1. Problem Sets

For your lab submission, take screenshots or cut-n-paste your solutions into a document or zip archive your generated solutions which you will submit through webcourses

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | Using Node, Express and Massive create the following HTTP API endpoints serving the following resources as JSON documents   |  |  | | --- | --- | | GET /users | List all users email and sex in order of most recently created. Do not include password hash in your output | | GET /users/:id | Show above details of the specified user | | GET /products | List all products in ascending order of price | | GET /products/:id | Show details of the specified products | | GET /purchases | List purchase items to include the receiver’s name and, address, the purchaser’s email address and the price, quantity and delivery status of the purchased item. Order by price in descending order |   Test each of these endpoints serves the expected data and briefly show how you did this |  |
| **2** | Building on your solution to part 1 for the API to the products resource from the pgguide database, extend the product indexing endpoint to allow the filtering of products by name as follows   |  | | --- | | GET /products[?name=*string*] |   For your solution you should implement the query (badly) in such a way as to allow an attacker to inject arbitrary SQL code into the query execution. Show, using your badly implemented approach, how an attacker can craft a query string to allow the deletion of a product from the products table.  For convenience, you can continue to use MassiveJS to interface with the database.      Part2 |  |
| **3** | Provide two solutions to eliminate the security hole in your approach from the previous section as follows:   * Using a parameterised query * Using a stored procedure using SQL or PLPGSQL whichever you prefer   Explicitly show that the injection attack is not now possible for each of your solutions  Again, you can just use MassiveJS as your database interface library here too.  Part1      Part2 |  |
| **4** | Create a brand new Express project using the Sequelize ORM. Install and configure Sequelize and wire it up to the pgguide database.. Verify that you have basic connectivity before proceeding.  Create Sequalize migrations for the pgguide sample database  Ensure that the appropriate associations and referential integrity checking are set up in your models |  |
| **5** | Use your models and Javascript code to populate the database with some additional test data for all of the models above  I only changed one section of the code which was the sequelize.sync() |  |
| **6** | Reimplement the RESTful API using Sequelize and Express for your system. Your API should support the following CRUD operations as follows, returning JSON responses   |  |  | | --- | --- | | GET /products[?name=string] | List all products | | GET /products/:id | Show details of the specified products | | POST /products | Create a new product instance | | PUT /products/:id | Update an existing product  Before updating number 21:    After updating number 21 New Book -> Updated Book    Code: | | DELETE /products/:id | Remove an existing product |   Show test cases for each of the API endpoint REST operations |  |