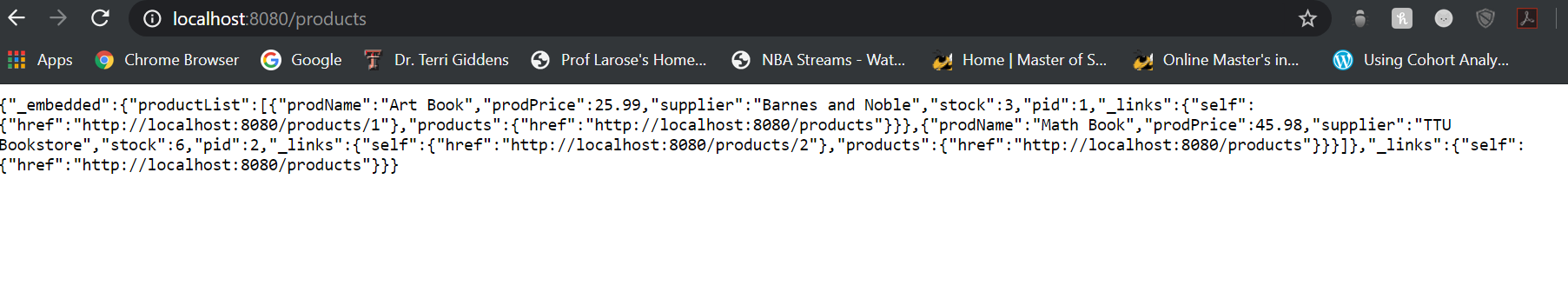
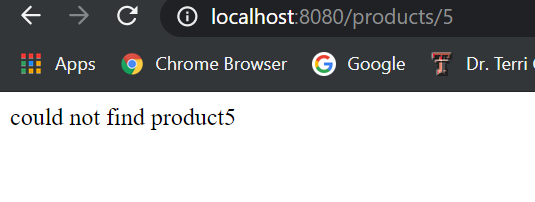
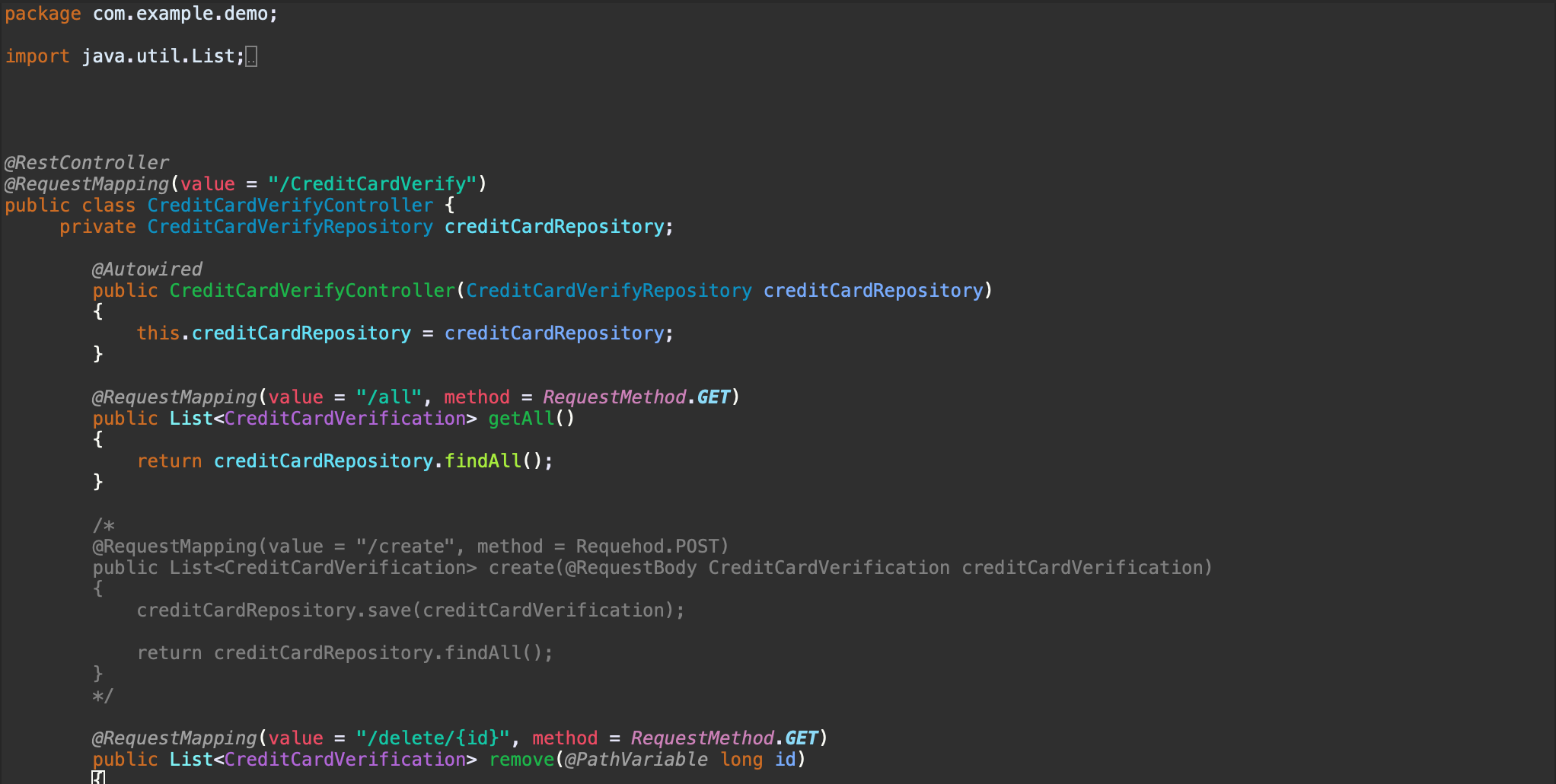
For our Shopping Cart project, we decided to implement our ‘Products’ class and ‘Credit Card Verification’ class as services. To do this, we used Spring Boot. Spring Boot makes developing production ready applications easy and straightforward. It is convention based, so there is no need for XML or code configuration most of the time. Using Spring Boot we can create self hosted web applications and execute them as a java JAR application.

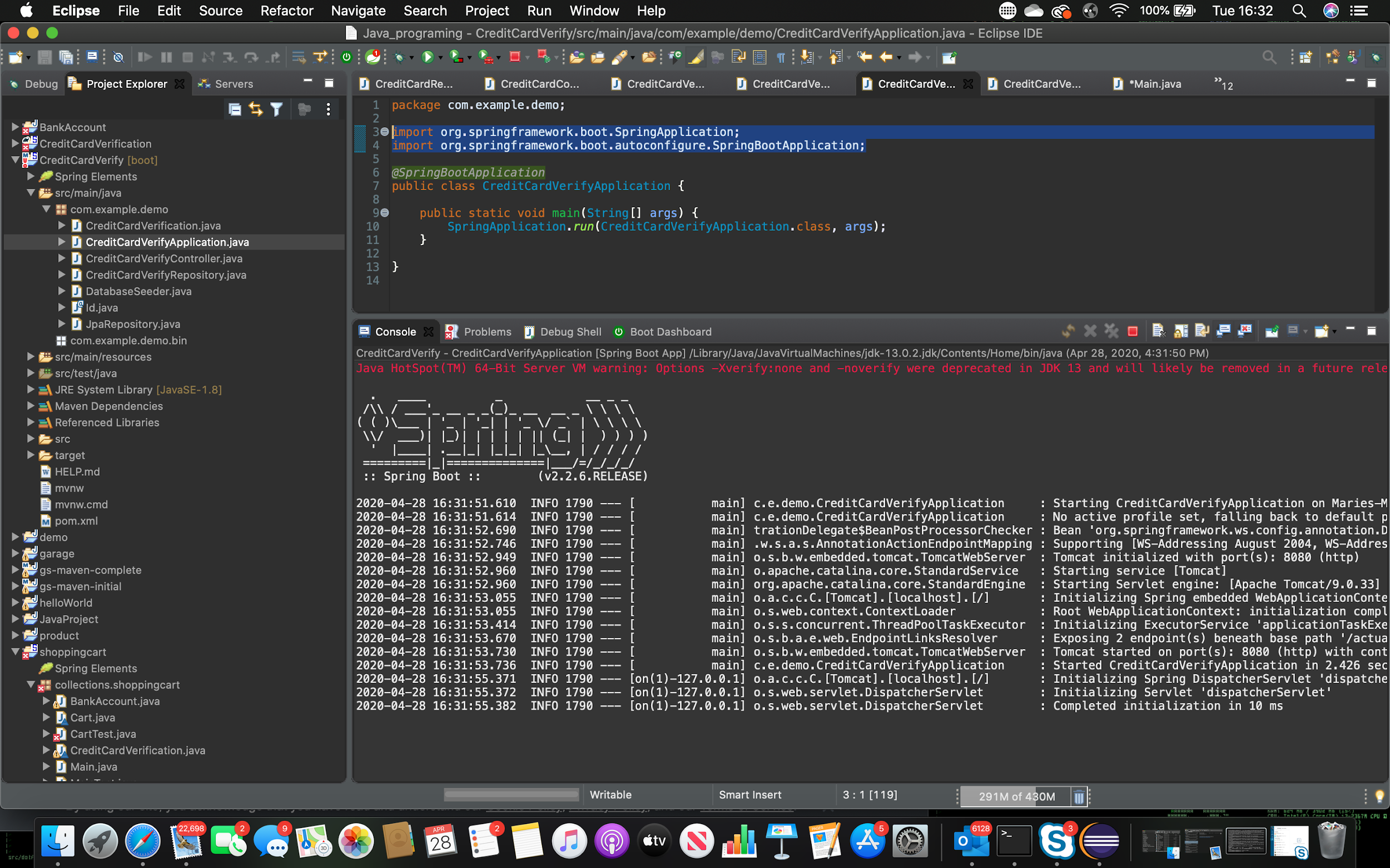
For our Product service, we start with a simple product.java to establish a constructor and @Entity that can be used for JPA storage. Next we created a products repository. This code contains a JPA Repository that will allow for creating new instances, updating new instances, deleting, and finding. Now we have to create a @SpringBootApplication that will fire up a servlet container. For our product service, we also need to load the database with some products, this is done with LoadDatabase.java. This code utilizes Spring Boot @Beans and Lombok configuration to log and create new products. We must use a product Controller program that will call the usual methods the product class had. We have to import tons of Spring frameworks to control mapping that corresponds to HTTP GET, POST, PUT etc. We also included an exception class to throw a HTTP404 error.







Our application includes a static home page that accepts HTTP GET requests at a designated localhost. The approach to build this web site was handled in our Controller section with different parameters. The @GetMapping annotation ensured that HTTP GEt requests to our view were mapped to our method. The @RequestBody indicated that a method parameter needed to be bound to the body of the web request and this body is now passed through an HTTpMessageConverter to resolve the method argument depending on the type of the request. The @RequestMethod just indicated the action to be performed.



For our credit card verification service the first method we used was creating a Java model class called CreditCardVerification.java which adds the CreditCardVerification class in order to set it up for the implementation of the REST Controllers in the CreditCardController.java file. In this CreditCardController.java we create a @RestController with a List with a constructor and a @RequestMapping to navigate within the local host using the GET method which can also be used to import other control mappings. (localhost:8080/CreditCardVerify/all) Next we created a repository interface that extends the JpaRepository interface to allow the creation of new instances, updating, deleting and finding. Once that is done we have to create a @SpringBootApplication annotation in order to run the program.

Once that is done we mark the CreditCardVerification class in the .java file as an entity (@Entity) then add an ID property (@Id) which will be generated sequentially using the generation type sequence (@GeneratedValue). Once done creating the repository we need to initialize the database using CommandLineRunner interface in the DatabaseSeeder.java file.

Once the application starts up and the contents are created, all the classes that implement CommandLineRunner are executed which makes it perfect for initializing the database which we can implement within the DataBaseSeeder.

The final application once ran on a localhost creates a webpage that can be used to load the database we created and see the data marked with ids (@Id) which can also be deleted using the remove(@PathVariable). The GET method was used to request the view of the webpage. Doing this the application ran smoothly and we were able to view the output.

