Building Systems in Rust - Exam Questions

You will select ONE of these questions at the beginning of the exam. If you would prefer a different question you can have ONE go at picking another one - 'stick or twist' - this is a once only swap (you can't keep swapping!).

The idea of the questions is to promote a 15 minute discussion of the topic with you demonstrating your understanding by showing relevant parts of your project and answering related questions.

The questions are in English but you should answer them in either English or Danish (by law you may also answer in Swedish or Norwegian.. please don't do that!)

1. Error Handling and Enums: How have you used enums to manage errors in your project? Discuss any patterns you implemented, such as the Result or Option types, and how they contributed to your application's robustness (Option data type [Some() and None()] and the Result type [OK() or Err()])

2. Ownership and Borrowing Concepts: Explain the concept of ownership and borrowing in Rust. How did these features influence the design and functionality of your project? Provide examples of how managing ownership and borrowing helped improve your code's safety or performance. What are some of the common problems you encountered using the Rust Ownership and Borrowing model?

3. Strings and String Handling in Rust: In Rust, the handling of strings is different to languages like Java C# - especially about mutability and ownership. Discuss how you managed strings in your project. What challenges did you face with String and &str types? Provide examples of where you optimised string usage for performance or memory efficiency (you could discuss the use of &'static str or [A]rc<str> where lots of cloning is needed)

4. Polymorphism: Traits and Enums in Rust allow for polymorphism. Can you discuss a place in your project where you implemented polymorphic behavior? How did this design choice benefit your project? How is the problem of inheritance avoided in Rust? (think about composition - use of structs and impl functions)

5. Memory Management: Memory management is handled differently in Rust compared to other languages like C++, C# or Java. Describe how Rust's memory management principles affected the way you structured your project. Include examples of how you used Box, Rc, or RefCell in managing heap data.

6. Pattern Matching and Control Flow: Pattern matching is a powerful feature in Rust. How did you use pattern matching to simplify complex control flow in your project? Provide examples of how this was used to help your code be more efficient and/or readable. What do you think of the pattern matching instead of using switch type statements?

7. Structs and Data Organization: Structs are fundamental for organizing data in Rust. How did you decide when to use structs versus tuples or arrays in your project? Discuss any instances where you used the derive attribute with structs or used impl to add behaviour.

8. Module System and Code Organization: Rust's module system aids in organizing large codebases. How did you utilize this system to structure your project's code? Explain your use of pub, mod, and other visibility qualifiers to manage encapsulation and modularity.

9. Concurrency in Rust: Rust provides several tools to handle concurrency, such as threads, Arc, and Mutex. Discuss how you utilized these tools in your project. What challenges did you encounter while implementing concurrency, and how did Rust's safety guarantees affect your solutions?