therefore, similar to **`App.get()`** is a method of the Express app, which is used to handle HTTP GET requests that the server receives - server uses this function to determine what

to do when a get request at a given route is called. Router could route to “subroutes” when a request to its main path is accessed.

- Defined in the library at:

<https://github.com/expressjs/express/blob/8368dc178af16b91b576c4c1d135f701a0007e5d/lib/response.js#L811>

- Like `app.get()`, `Router.get()` uses the `getHeader()` method that retrieves the header from the GET request and routes a callback to that path.
- For example in our `auction.js` [Line 137](#), once we get a GET request on the path `/all-auction-items`, we send a list of our items that are still on sale.

- **Router.post()** → Since the router is an instance of the app application, and Express parses the HTTP methods and routes the request to the specified path with the callback functions. Specifically, `Router.post()` tells the server that there is a new resource that needs to be created.

- In our project under `users.js` on [Line 15](#), we can see that the path `/signup` will prompt our server for a post request where it will store the user information and send a post response assigning them a validation token. Similarly, on [Line 66](#) a POST request to the path `/login` will have our server verifying user’s credentials and also generating an authentication token.

- **Router.put()** → This method allows the server to update a resource within the application. The main difference between the `put()` function and the `post()` function is that `put()` is idempotent meaning multiple calls do not affect the integrity of the app, but `post()` might.

- In our project under `items.js`, in [Line 290](#), the path `/resell-marketplace-item` will prompt our server for a put request in which a user could resell an item that they own. Once called, the server checks for user authentication and checks if the user owns the item that they want to resell.
- Once the user and the listing is verified, the items database will be updated with the new data that the user.
- The server responds with a status `200` letting the user know that the database has been updated.



# Handling

- Express handles the HTTP parsing in its request.js and response.js. In request.js, a request object is made that is of **Class: http.IncomingMessage**, which is an object created by the app which is used to access response status, headers, and data. Therefore, the req also supports Node.js req's built in methods. [1]
- Header Parsing:
  - <https://github.com/expressjs/express/blob/8368dc178af16b91b576c4c1d135f701a0007e5d/lib/request.js#L31>
    - [Line 31](#), initializes incoming HTTP request message with object methods.
    - [Line 64](#), defines the functionality of `req.get()` which could be used interchangeably with `req.header()`.
    - [Line 65-Line 72](#), checks if the parameter passed, *name* which is the HTTP header that is requested exists and if it is a string and throws an error if it isn't.
    - [Line 74-Line 82](#), converts *name* to lowercase and checks for a special case "Referer" and accesses the header through using Node.js' Request.headers instance which is accessed through a key-value pair.
      - [https://nodejs.org/api/http.html#http\\_message\\_headers](https://nodejs.org/api/http.html#http_message_headers)
- Express also handles parsing the body of a HTTP request through the use of Express' body-parser library. Within the library we utilize the "urlencoded" and "json" parsers. Implementation follows similar structure to that of **homework 2**.
  - `bodyParser.urlencoded` is used in [Line 12](#) of Server.js and can be found within the express repository [here](#).
    - This particular library specifically parses "application/x-www-form-urlencoded" which is the most common format for HTTP post forms.
    - This library was important for parsing through form data from which allowed the server to get the body of the form such as a key/value pair for user signup in [Line 16](#) of users.js in our project.
  - `bodyParser.json` is used in [Line 13](#) of Server.js and can be found within the express repository [here](#).
    - This library is used to parse JSON and is parsed through the "Content-Type" header.
    - The method parses the request body and makes the request object into

- a javascript object.
  - A new body populated with the parsed json is added to the request object.
- Express also handles parsing the HTTP response such as sending a status codes. The implementation for parsing the HTTP response is found in Express' response.js library where the response object inherits `http.ServerResponse` allowing headers to be set and sent.
  - The setting of HTTP headers follows that of **Homework 1** where its responsible for formatting the headers using the `set()` method.
  - `App.set()` in application.js [Line 359](#) defines the `set()` function as a key/value pair as the parameter where `param(setting)` represents the key and `param(val)` represents the value. In [Line 378](#), HTTP header (setting) is assigned a value.
    - Defined in response.js [Line 250](#) `res.json()` which sends a json response to the client.
    - In [Line 275](#), the response object invokes the `set()` method setting a json HTTP response with headers (`'Content-Type', 'application/json'`) which tells the browser how to parse this content as shown in homework 1.
- The server sending the status code is defined in [Line 369](#) in response.js where it utilizes the HTTP status codes as defined above. This aligns with the required work for **homework 1** where the server has to respond with the required status line format.
  - The method sets the (statusCode) in [Line 372](#) to the defined code that is passed through the parameter.
  - The statuses library then extracts the status message based on the code in [Line 370](#) from the defined `http.STATUS_CODES`. It could also be defined as `res.status(code).send(message)` in [Line 67](#) where the status code is also accessed by invoking the node httpserver status codes and getting the status message.