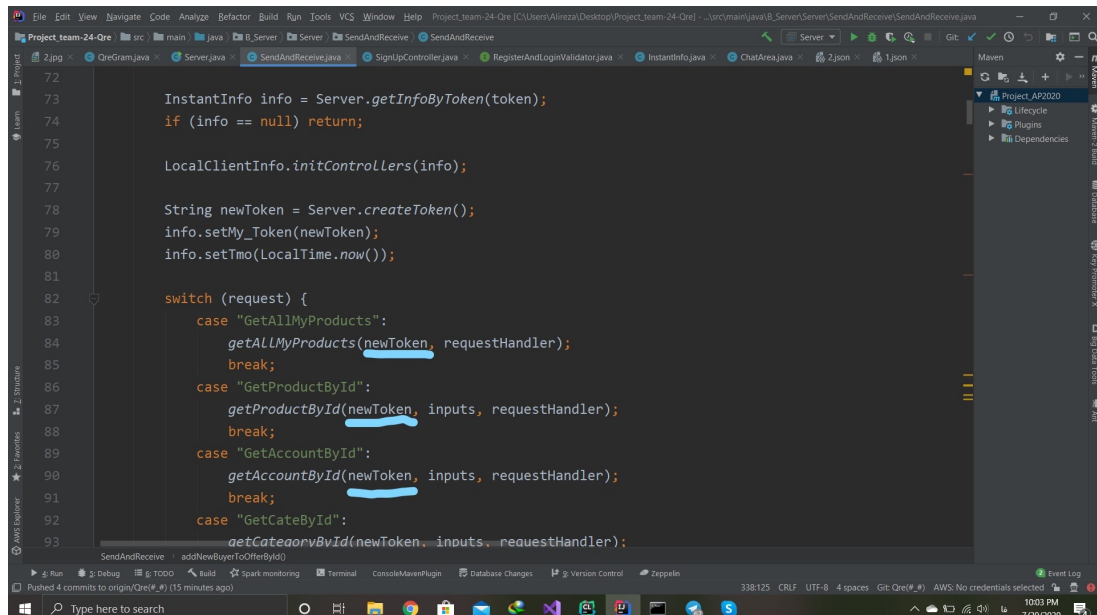


۱- اولین نوع حمله **reply attack** است. در این حمله پیام های بین سرور و کلاینت را حمله کننده مورد ذخیره کرده حجم زیادی از آن را دوباره می فرستد و باعث اختلال در برنامه می شود . برای جلوگیری از این اتفاق:

(۱) تعداد دستور ها را محدود کردیم تا حجم زیاد به سرور فرستاده نشود .

(۲) مطابق هر دستوری که به سرور فرستاده می شود به کلاینت یک توکن ارسال میشود و برای تکرار همان دستور قبلی باید آن توکن را داشته باشد بنابراین از ارسال پیام های تکراری توسط هکر جلوگیری میشود.

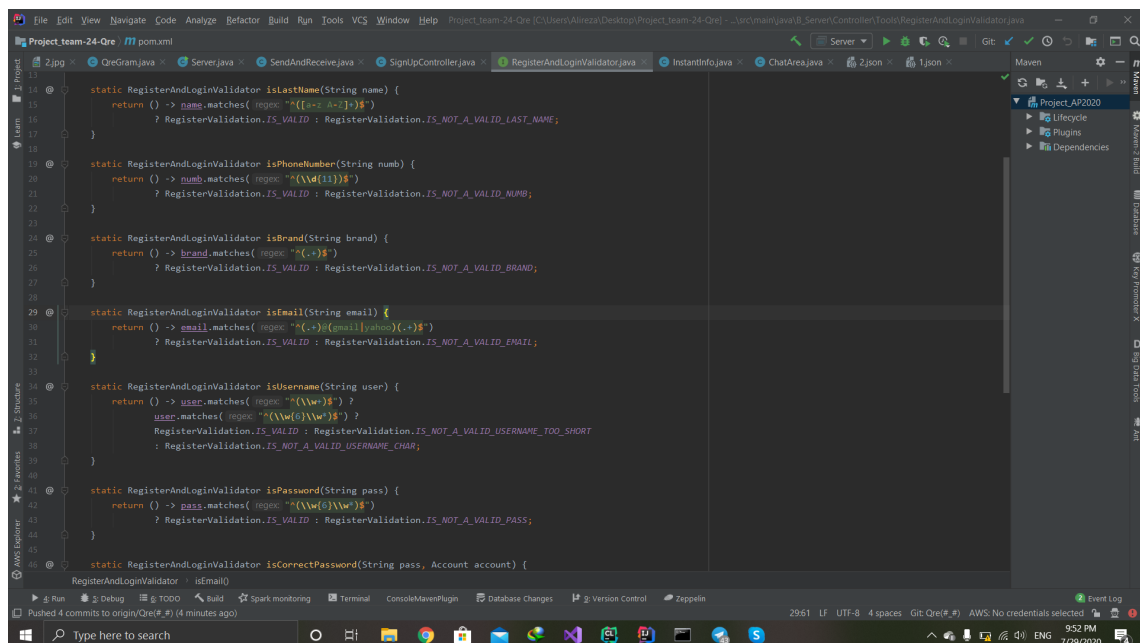


```
72 InstantInfo info = Server.getInfoByToken(token);
73 if (info == null) return;
74
75 LocalClientInfo.initControlllers(info);
76
77 String newToken = Server.createToken();
78 info.setMy_Token(newToken);
79 info.setTmo(LocalTime.now());
80
81
82 switch (request) {
83     case "GetAllMyProducts":
84         getAllMyProducts(newToken, requestHandler);
85         break;
86     case "GetProductById":
87         getProductById(newToken, inputs, requestHandler);
88         break;
89     case "GetAccountById":
90         getAccountById(newToken, inputs, requestHandler);
91         break;
92     case "GetCateById":
93         getCategoryById(newToken, inputs, requestHandler);
94 }
```

۲- حمله بعدی به شبکه **Improper Inputs** است که این حمله در وجود ضعف اعتبار سنجی ایجاد می شود. در این حمله اگر اعتبار سنجی قوی نداشته باشیم سبب متوقف شدن برنامه می شود و دسترسی به برنامه از بین می رود. حالت دیگر آن است که کاربر اجازه ی وارد کردن اسم با هر طولی را داشته باشد و باعث اشغال حافظه می شود. برای جلوگیری از این اتفاق :

(۱) برای این حمله برای قسمت های مختلف (ورودی های مهم) را مجهز به اعتبار سنجی کرده تا بخش منطق برنامه مرتب با **expection** برخورد نکند.

(۲) برای آنکه پیام با هر طولی ارسال نشود از **readUTF** و **rightUTF** استفاده کردیم این دو دارای محدودیت طولی هستند و به طور مثال کاربر برای ورودی نمیتواند حجم زیادی وارد کند. نمونه کد زده شده را مربوط به این حمله قرار دادیم.



۳- در برنامه ی ما دیتابیس از نوع SQL می باشد. در این نوع از دیتابیس هکر ها میتوانند دیتابیس ما را مورد حمله قرار بدهند و دیتابیس را به طور کامل پاک کنند. برای خنثی کردن این امر اجازه ی وارد کردن هر ورودی مانند ۱==۱ را از کاربر گرفتیم (زیرا این گزاره ی درست از شرط ها عبور می کند) بنابراین برای جلوگیری از این اتفاق ورودی های برنامه مورد بررسی قرار دادیم تا ورودی های نا مفهوم برای ما نظیر کدی که دستور پاک کردن در آن وجود دارد نتوانند وارد کنند.

۴- حمله بعدی Broken Authentication است که هکر ها به جای رمز عبور واقعی میتوانند با زدن تعداد رمز هایی ساده، وارد اکانت یک کاربر شوند. برای خنثی کردن این امر اجازه ی وارد کردن هر ورودی مانند ۱==۱ را از کاربر گرفتیم (زیرا این گزاره ی درست از شرط ها عبور می کند) و تنها کاربر اجازه دارد نام های کابری درست را وارد کند و برای رمز عبور نیز کراکتر های مشخصی را تعیین نمودیم. یعنی برای ست کردن پسورد هنگام ثبت نام باید یک پسورد قوی انتخاب شود.

۵- در برنامه ی ما اطلاعات ذخیره شده توسط کاربران اعم از مدیرها، پشتیبان ها ، فروشندگان و همچنین اطلاعات شخصی خریداران کاملاً باید محرمانه باشد، هر کس تنها به حساب خود و آن بخش از حافظه ی ذخیره شده ی مربوط به خود دسترسی دارد. (حافظه را رمز گذاری کردیم) بنابراین جلوی حمله ی Sensitive Data Exposure گرفته میشود. همچنین ممکن است پیام های رد و بدل شده توسط کلاینت و سرور شنود شود که به این منظور پیام ها را رمز گذاری کردیم که به آن دسترسی نداشته باشند. نمونه کد زده شده را مربوط به این حمله قرار دادیم.

```

75 private static class Blabber {
76
77     protected DataInputStream inputStream;
78     protected DataOutputStream outputStream;
79
80     public Blabber(@NotNull Socket socket) {...}
81
82     public void close() {...}
83
84     public void sendMessage(String message) throws IOException {
85         outputStream.writeUTF(CodeMessage.code(message));
86         outputStream.flush();
87     }
88
89     public String receiveMessage() throws IOException {
90         return CodeMessage.decode(inputStream.readUTF());
91     }
92 }

```

- ۶- حمله دیگری که برنامه را مورد تهدید قرار می دهد آن است که به تعداد زیاد نام کاربری و رمز عبور را وارد کنند تا آن را حدس زده یا سیستم برنامه را مختل کنند. برای رفع این حمله یک شمارنده برای تعداد تلاش های ناموفق برای یک ایپی قراردادیم تا از میزان محدودیت تعیین شده که بیش تر شد آن ایپی در لیستس جداگانه قرار بگیرد و طی زمانی مشخص دوباره اجازه ورود به آن ایپی داده شود و این گونه جلوی حمله ی **Brute force attack** را گرفتیم. (بیش از ۱۰ درخواست اشتباه از آن کلاینت از سرور قطع میشود)
- ۷- یکی از حمله هایی که با آن روبه رو هستیم **Denial of Service (DoS)** است. در این حمله تعداد زیادی درخواست توسط یک کاربر داده می شود و سیستم مختل می گردد. برای رفع این مشکل در زمان ۱۰ ثانیه یک ایپی بیش از ۱۰۰ درخواست داشته باشد، درون یک لیست قرار می گیرد و به طور موقت اتصال آن از برنامه قطع می شود.
- نمونه کد زده شده را مربوط به این حمله قرار دادیم.

```

70 private static ThreadLocal<Integer> numOfMessageInMin = new ThreadLocal<>();
71 private static ThreadLocal<CountDownLatch> latch = new ThreadLocal<>();
72
73 static {
74     resetNumOfMess();
75     // numOfMessageInMin: At the time of sending the answer, this has increase
76     resetLatch();
77     ScheduledExecutorService resetService = Executors.newSingleThreadScheduledExecutor();
78     resetService.scheduleAtFixedRate(SendAndReceive::resetNumOfMess, initialDelay: 0, period: 1, TimeUnit.MINUTES);
79 }
80 private static void resetNumOfMess() {
81     numOfMessageInMin.set(0);
82     latch.get().countDown();
83 }
84 private static void resetLatch() { latch.set(new CountDownLatch(1)); }
85
86 public static void messageAnalyser(@NotNull String token, String request, List<String> inputs, RequestHandler requestHandler) {
87
88     if (numOfMessageInMin.get() > 100) {
89         latch.wait();
90         resetLatch();
91     }
92 }

```

