```
        public class ExtractParameterRequest
        {
            public string Data { get; set; }
            public ProductForUpdateDto Product { get; set; } = new ProductForUpdateDto();
        }

        [HttpPost("extract-parameter")]
        public async Task<IActionResult> ExtractParameter([FromBody] ExtractParameterRequest request)
        {

            if (request == null || string.IsNullOrEmpty(request.Data))
            {
                return BadRequest("Request body or data is required.");
            }

            string data = request.Data;

            // Split the data by spaces
            string[] parts = data.Split(' ', StringSplitOptions.RemoveEmptyEntries);

            if (parts.Length < 18) // Must be at least 19 parts: 1 + 1 + 16 (GUID) + 1
            {
                return BadRequest("Data format is incorrect.");
            }

            // Parse inOrOut (first value)
            bool inOrOut;
            if (parts[0] == "1")
                inOrOut = true;
            else if (parts[0] == "0")
                inOrOut = false;
            else
                return BadRequest("Invalid inOrOut parameter. Expected '1' or '0'.");

            // Parse warehouse (second value, should be a single character)
            if (parts[1].Length != 1)
            {
                return BadRequest("Invalid warehouse parameter. Expected a single character.");
            }
            char warehouse = parts[1][0];

            // Extract the 16-byte serial number (from index 2 to index 17)
            string serialHexString = string.Join("", parts.Skip(2).Take(16)); // Remove spaces and join

            if (serialHexString.Length != 32) // A valid GUID must have 32 hex digits
            {
                return BadRequest("Invalid serial number length. Expected 16 hexadecimal values.");
            }

            // Convert hex string to byte array
            byte[] serialBytes = new byte[16];
            for (int i = 0; i < 16; i++)
            {
                if (!byte.TryParse(parts[i + 2], System.Globalization.NumberStyles.HexNumber, null, out
serialBytes[i]))
                {
                    return BadRequest($"Invalid hex value at position {i + 2}: {parts[i + 2]}");
                }
            }

            // Convert byte array to GUID by ensuring correct endianess
            Guid serialNumber = new Guid(
                new byte[]
                {
                    serialBytes[3], serialBytes[2], serialBytes[1], serialBytes[0], // Data1 (4 bytes,
reversed)
                    serialBytes[5], serialBytes[4], // Data2 (2 bytes, reversed)
                    serialBytes[7], serialBytes[6], // Data3 (2 bytes, reversed)
                    serialBytes[8], serialBytes[9], serialBytes[10], serialBytes[11], // Data4 (big-
endian, unchanged)
                    serialBytes[12], serialBytes[13], serialBytes[14], serialBytes[15]  // Data5 (big-
endian, unchanged)
                }
            );
            var productEntity = await _repository.Product.GetProductBySerialNumberAsync(serialNumber,
trackChanges: false);
            try
            {
                return await AssignShelfToProduct(inOrOut, warehouse, serialNumber,
_mapper.Map(request.Product, productEntity));
            }
            catch (Exception ex)
            {
                _logger.LogError($"Error in ExtractParam calling AssignShelfToProduct: {ex.Message}");
                return StatusCode(500, "Internal server error.");
            }
        }
```