Data-driven systems

Exam

January 11, 2024

NAME: _____

CAPITALIZED, -1p if you fail

NEPTUN CODE: _____

CAPITALIZED, -1p if you fail

Exercise 1 (9 points)	
Exercise 2 (10 points)	
Exercise 3 (6 points)	
Exercise 4 (9 points)	
Exercise 5 (8 points)	
Exercise 6 (8 points)	
Σ (50 points)	
Grade	

You have **60 minutes** to complete the following exercises. Solve the exercises on this test sheet, use pen! If you need more paper, ask for it! Make sure your **hand writing is readable** and your work is clear-cut. Be clear and **precise** when explaining your answers! Do not use inefficient solutions unless the exercise explicitly allows it! Please hand in the sheet folded in the middle!

1. Consider the JSON document below. It is a sample for a MongoDB collection storing entities of this kind. Write C# code that solves the following tasks. Each task should be solved with a single query/statement. The MongoDB collection is available in a variable `collection` of type `IMongoCollection<T>`. (Note: for filtering and updating you may use the `Builders<T>.Filter` and `Builders<T>.Update` factory helpers.) (3*3p)

Sample document, Product class:

```
{
    "name": "Apple",
    "price": 120,
    "categories": ["fruits", "on sale"]
}
```

a) List the items cheaper than 1000

```
var items = collection.Find(i =>  i.price < 1000).ToList();
Foreach(var i in list)
        Console.WriteLine(i.Name);
```

b) Double the price of all items with name "Apple". You should use atomic update! (Tip: the multiplication operator is called "mul.")

```
collection.UpdateMany( filter: p => p.name == "apple", update:
Builders<Product>.Update.Mul(p => p.Price, 2));
```

c) Delete category "on sale" from each item. You must use filtering too! (Tip: use the operator called "Pull.")

2. Decide whether the following statements are true! Mark them with T and F letters! -1 point for an incorrect answer, 0 points for a blank answer. The sum of the subtask cannot be less than 0. (10p)

Stored procedures in MongoDB are written in JSON.	F
In a three-tier architecture ORM is the responsibility and concern of the data access layer.	T
Logical design optimization uses equivalent transformations.	T
In Microsoft SQL Server every query runs within a transaction, even if we do not define one explicitly.	F
Th purpose of query optimization is the effective utilization of the hardware resources.	T
The nested loop join algorithm can be used on tables of arbitrary size.	T
The SOAP protocol also supports JSON format.	F
REST is a standard.	F
With JPQL, SELECT, UPDATE and DELETE queries can be written as well.	T
You can store client-specific state in singleton session beans.	F

3. Answer the following questions. Explain your answer and support it with arguments! (3+3p)

a) If a database is unable to generate unique values for primary keys, where should this action be performed in the three-tier architecture? Explain your argument.

b) Imagine a three-layered architecture. Searching and filtering in a dataset can be performed solely by the presentation layer (fetch all data then perform the filtering on-demand). Or the filtering can be performed in the data access layer. Choose either alternative and explain why it is a good idea, and when is it recommended to do so?

4. Answer the following REST related questions!

a) In ASP.NET Core environment, a service is being developed that manages a recipe registry with a REST API. The task is to implement a WebApi controller function that returns the data of those recipes in JSON format, whose name contains a given character sequence (the character sequence is given in the request). Follow the recommendations when solving the problem! It can be assumed that a Recipe model class already exists.

Provide the definition of the WebAPI controller function! You can leave the body of the function empty, provide the definition of the function (there is no need to provide the definition of the class).

```
[HttpGet("{name}")]
GET http://example.com/api/recipes/name
```

b) In the context of the above task provide a sample http request for creating an Recipe resource! Only provide the https verb and the url (there is no need to provide the body). (1p)

```
Post http://example.com/api/recipes
```

c) When designing a REST API, which http verb is the right solution for partially overwriting the properties of an existing resource? (1p)

```
Patch
```

d) When designing a REST API, which http verb is the right solution for overwriting an existing resource and all its properties? (1p)

```
Put
```

e) Show an example (URL only) of querying hierarchical data! For example, querying all comments belonging to a specific blog post (the blog post is given by its ID, which is a number). (1p)

```
GET http://example.com/api/blog-post/123/comments
```

f) A REST API operation that creates a new resource should return the URL of the newly created resource. How should this be done? (1p)

g) Specify two things provided by the Swagger/Open API specification (what it can provide for developers)! (2p)

```
Available endpoints (/users) and operations on each endpoint(GET/users, POST/users)
Operation parameters input and output for each operation Authentication methods
Contact information, license, terms of use and other information
```

5. Answer the following questions related to SOAP/REST/GraphQL and dependency injection! (6p)

a) Give a one-line example where REST is used - with a justifiable reason - in a non-REST(ful) way! (2p)

b) (Web) services have four important basic characteristics/criteria. Name three of them! (2p)

```
Communication
Operation
Interoperability
Autonomous units
```

c) Specify a GraphQL query that retrieves for each user the user's phone number (phone) and list of addresses (addresses). For each address, retrieve the zipcode (zipcode) and city name (cityname). It is important that you retrieve all the above information in a single query. (2p)

```
query {
        user(id: “123”) {
                phone
                addresses {
                        zipcode
                        cityname
                }
        }
}
```

d) Briefly describe what it means when a dependency is transiently registered in the ASP.NET Core dependency injection container. (2p)

It means that a new instance of the dependency is created each time it is requested from the service container.

6. The Accounts and Clients of a bank are stored with JPA entities. One client can have multiple accounts, but each account has one owner.
Fields of an Account: long id, LocalDate created (the day when it was created), double balance
Fields of a Client: long id, String name, String address
In both classes, the id field is the primary key, and it is auto-generated. The relationship between the two entities is bidirectional. Write a Spring bean with a method that accepts two arguments: the id of a client and an initial balance. The method creates a new account, owned by given client, with the given balance, with the created field set as today. It throws an exception if the client with the given id does not exist. If you find it useful, you can write a Spring Data Repository interface and call it from the method.

(8p)

```
@Entity
@Setter
@Getter
public class Account
{
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;

	private LocalDate created;
	private double balance;

	@ManyToOne
	@JoinColumn(name = “client_id”)
	private Client owner;
}

@Entity
@Setter
@Getter
public class Client
{
	@Id
	@GenerateValue(strategy = GenerationType.AUTO)
	private Long id;

	private String name;
	private String address;

	@OneToMany(Mappedby = “owner”)
	private Collection<Account> accounts;
}
```

```
public interface AccountRepository extends JpaRepository<Account, Long> { }

Public interface ClientRepository extends JpaRepository<Client, Long> { }

@Service
public class BankService {
	private final ClientRepository clientRepository;
	private final AccountRepository accountRepository;

	@Autowired
	public BankService(ClientRepository clientRepository, AccountRepository accountRepository) {
	this.clientRepository = clientRepository;
	this.accountRepository = accountRepository;
	}

	@Transactional
	public void createAccountForClient(Long clientId, double initialBalance) {
		Optional<Client> optionalClient = clientRepository.findById(clientId);
		If(optionalClient.isPresent()) {
			Client client = optionalClient.get();

			Account newAccount = new Account();
			newAccount.setCreated(LocalDate.now());
			newAccount.setBalance(initialBalance);
			newAccount.setOwner(client);

			accountRepository.save(newAccount);
		} else {
			throw new RuntimeException(“Client with id ” + ClientID + “does not exist.”);
		}
	}
}
```