

**Introduction to Modern Application Development**  
**Dr. Gaurav Raina**  
**Prof. Tanmai Gopal**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 26**  
**Module P11**

**Practical: Introduction to authentication, hashing, curl & sessions**

Hi all welcome to module P11. This will be perfect module that introduces us to the basic concepts of authentication password hashing, a command length tool called curl which is one of the most useful and work style tools for testing HTTP and API endpoints and the concept of sessions, then how to implement them. We will be looking at implementing a basic password hashing and storage mechanism.

(Refer Slide Time: 00:20)

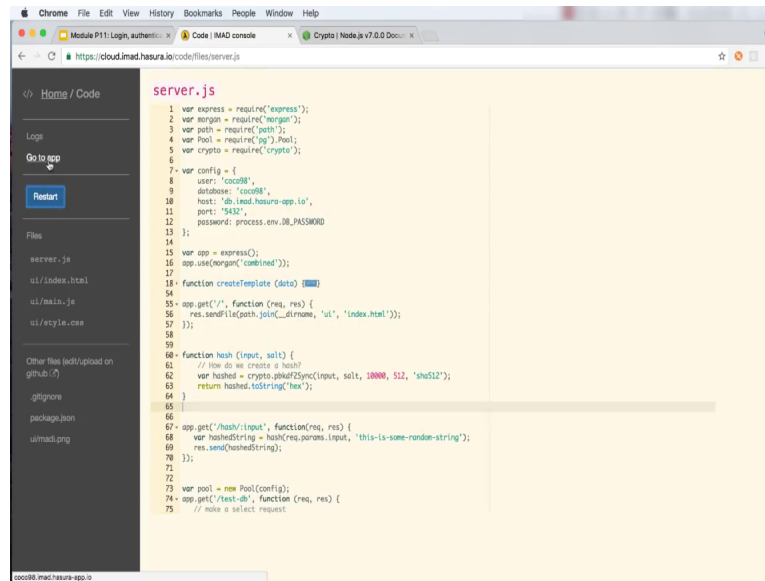
## Contents

Objective:

1. Implement a password storage system
  - a. Password hashing
  - b. Why salt?
2. Implement a login system
  - a. API based
  - b. User creation
3. Implement a session system
  - a. Cookie based
4. Testing our APIs/endpoints using curl

We will also look at the concept of salting or password and why it is required, apart from implanting a login endpoint we will also have a user creation endpoint we will then implement cookie based session system, and then we will testing our APIS and endpoints using curl.

(Refer Slide Time: 00:53)

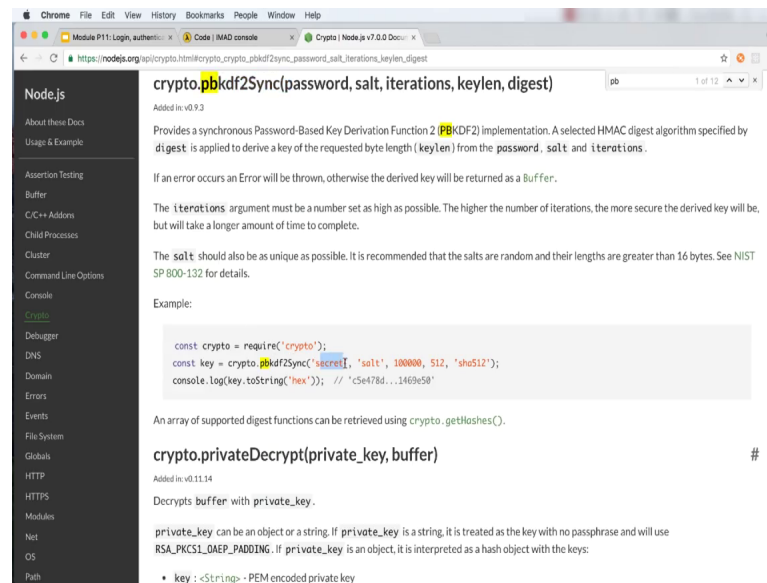


```
server.js
1 var express = require('express');
2 var morgan = require('morgan');
3 var path = require('path');
4 var pool = require('app').pool;
5 var crypto = require('crypto');
6
7 var config = {
8   user: 'ccccc8',
9   database: 'ccccc8',
10  host: 'db.leadhasura-app.io',
11  port: '5432',
12  password: process.env.DB_PASSWORD
13 };
14
15 var app = express();
16 app.use(morgan('combined'));
17
18 function createTemplate (data) {}
19
20 app.get('/', function (req, res) {
21   res.sendFile(path.join(__dirname, 'ui', 'index.html'));
22 });
23
24
25 function hash (input, salt) {
26   // How do we create a hash?
27   var hashed = crypto.pbkdf2Sync(input, salt, 10000, 512, 'sha512');
28   return hashed.toString('hex');
29 }
30
31
32 app.get('/hash/:input', function (req, res) {
33   var hashedString = hash(req.params.input, 'this-is-some-random-string');
34   res.send(hashedString);
35 });
36
37
38 var pool = new Pool(config);
39 app.get('/reset-db', function (req, res) {
40   // make a select request
41 }
```

Let us set to our coding console all right. So, in the last session we will left off at creating a test d b endpoint and then articles end point which was talking to the data base, we will first to start off will look at creating a password hashing endpoint. So, what you would like to do is create an endpoint that takes and input from the user as a part of the URL, and returns a hash string which is represents for the possibilities stored as. So, just to sort of a at a pseudo code what you want to do is well. We extract the input value and then we have a function called hash which we have not (Refer Time: 01:30) yet and if you have this function called hash and then we will return this string back to the user. So, this is kind of what you want to do of course, we need to have this function called hash.

So, let us write the function hash here, which takes is an input the input and what it should do here is it return the hash tag right. So, what we need to now figure out is, how do we create the hash. So, we will be using the default library called crypto that is a part of node you. So, if you had to the documentation for node j s you see library called crypto and there if you go to.

(Refer Slide Time: 02:16)



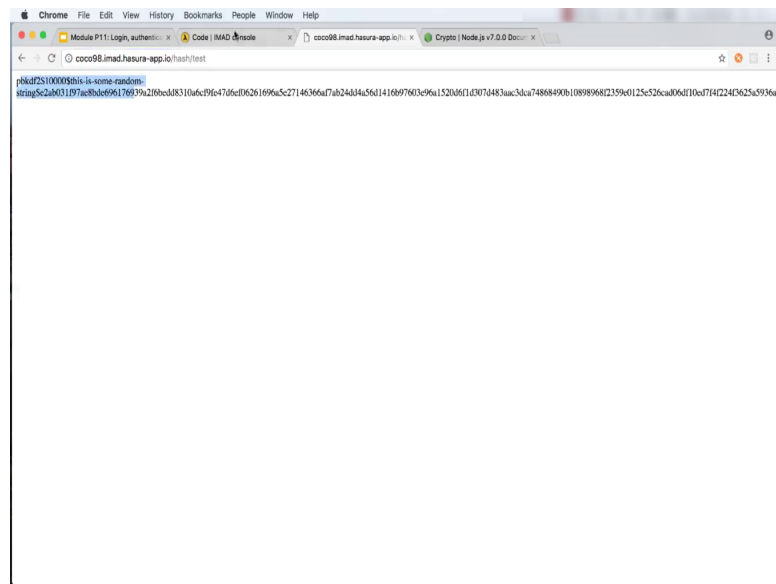
And this function provide us a way to create a hash of a particular input. So, for example, here a value for secret is taken and hash is returned. So, let us see what it looks like. So, this is kind of what we need to do copy the function name here.

Now, of course, we need to have crypto here. So, let us include crypto. So, we include crypto. So, now, we have the crypto library, now what we need to do when we hash this is that we need to give the input we need to give it something called a salt we will understand what is salt is in little bit, the next argument is a number of iterations. So, this number of times the hash functionality of light and again we will understand why we are doing a certain number of iterations. So, let us say we do 10,000 iterations up to which we will use the standard default values that you want a key length of 512 and we will you want to use the sho 512 digest.

Let us not worry about what these are just yet. So, this is what we will do and we will return hashed of course, what we need here is the salt value. So, let us take the input called salt from here and that means that we need to pass and input salt here. So, for now let us say we create temporary salt value let us just create the value called this is some random string. So, I just created a random string here, now when we hash this particular value the output that we get will be sequence of bytes. So, to convert that into something that we can read we are going to convert that to string and use the hexadecimal and coding to convert them into something that is readable and printable on our screen.

Now, let us first quickly see if all of this works and then we will come back and try to understand what this function is actually doing right. So, say this whole go to app and so I typed hash password and this is the output that I got right so; that means, that our hash is working let us try to hash this again. So, you can see there if called the endpoint here I refreshed it called the endpoint again you can see that the value of the hash does not change right, but if I change this.

(Refer Slide Time: 05:02)



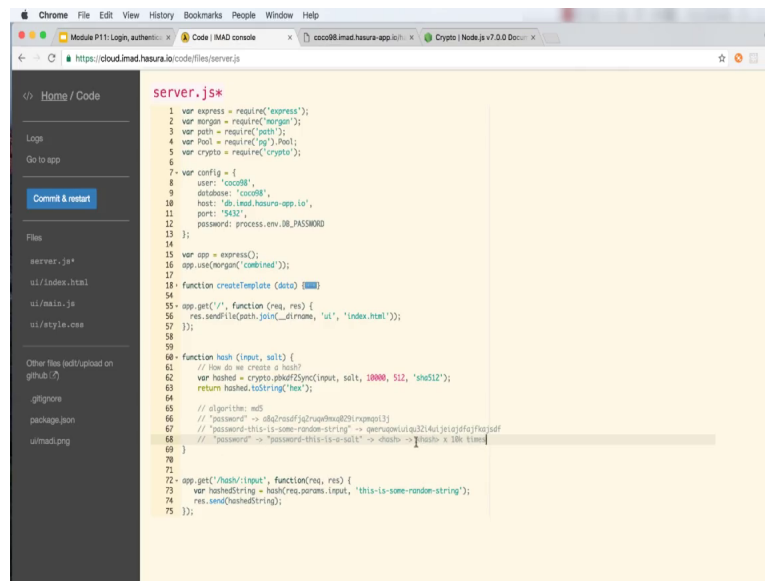
And I make a small change in the input and I make it say password 2 and I refresh it you can see that the entire value looks very different right so; that means, that what is happening is that it is converting this string into a 512 byte string right which is a random value right.

This is how we will store the users password right you want actually store users password as password two row means store it has the hash. So, let us just try out for the few more things let me example test right I refresh this it is the same grid. So, now, let us take a step back and come back to understand what we are doing when we call this function. So, what this function does is that, it converts it takes our input it appends the value of the salt right and then it applies the hash function 10,000 times right why do we do this why are we simply not just doing a hash of the input. For example, if you go to the crypto library you will see there is the function called create hash and what create

hash is doing is that it is taking a particular input and it is converting that into hash value right.

So, why are we just not using a hash right why are we using a special algorithm which is called the password paste key derivation function at why are we using this. The reason is that if we just save the hash without adding this random string the hash should always evaluate to the same value for the same algorithm.

(Refer Slide Time: 06:47)



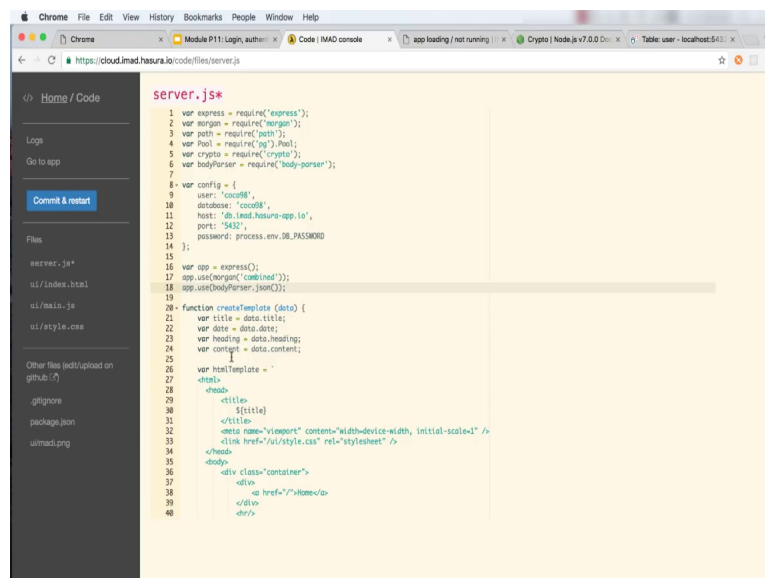
```
server.js
1 var express = require('express');
2 varmorgan = require('morgan');
3 var path = require('path');
4 var Pool = require('pg');
5 var crypto = require('crypto');
6
7 var config = {
8   user: 'cocob8',
9   database: 'cocob8',
10  host: 'db-load.hasura-app.io',
11  port: '5432',
12  password: process.env.DB_PASSWORD
13 };
14
15 var app = express();
16 app.use(morgan('combined'));
17
18 function createTemplate (data) {}
19
20 app.get('/', function (req, res) {
21   res.sendFile(path.join(__dirname, 'ui', 'index.html'));
22 });
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
```

value toward. So, now, the value that will be hashed is this, this value will of course, have some completely different hash value right and there is no table in the world that will have stored this particular hash value because if we choose our salt string randomly enough.

Then there is no way there somebody would have pre created tables that contain the hash values for all the commonly non strings and that way even if we users use strings that are very common words for passwords for example, password or common names even then by adding this random salt value, we can ensure that the values of the hash generated as you need and cannot be reverse engineer. Further to protect ourselves even more p hash the value 10,000 times which means that a particular value password is first taken this is then converted to a value that contains the salt, this is then converted into a hash and this is converted into another hash and so on 10,000 times right and. So, the final value that is obtained that the hash is certainly not going to be present in any kind of a lookup table or a hash table somewhere that will allow hackers to find out what the original string was right.

So, now that we have done this let us make our hash function return; return a slightly different string and we will understand why we will turning that string a little later. So, let us create an array of strings where I first store the name of the algorithm that I am using for doing the hashing, I then store the number for iterations.

(Refer Slide Time: 10:04)

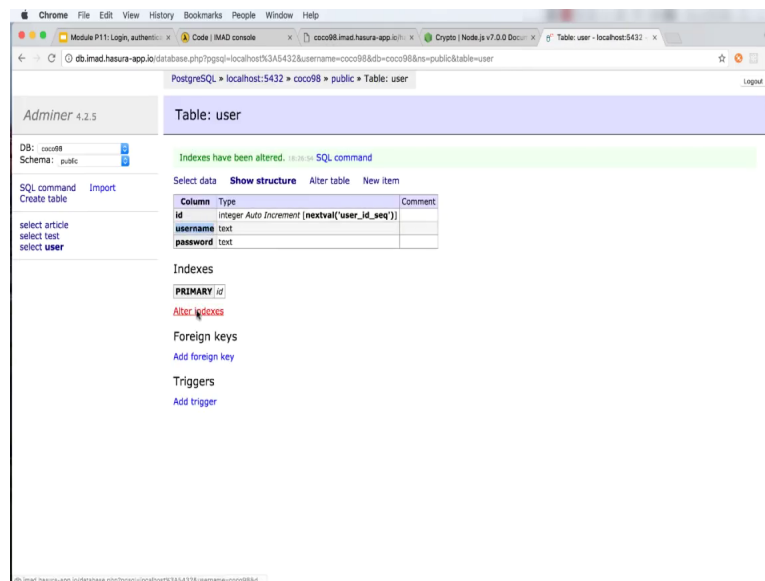


```
1 var express = require('express');
2 var mongoose = require('mongoose');
3 var path = require('path');
4 var Pool = require('pg').Pool;
5 var crypto = require('crypto');
6 var bodyParser = require('body-parser');
7
8 var config = {
9   user: 'postgres',
10  database: 'postgres',
11  host: 'db.lead.hackerrank.com',
12  port: '5432',
13  password: process.env.DB_PASSWORD
14 };
15
16 var app = express();
17 app.use(morgan('combined'));
18 app.use(bodyParser.json());
19
20 function createTemplate (data) {
21   var title = data.title;
22   var date = data.date;
23   var heading = data.heading;
24   var content = data.content;
25
26   var htmlTemplate = `
27     <html>
28     <head>
29       <title>
30         ${title}
31       </title>
32       <meta name="viewport" content="width=device-width, initial-scale=1" />
33       <link href="/ui/style.css" rel="stylesheet" />
34     </head>
35     <body>
36       <div class="container">
37         <div>
38           <a href="/">Home</a>
39         </div>
40       </div>
```

So, that is 10,000 iterations, I will then store the salt value and after storing the salt value I stored the hashed value right and then once I have these values, I will join them with electro dollar right. So, let us say this to the string that we have turning contains the name of the algorithm, the number of iterations the random salt value and in the fine hashed value. What is important to know is that even if a hacker has access to this entire string which is the hash value the salt and the number of iterations and the hacker also knows that we are using the algorithm p b k d f 2; there is no way that the hacker will be able to figure out that this hash actually comes from the string called test and that is why this is the string that will actually store inside the database.

Let us go ahead now and create our database that we will store the username and password fields up. So, let us head to our database console let us create a user table. So, I will go to create table, I will create an id which should be my primary key create a username which is a text column.

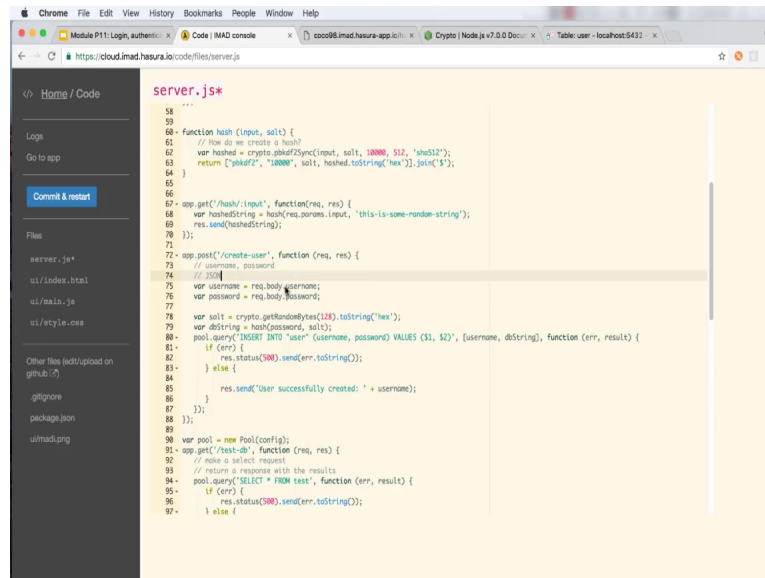
(Refer Slide Time: 11:21)



And then I will create password which is also text column and then I said id to be the primary key and because we will be fetching data by the user name very often, I will make this a unique index now so to ensure that duplicate usernames will never rise. Now let us implement a function to create a new user. So, this function has an input we will take the username and the password and it will create an entry in the user table.

So, let us assume that we somehow have the username and password that was sent to us in the create user request and let us create the password.

(Refer Slide Time: 12:34)



```
server.js
58
59
60 function hash (input, salt) {
61   // How do we create a hash?
62   var hashed = crypto.pbkdf2Sync(input, salt, 10000, 512, 'sha512');
63   return [input, salt, hashed.toString('hex').join('$');
64 }
65
66
67 app.get('/hash/:input', function (req, res) {
68   var hashedString = hash(req.params.input, 'this-is-some-random-string');
69   res.send(hashedString);
70 });
71
72 app.post('/create-user', function (req, res) {
73   // username, password
74   // salt
75   var username = req.body.username;
76   var password = req.body.password;
77
78   var salt = crypto.getRandomBytes(128).toString('hex');
79   var dbString = hash(password, salt);
80   pool.query('INSERT INTO `user` (username, password) VALUES ($1, $2)', [username, dbString], function (err, result) {
81     if (err) {
82       res.status(300).send(err.toString());
83     } else {
84       res.send('User successfully created: ' + username);
85     }
86   });
87 });
88
89
90 var pool = new Pool(config);
91 app.get('/view-db', function (req, res) {
92   // make a select request
93   // return a response with the results
94   pool.query('SELECT * FROM test', function (err, result) {
95     if (err) {
96       res.status(300).send(err.toString());
97     } else {
```

So, there db string is equal to hash password comma salt; obviously, we need to generate a salt for this user. So, let us quickly generator a salt as well. So, you use the get random by it is function to generate the salt, now once you have this database string we want to save it in the database. So, we will do a pool dot query, you will make an insert query. So, insert into the user table that is important to use his double quotes here because for (Refer Time: 13:09) user is a bit for reserved keyword and we want to insert the columns username and password, and the values that we are inserting into it are dollar one and dollar two which represent two elements in an array, which are going to be username and db string. So, db string is the hashed password once we have. So, I will write the call back here and again we will handle the errors in the same way. So, for example, if you are obtaining an error and I am going to copy paste, I am going to copy the code that we have from earlier.

So, in case there is an error I will reply with error and otherwise we will send a response saying user successfully created with the username right. So, the first problem that we need to solve is where is username and password, where are these two values going to come from. We can do we can use the same method of making it a part of URL where we can have username and password right as a part of the URL, but that is not recommended



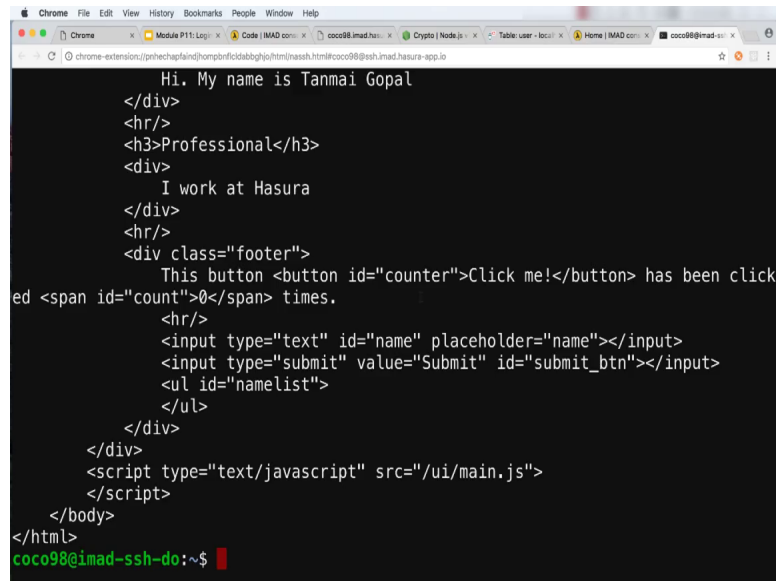
practice because ideally we should not be sending data in the get requests right especially because the password is raw and in case for example, let us say the logs will be printed and the logs will show and the logs will show what password is being printed and this is very dangerous because anybody who has access to the logs will have access to the password in which case it makes sense to make this a post request right and assume that we will be able to extract the username and the password from the request body.

So, let us say that where user name is equal to req dot body dot username and let us say where password is equal to req dot body dot password. So, now, we come to the next question which is where is this data coming in the req dot body and what is the format of the data that is coming in. So, we are going to assume that this is json request, and if this is the json request we have to tell our express framework to look for these keys inside the request body and this request body is going to be a json and the way if we do that is to you something called the body parser which is an express library, and we need to tell our express app that in case you see json content, load the json content in the req dot body variable.

So, we will be doing and this is the way of telling our express framework that for every incoming request in case it sees a content type json, it uses that and such req dot body we need to figure out how we are going to test this right because this is not a URL that we can for example, do this way. For example, if you make the request to create user such cannot get such create user because this is not a get request, but a post request. So, the question that we have to answer now is how do we make a post request and how do we test out this API end point that we have made. In our application ultimately when we use this create user API we will be writing code in main dot j s function right and side our request here and instead of making a get request that you making here when we are submitting to the name API we will actually changes to post request.

This is how we will do it in when we write the java script that will actually use this API, but when we just want to test this out we will use the tool called curl. So, to use curl because you might not have curl installed on your windows systems, let us go back to our s h console I will take my (Refer Time: 17:17) here. So, I am logged into my (Refer Time: 17:24) console I am going to zoom that up a little bit.

(Refer Slide Time: 17:31)



```
Hi, My name is Tanmai Gopal
</div>
<hr/>
<h3>Professional</h3>
<div>
  I work at Hasura
</div>
<hr/>
<div class="footer">
  This button <button id="counter">Click me!</button> has been click
ed <span id="count">0</span> times.
<hr/>
  <input type="text" id="name" placeholder="name"></input>
  <input type="submit" value="Submit" id="submit_btn"></input>
  <ul id="namelist">
  </ul>
</div>
</div>
<script type="text/javascript" src="/ui/main.js">
</script>
</body>
</html>
coco98@imad-ssh-do:~$
```

So, let us quickly check out what curled as. So, if I do curl this actually queries Google dot com and whatever is h t m l response of Google dot com it shows that in the terminal. This is exactly what the browser would have done, but the browser instead would have, but the browser instead would have displayed the h t m l, now instead of displaying the h t m l this is just going to display the h t m l string here.

So, what we are going to do is we are going to make a request to our app I made a mistake here, I did not put I m a d. So, let me add I m a d right and you can see that the h t m l has loaded right.

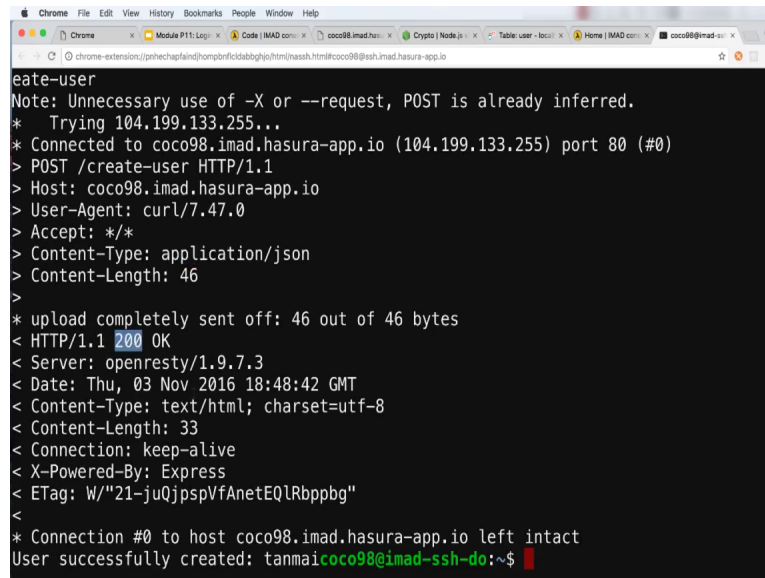


And now let us enter the u r l. So, it should be. So, that is the request that we are making now if you see the error that I am getting it is saying that it sent me an error saying that crypto dot get random bytes is not a function. So, we have made a mistake in the code that I have wrote. So, let us go to the crypto documentation and save it. So, it is called random bytes the word get is wrong. So, let us remove this and changes to random bytes from it and restart this app and let us make the same request again. So, I press the (Refer Time: 19:27). So, the error that will making is that if you should read the error carefully and it says that the p b k d f two function is receiving a wrong value and it is saying that it is not receiving a buffer by a buffer it means a string buffer.

So, it is not receiving the right type of value and what; that means, is that somehow it is not able to access the username and password. Now why is it not able to access the username and password because the username and password is probably not coming in from this req dot body here right and why is this json object not being loaded? The reason why this json object is not being loaded or we are not able to extract the username and password from the update correctly is because express does not know that we sent it json. So, how do we tell express that we are sending it the content type json in the data that we are sending because only if we tell express that we are sending the json content type it will use the body (Refer Time: 20:20) and extract the json value and put it into req dot body. So, the way to do that is to add a http header right and called content type.

So, this is the content type header and we have submitting this content type header, I am going to use the minus v flag on curl to see the request in more detail that curl is making, let us you the request being made is. So, it is saying that it is making a post request to the create user end point it is sending the content type as application json.

(Refer Slide Time: 21:05)



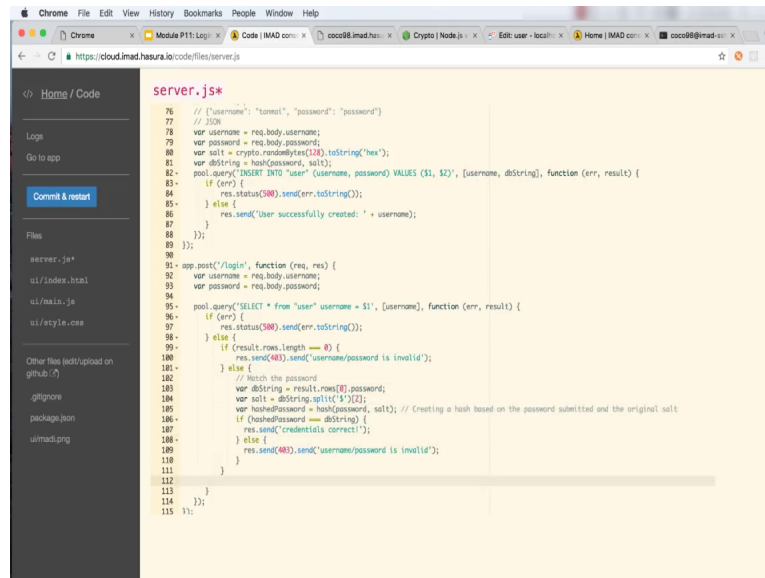
```
chrome
chrome
Module P11: Logg
Code | IMAD com
coco98.imad.hes
Crypto | Node is
Table user - loc
Home | IMAD com
coco98@imad-ss

create-user
Note: Unnecessary use of -X or --request, POST is already inferred.
* Trying 104.199.133.255...
* Connected to coco98.imad.hasura-app.io (104.199.133.255) port 80 (#0)
> POST /create-user HTTP/1.1
> Host: coco98.imad.hasura-app.io
> User-Agent: curl/7.47.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 46
>
* upload completely sent off: 46 out of 46 bytes
< HTTP/1.1 200 OK
< Server: openresty/1.9.7.3
< Date: Thu, 03 Nov 2016 18:48:42 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 33
< Connection: keep-alive
< X-Powered-By: Express
< ETag: W/"21-juQjpspVfAnetEQlRbpbpg"
* Connection #0 to host coco98.imad.hasura-app.io left intact
User successfully created: tanmai@coco98@imad-ssh-do:~$
```

The response that was received is HTTP 200 which means there are response a successful, and if you look at the response it says that the user has been successfully created and the user is tanmai. So, let us go and look at our data and so we see that this entry has been received. So, now, let us do the tricky bit of actually login this user in. Login is also going to be a post request because it is going to accept the same arguments username and password, but instead of inserting them into the database it is actually going to fetch the value from the database to check if the value is matching.

So, let us copy paste this code here we do not need to create a salt. So, let us remove this let us remove right; what we now want to do is actually select from the user table. So, let us do select start from right and we have selected using the username value.

(Refer Slide Time: 21:58)

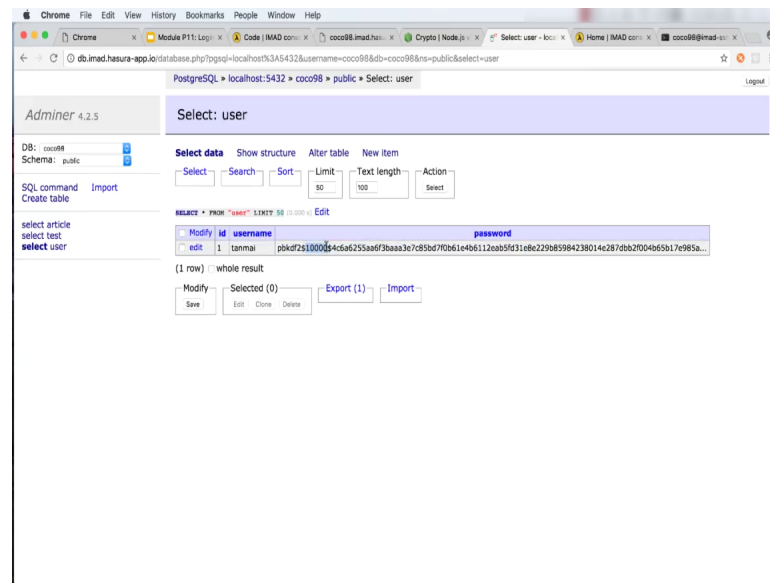


```
server.js
76 // ['username': 'tomcat', 'password': 'password']
77 // JSON
78 var username = req.body.username;
79 var password = req.body.password;
80 var salt = crypto.randomBytes(128).toString('hex');
81 var dbString = hash(password, salt);
82 pool.query('INSERT INTO `user` (username, password) VALUES ($1, $2)', [username, dbString], function (err, result) {
83   if (err) {
84     res.status(500).send(err.toString());
85   } else {
86     res.send('User successfully created: ' + username);
87   }
88 });
89 });
90
91 app.post('/login', function (req, res) {
92   var username = req.body.username;
93   var password = req.body.password;
94
95   pool.query('SELECT * from `user` where username = $1', [username], function (err, result) {
96     if (err) {
97       res.status(500).send(err.toString());
98     } else {
99       if (result.rows.length === 0) {
100         res.send(403).send('username/password is invalid');
101       } else {
102         // Match the password
103         var dbString = result.rows[0].password;
104         var salt = dbString.split('$')[2];
105         var hashedPassword = hash(password, salt); // Creating a hash based on the password submitted and the original salt
106         if (hashedPassword === dbString) {
107           res.send('credentials correct');
108         } else {
109           res.send(403).send('username/password is invalid');
110         }
111       }
112     }
113   });
114 });
115 });
```

So, selected where the user name is equal to dollar 1, and dollar 1 is the username value. So, what we are going to do is first search in our table to see this username exists. As this username exists and if the SQL response is successful we need to quickly check if no rows were received, which means that we will send the message saying we have send a four naught three which means that it is a forbidden request and we will say that username or password is invalid right. So, this is the responsible that we sent back and in case we do have a certain number of rows then we know that the user name is that the username exists.

Now, what we need to do is now we need to match the password. So, first let us extract the password that is stored in the database. So, let us call it d b string which is result dot rows the first element in the password field.

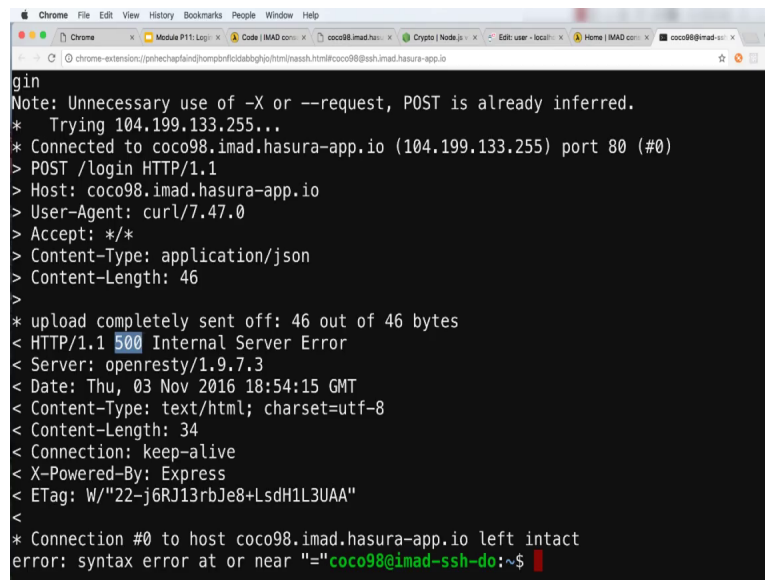
(Refer Slide Time: 23:10)



Now, if you look at the password field we have store the number of iterations here, this value is the salt right and after that is the actual password data. So, we can use the split function to split by dollar. So, this will now return an array right the value of this thing we will be exactly go to use in the hash here. So, it will this array right and so what you want to do is extract the salt value. So, the salt value is the third element in the array right. So, we have the salt value now. So, now, let us create a hash using the salt value. So, we will say that hash is equal to. So, here we are creating a hash based on the password submitted and the original salt right and the test of testing whether this is a valid user is to test if this hashed password is exactly equal to the value that was originally stored in the database, and this is correct that makes the user is successfully logged in.

So, we will return a message saying credentials to the correct and if this is wrong you will return an error message thing that the username or password is invalid let us see this in action . So, now, instead of making a request to the create user endpoint let us make a request to the login endpoint right and we are going to use the same data username.

(Refer Slide Time: 25:05).



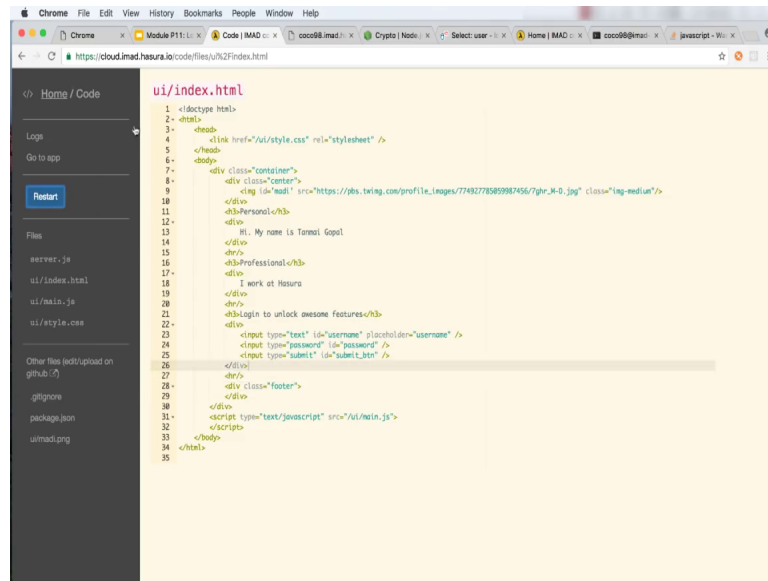
```
gin
Note: Unnecessary use of -X or --request, POST is already inferred.
* Trying 104.199.133.255...
* Connected to coco98.imad.hasura-app.io (104.199.133.255) port 80 (#0)
> POST /login HTTP/1.1
> Host: coco98.imad.hasura-app.io
> User-Agent: curl/7.47.0
> Accept: */*
> Content-Type: application/json
> Content-Length: 46
>
* upload completely sent off: 46 out of 46 bytes
< HTTP/1.1 500 Internal Server Error
< Server: openresty/1.9.7.3
< Date: Thu, 03 Nov 2016 18:54:15 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 34
< Connection: keep-alive
< X-Powered-By: Express
< ETag: W/"22-j6RJ13rbJe8+LsdH1L3UAA"
<
* Connection #0 to host coco98.imad.hasura-app.io left intact
error: syntax error at or near "="coco98@imad-ssh-do:~$
```

And password something of 500 internal error and it is saying this is syntax error that I have made right which is an error that I have made in writing the code. So, so you are returning an error here this is where we returning 500 error and so that means, says in error in our s q l query. So, if you look at the s q l query I say let us start from user and if forgot to put the where keyword. So, let us put the where keyword here let us make this more readable it let us save this let us make a query again great so; that means, that a credentials are matched.

Lets deliberately change our credentials here for example, let us send the wrong password and I get a response saying for bidden right which is a 403 right that is the response that I get; that means, that we have now implemented a login function which is not actually doing anything, but it is just testing that the credentials are valid what we ideally want to do here we want to set a session. So, that once the users logged in the users stays logged in, but before we add session let us actually implement the login on the u I so; that means, that what we will do is if I go to the home page then what I would like to do is replace all of this and change it will login from here. So, let us quickly make those changes let us go to index dot h t t m l let us remove all this right we do not want anything here let us instead change this to log in to unlock cause and features ok



(Refer Slide Time: 26:27)

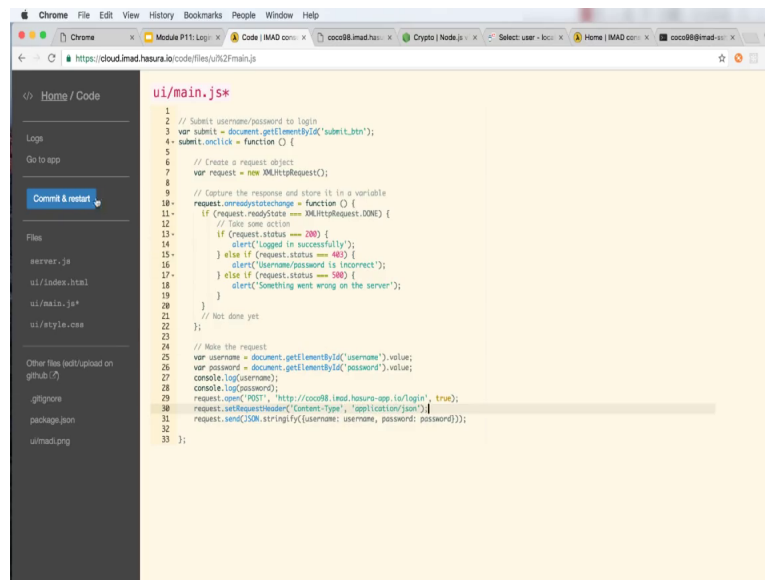


```
1 <!doctype html>
2 <html>
3 <head>
4 <link href="/ui/style.css" rel="stylesheet" />
5 </head>
6 <body>
7 <div class="container">
8 <div class="center">
9 
10 </div>
11 <h3>Person</h3>
12 <div>
13 Hi, My name is Tannai Gopal
14 </div>
15 <div>
16 <h3>Professional</h3>
17 </div>
18 I work at Hasura
19 </div>
20 <div>
21 <h3>Login to unlock awesome features</h3>
22 <div>
23 <input type="text" id="username" placeholder="username" />
24 <input type="password" id="password" />
25 <input type="submit" id="submit_btn" />
26 </div>
27 </div>
28 <div class="footer">
29 </div>
30 </div>
31 <script type="text/javascript" src="/ui/main.js">
32 </script>
33 </body>
34 </html>
35
```

So, now what we are going to do is. So, what we are going to say you are going to click here input type element called type text and make this is the username field, here which is called password which is the input type password, we will give this element it h t m l element in idea of password of it is we want have a place hold of this right and then we will have a button and this is the submit button right. So, let us go to our min dot j s right and let us remove all this counter code that we had right and now instead of having the submit name we are going to this submit username password to login right.

So, now what we are going to do is we are going to be making a post request right and in the post request we will be sending some data. So, instead sending null we will be actually sending data here, and the data that we want to send is json string.

(Refer Slide Time: 28:01)



```
1 // Submit username/password to login
2 // submit = document.getElementById('submit_btn');
3 var submit = document.getElementById('submit_btn');
4 submit.onclick = function () {
5
6 // Create a request object
7 var request = new XMLHttpRequest();
8
9 // Capture the response and store it in a variable
10 request.onreadystatechange = function () {
11 // Take some action
12
13 if (request.status === 200) {
14 alert('Logged in successfully');
15 } else if (request.status === 403) {
16 alert('Username/password is incorrect');
17 } else if (request.status === 500) {
18 alert('Something went wrong on the server');
19 }
20 }
21 // Not done yet
22 };
23
24 // Make the request
25 var username = document.getElementById('username').value;
26 var password = document.getElementById('password').value;
27 console.log(username);
28 console.log(password);
29 request.open('POST', 'http://coco98.imad.hasura.io/login', true);
30 request.setRequestHeader('Content-Type', 'application/json');
31 request.send(JSON.stringify({username: username, password: password}));
32
33 };
```

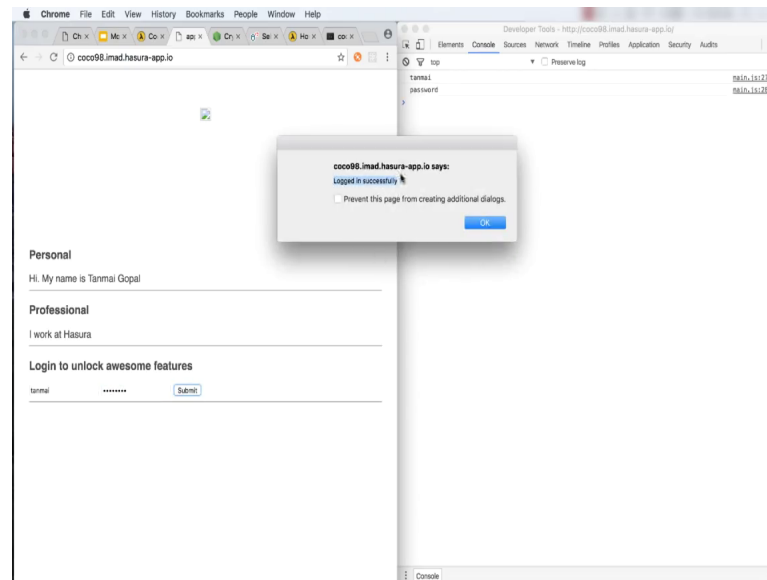
And. So, let us convert an object into a json string and so, the object that we will have is username let us assume we were able to extract the username value and password let us use that variable to extract the password value right. So, where is this username and password coming from as soon as the button is clicked we will extract it from the input element. So, let us change this to user name. So, username will be this start value and similarly password will be obtained from the password element right.

So, we will extract. So, every time the button is clicked we will figure this out part we will figure this part later, but every time is the button is clicked we will extract the username we will extract the password right. Just for debugging purposes let us print this out right we will be making a request to the URL slash login right that is the post request and once this request is sent right we will have to handle the end of the request. So, this request is successful we have to save the user is logged in. So, we will remove this and we will say right and what we can have in fact do to is we can print out in alert box saying that logged in successfully right and in case there was an error. So, let us say have the status was not 200, let us say if the status was 403 we will say that the password is incorrect right.

And the other possibility is that the error status is 500, in which case we will say that something went wrong on the server right we do not know what the error is some 100 error. So, let us make sure the these are else if conditions right and they should be string

if I the content type here. So, let us said the header here this is the same thing that we did not curl. So, it can save this. So, reload this page. So, we have page load here let us load upon stuck element, actually this is half the page that is the console here.

(Refer Slide Time: 31:19)



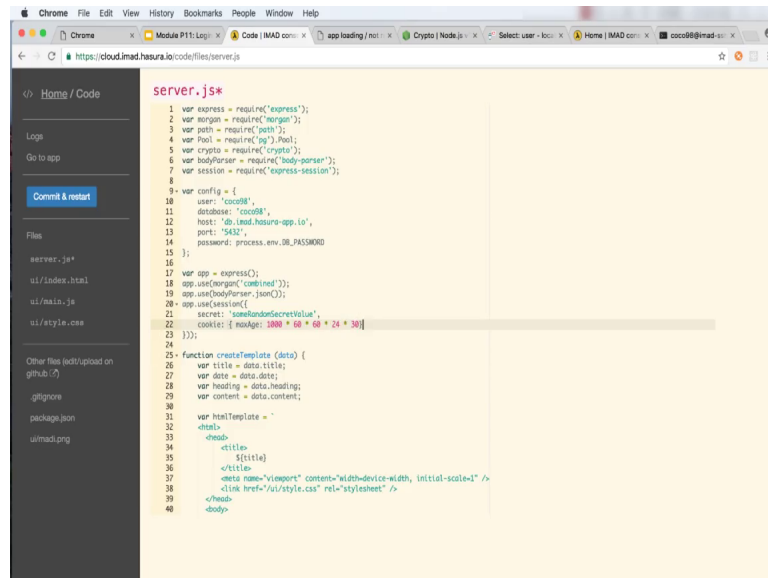
So, that we can see what is going on here the console let us make a request I click on submit and I get it this alert box saying logged in successfully right is printing out the debug things if you going to the network tab, you will see that the login request is made it is a post request right and saying a response credentials correct.

It sending json object right it sending the right content type previous in json. Let us change this (Refer Time: 31:38) something else and submit and you will see the username password is incorrect if you look at this request that is being sent it is responding to the 403 right exactly like how we were testing out to the curl. So, now, let us get two adding sessions. So, when we add a session what we mean by that is that once the users logged in, there is some way of telling the client that take this particular session id and if you have the session id and you ever make a request again with the same session id, then I will know that you were the same user who made the login request.

So; that means, we need to respond with a particular session id and the client needs to remember that session id and send that back to us next time. Since this is a very common functionality there is several libraries to this and we will be using the node session library to this to make it easy to send the session id to the web client and for the web

client to repeatedly send as the same session id again and again we will use cookies let us use the express session library it to this. Once again I have to tell express to use the session library, I we are going to use the session library there are two configurations that I need to give the session library one configuration is the secret which is the value that it will used to encrypt the cookies with.

(Refer Slide Time: 32:57)

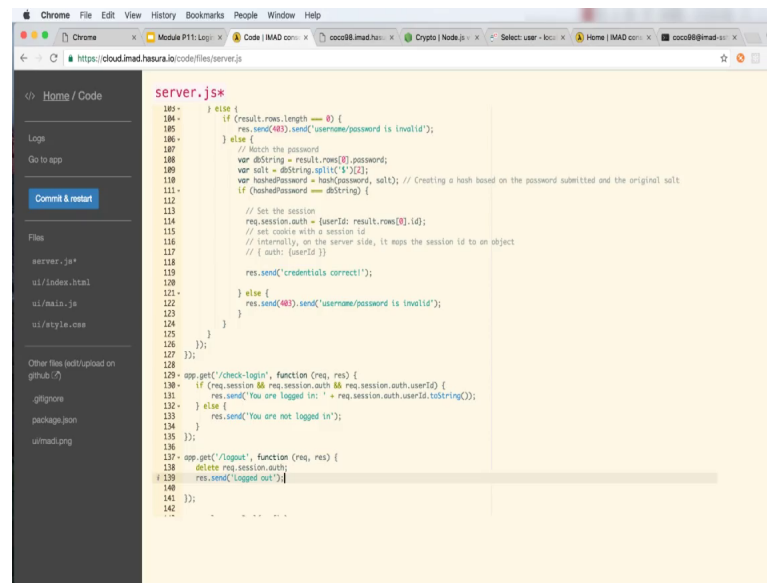


```
server.js
1 var express = require('express');
2 var Morgan = require('morgan');
3 var path = require('path');
4 var Pool = require('pg').Pool;
5 var crypto = require('crypto');
6 var bodyParser = require('body-parser');
7 var session = require('express-session');
8
9 - var config = {
10   user: 'coco98',
11   database: 'coco98',
12   host: 'db-1mad-hasura-app-1o',
13   port: '5432',
14   password: process.env.DB_PASSWORD
15 };
16
17 var app = express();
18 app.use(morgan('combined'));
19 app.use(bodyParser.json());
20 app.use(bodyParser.urlencoded());
21 app.use(session({
22   secret: 'someRandomSecretValue',
23   cookie: { maxAge: 1000 * 60 * 24 * 30 }
24 }));
25
26 function createTemplate (data) {
27   var title = data.title;
28   var date = data.date;
29   var heading = data.heading;
30   var content = data.content;
31
32   var htmlTemplate = `
33   <html>
34     <head>
35       <title>
36         ${title}
37       </title>
38       <meta name="viewport" content="width=device-width, initial-scale=1" />
39       <link href="/ui/style.css" rel="stylesheet" />
40     </head>
41     <body>
```

So, I will just set this to some random value if now right and we have to tell the session library that when it creates cookies, the cookie should have a particular age.

So, we will say that the cookies and so, we will said the cookie can have a max age of. So, this this is value specified in milliseconds. So, you will say 1060, which is one minute and 260 which is one hour into 24 which is one day into 30. So, all are cookies are long lasting cookies they will last for a month once a cookies said it will be set for a month. So, now, we can come back and try to set the session value, we have to set the session value before we actually send the response. So, let us move this up here.

(Refer Slide Time: 33:55)



```
server.js
185- } else {
186-   if (result.rows.length === 0) {
187-     res.send(403).send('username/password is invalid');
188-   } else {
189-     // Match the password
190-     var dbString = result.rows[0].password;
191-     var salt = dbString.split('$')[2];
192-     var hashedPassword = hash(password, salt); // Creating a hash based on the password submitted and the original salt
193-     if (hashedPassword === dbString) {
194-       // Set the session
195-       req.session.auth = {userId: result.rows[0].id};
196-       // set cookie with a session id
197-       // Internally, on the server side, it maps the session id to an object
198-       // { auth: {userId}}
199-       res.send('credentials correct!');
200-     } else {
201-       res.send(403).send('username/password is invalid');
202-     }
203-   }
204- }
205- }
206- });
207- });
208-
209- app.get('/check-login', function (req, res) {
210-   if (req.session && req.session.auth && req.session.auth.userId) {
211-     res.send('you are logged in: ' + req.session.auth.userId.toString());
212-   } else {
213-     res.send('you are not logged in');
214-   }
215- });
216-
217- app.get('/logout', function (req, res) {
218-   delete req.session.auth;
219-   res.send('logged out');
220- });
221-
222- ...
```

So, let us set the session as req dot session dot auth of is equal to user id which will be result dot rows of 0 dot id right. So, what we have done here is that we have assume that there is a session object on the request right just like we assume that there was a body on the request which was created by body parts or they are assuming a there is a session object on request which is being created by the session library.

So, there is a req dot session, we have saying that there is going to be a key called auth inside that object and that key will map to this particular object. And what is this subject this subject says that there is a user id and the user id value is equal to the id of the user that I have got in the database. So, what is actually happening in the background? So, in the background what is happening is that the session library the session middleware to be technically correct is setting a cookie with a session id right that it is randomly generating by itself. Internally on the server side it maps the session id to an object what is this object contained? This object contains a value called auth and what does auth contain? Well auth intern contains another object which is our user id object right. So, this information is maintained in the server side all that the cookie contains is a section id right as soon as we do this here the express session library we will automatically make sure that this object is saved internally as soon as the response is sent.

So, how do we test that this session object is actually being created? Let us create another endpoint which we can use in the browser called check login and what this

endpoint we will do is that it will check that if there is a req session object, and if there is a session object it will check if this and auth object inside it and if there is an auth object it will check if there is a user id key inside their solve inside this auth object and if there is then we will return a response saying that you are logged in and the user id that you have is req dot session dot auth dot user id right and this is an integers let us convert that to string. So, this is the value here, in case this none of these objects are found we know that the user is not logged in. So, we will return as think you are not right. So, let us put this in action to see if sessions are working for us.

So, let us enter the username and password. So, we saying that you logged in successfully let us now go back to the check login and you can see that it is saying that I am logged in is one right. So, now, let us try to implement a logout function. So, let us say that there is a logout endpoint which is again a get request, because no data needs to be sent and if this happens what we will do is that we will remove the auth object right and so, this deletes the auth object from the session object right so that means, that we are not storing the session anymore right. So, let us send respond in that your logged out let us say this. So, I refresh check logged in and not logged in, every time I refresh the browser because the internet session objects are reset then the sessions are lost. So, I do not stay logged in because when I restart the server the session object that is maintained internally is lost. Let us go back to our login page and enter a value here I am logged in unsuccessfully.

So, now let us go to check login and I can say there I am logged in is one in keep refreshing this, let us now call the logout endpoint logout end point says that I am logged out now let us go back toward check log in and it is say that you are not logged in right because this session object that we are maintaining has been deleted bias right. So, this is how we implemented a very simple login and logout functionality and we also maintained sessions.

(Refer Slide Time: 38:34)

**Todo**

1. Add a registration page
2. Check if the user is logged in
  - a. If the user is not logged in, show login/registration form
  - b. If the user is logged in, show a welcome 'user' section
  - c. If the user is logged in, show a logout button on the header
3. Add a comments section to all the article pages
4. If the user is logged in, allow the user to submit comments

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

Slide 5 Presenter view Exit

So, now let us put together all the knowledge that we have gained so far, and create a registration experience we should check is the user is logged in and we should use that we will shows there is a registration form then to show logout button we can then add a comment section to all the article pages and if the users logged in then we should a logged the user to submit comments, but even if the users not logged in we should be able to see all the comments. You should try to do this exercise by yourself to sort of really see you will your knowledge about how API is work, how sessions work, how login works, how the database works how frontend java script to works and how back and java script works.

(Refer Slide Time: 39:14)

## Final result

1. Demo of all basic features working together
2. Check out source code at:
  - a. <https://github.com/coco98/imad-2016-app>
3. Check out SQL schema if required at:
  - a. [Github - coco98 - schema.sql](#)
  - b. Images below as reference to create comments table

Column name	Type	Length	Options	NULL	AI?	Default value
id	integer			<input type="checkbox"/>	<input checked="" type="checkbox"/>	nextval('comment_id_seq')
article_id	integer			<input type="checkbox"/>	<input type="checkbox"/>	
user_id	integer			<input type="checkbox"/>	<input type="checkbox"/>	
comment	text		(collation)	<input type="checkbox"/>	<input type="checkbox"/>	
timestamp	timestampz			<input type="checkbox"/>	<input type="checkbox"/>	current_timestamp

Table name: comment Save

Column Type Comment  
id integer Auto Increment [nextval('comment\_id\_seq')]  
article\_id integer  
user\_id integer  
comment text  
timestamp timestampz [now()]

Indexes  
PRIMARY id  
Alter indexes

Foreign keys  
Source Target ON DELETE ON UPDATE  
article\_id article\_id NO ACTION NO ACTION Alter  
user\_id user\_id NO ACTION NO ACTION Alter

Add foreign key

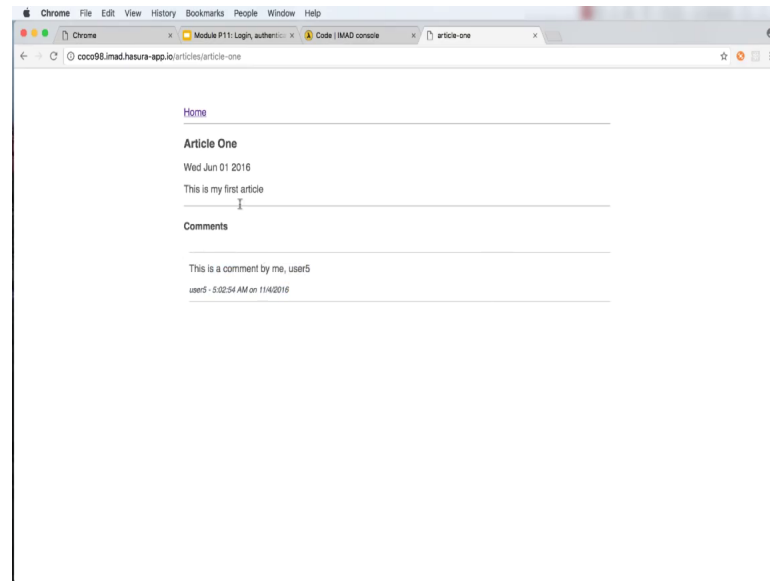
Triggers  
Add trigger

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

I have created this application myself. So, let us have a look at what this application looks like. So, I try to register the new users a user 5, users have been created successfully. So, now, I am registered. So, now, I am going to try to login with the same credentials. So, I will try to log in as soon as the login is done this area changes to a logout section. So, I can go to any particular article let us say got to article one there are no comments here. So, I can post a comment saying this is the comment by me user 5 and let us post this comment. Let us comment is appeared let us go back to the home page let us logout and I have to go back to article one we will see with the comment remains there, but the box two insert the comment has disappeared right.



(Refer Slide Time: 40:07)



All the source code for this is available on this github repository. So, you can use that for a reference the a new comment table was created to be able to stored comments and you can check the SQL for that again inside the github repository you can or you can also use the screenshots of adminer to see how this comment table was created.

(Refer Slide Time: 40:31)

**Very Important things to remember!**

1. This entire exercise is educational!
2. Don't try to implement password hashing completely manually
3. In a production app try to use libraries like passport.js to make life easy and to make sure that your implementation is secure.

Introduction to Modern Application Development Dr Gaurav Raina (IIT Madras), Tanmai Gopal (Hasura)

So, the important thing to remember as we bring this module to close is that the entire exercise is educational and you should try not to implement things like password hashing completely manual like we have done ourselves, I delete for example, if you using node j

s you should use a library like passport or j s to make life easy and almost all other languages and frameworks provide a kind of system to take care of authentication so that we can ensure that the implementation is secure and easy to do.