

Matriculation number: _____

Good luck!
Robert Schäfer

[illegible]

1. Project Management

Mentoring

Knowledge transfer

Code Reviews

Testing

Continuous Integration

1

(a) Complete the following sentence: The motto of our course is “Le__n and S__e”

1

(b) What is the bus factor? Write a definition.

.....

.....

.....

1

(c) What is better: A high or a low bus factor?

.....

2

(d) In the lecture we covered five benefits of software testing. Please fill in at least two of them in figure 1 and explain them by example: What could go wrong without software testing with respect to that benefit? Why does something work better with software testing?

.....

.....

.....

.....

.....

.....

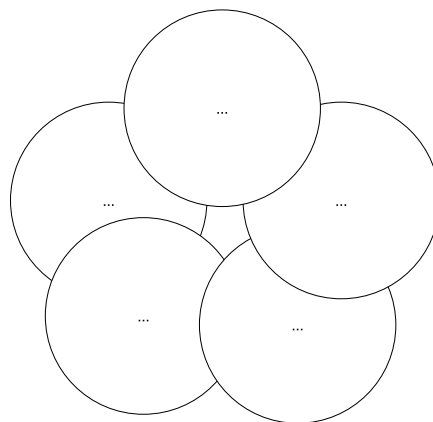


Figure 1: Five benefits of software testing

Name:

Systems-Development and Frameworks

3

(e)

There will be at least three more questions here

2. Git

git rebase

git clean

git reflog

git reset

git blame

git remote

git checkout <REFERENCE> - path/to/file

Difference between working tree and index

How to write better commit messages

1

(a) What data is hashed in order to generate a git commit id? Enumerate at least three attributes.

.....

.....

2

(b) Is `git reset <ID> --hard` reversible? If yes, how can you reverse it? If no, please explain why.

.....

.....

.....

.....

2

(c) Is `git clean --force` reversible? If yes, how can you reverse it? If no, please explain why.

.....

.....

.....

.....

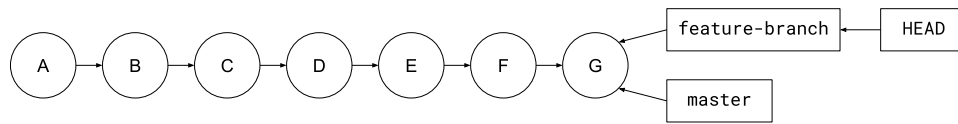
(d)

We will do the following exercise the other way around: Instead of you writing down the git commands that produce a given history, I will give you some commands and your task is to draw the resulting git history.

Consider the following scenario: You forgot to create a feature branch and committed on branch `master` by accident.

Now you want to create a pull request. In order to do that you create a new branch with:

```
git checkout -b feature-branch
```

Figure 2: After `git checkout -b feature-branch`

You can see your current git history in figure 2.

1

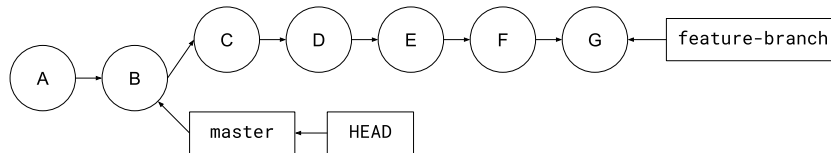
- i. What are the commands that produce a git history as shown in figure 3?

.....

.....

.....

.....

Figure 3: Remove commits from `master`

1

- ii. Consider your git history looks as shown in figure 3. On wich commit do you need to rebase in order to produce a git rebase log as shown in figure 5 and finally a git history as shown in figure 4?

```

git checkout feature-branch
git rebase --interactive ....

```

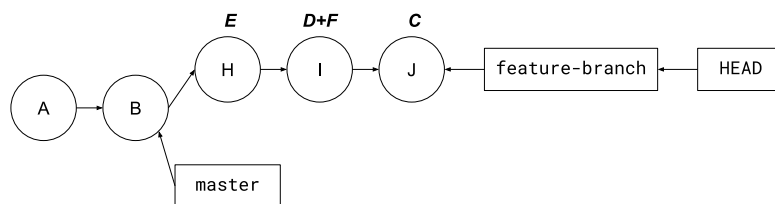


Figure 4: Commits contain changes of the commits on top of the respective nodes

2

- iii. How do you need to change the rebase log in figure 5 to produce a git history as shown in figure 4?

```
pick C .....  
pick D .....  
pick E .....  
pick F .....  
pick G .....
```

Figure 5: Rebase log of your editor

3. Functional Programming

- 1 (a) What are higher order functions?

.....

- 1 (b) Give an example of a higher order function in JavaScript:

(c)

Array.prototype.forEach

Array.prototype.map

Array.prototype.find

Array.prototype.filter

Array.prototype.reduce

The following code examples have unnecessary temporary variables. How can you refactor the code examples and eliminate these temporary variables?

```
function findNextId (){
  let lastId = 0;
  for (i = 0; i < data.todos.length; i++) {
    if (data.todos[i].id > lastId) {
      lastId = data.todos[i].id;
    }
  }
  lastId += 1;
  return lastId;
}
```

```
function filterTodos(mok, userAuth) {
  var retArray = []

  for (var i = 0; i < mok.length; i++) {
```

```
        if (mok[i].userAuth == userAuth) {  
            retArray.push(mok[i])  
        }  
    }  
    return retArray  
}
```



Add one more code example here

3

i. You will get a point per code example if you can name the right method which you could apply.

3

ii. You will get a point per code example if you can write down the correct (or almost correct) refactored source code.

4. GraphQL and Apollo-Server

- 1 (a) What are the problems that GraphQL solves in comparison with REST? Explain how exactly GraphQL solves these problems.

.....

- 2 (b)

Few groups used type resolvers in their assignments. Understanding the difference between a top level resolver and a type resolver is crucial. You should also know the use cases, for what you should use a top level resolver and when to use a type resolver.

Look at the implementation in figure 7. When we send the graphql query in figure 6 to the server, what is the response of the server?

```
query {
  hello(name: "Again")
}
```

Figure 6: Graphql query



- (c) What gets written to the terminal of the backend? Write down the output of the `console.log` statements. You will get points for the following:

- 1 i. Correct selection of traversed calls
- 1 ii. Correct number of calls
- 1 iii. Correct order of calls
- 1 iv. Correct value of `parent`
- 1 v. Correct value of `args`

Name:

Systems-Development and Frameworks

A large, empty rectangular box with a thin black border, occupying the upper half of the page. It is intended for a drawing or a detailed answer.

```
1  const { applyMiddleware } = require('graphql-middleware')
2  const { ApolloServer, gql } = require('apollo-server')
3  const { makeExecutableSchema } = require('graphql-tools')
4
5  const typeDefs = `
6  type Query {
7    hello(name: String!): String!
8    bye(name: String!): String!
9  }
10 `
11
12 const logInput = async (resolve, parent, args, context, info) => {
13   console.log(`1. logInput: ${JSON.stringify(args)}`)
14   const result = await resolve(parent, args, context, info)
15   console.log(`5. logInput`)
16   return result
17 }
18
19 const logResult = async (resolve, parent, args, context, info) => {
20   console.log(`2. logResult`)
21   const result = await resolve(parent, args, context, info)
22   console.log(`4. logResult: ${JSON.stringify(result)}`)
23   return result
24 }
25
26 const middlewares = [logInput, logResult]
27
28 const resolvers = {
29   Query: {
30     hello: (parent, args, context, info) => {
31       console.log(`3. resolver: hello`)
32       return `Hello ${args.name ? args.name : 'world'}!`
33     },
34     bye: (parent, args, context, info) => {
35       console.log(`3. resolver: bye`)
36       return `Bye ${args.name ? args.name : 'world'}!`
37     },
38   },
39 }
40
41 let schema = makeExecutableSchema({ typeDefs, resolvers })
42 schema = applyMiddleware(schema, ...middlewares)
43 const server = new ApolloServer({ schema });
44 server.listen().then(({ url }) => {
45   console.log(`  Server ready at ${url}`);
46 });
```

Figure 7: Apollo server implementation

5. Neo4J

Behaviour of MERGE

Pattern with multiple hops (e.g. friend of a friend)

How to select nodes without a particular relationship

OPTIONAL MATCH

(a) Consider the cypher statements:

```
CREATE(romeo:Person {name: 'Romeo'})  
CREATE(juliet:Person {name: 'Juliet'})
```

Figure 8 shows how the graph looks initially.



Figure 8: Romeo and Juliet at the beginning of the story

When you run the following statement:

```
MERGE(:Person {name: 'Romeo'})-[:LOVES]->(:Person {name: 'Juliet'})
```

The graph looks like shown in figure 9.

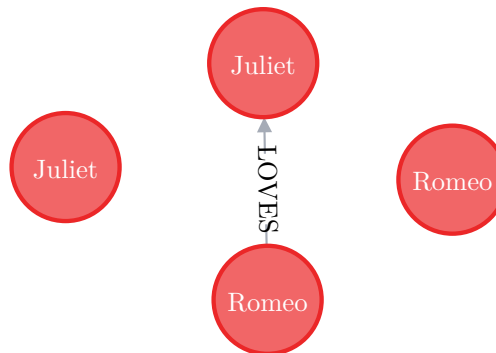


Figure 9: Oops, we have two Romeos and two Juliets now

1

i. How can you fix the cypher statement so that we don't get duplicate nodes?

2

ii. What has happened, why have duplicate records been created at all? Can you explain this?

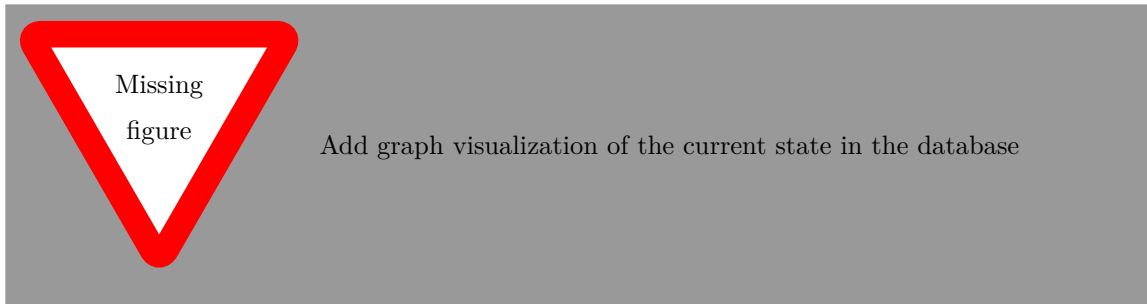
.....

.....

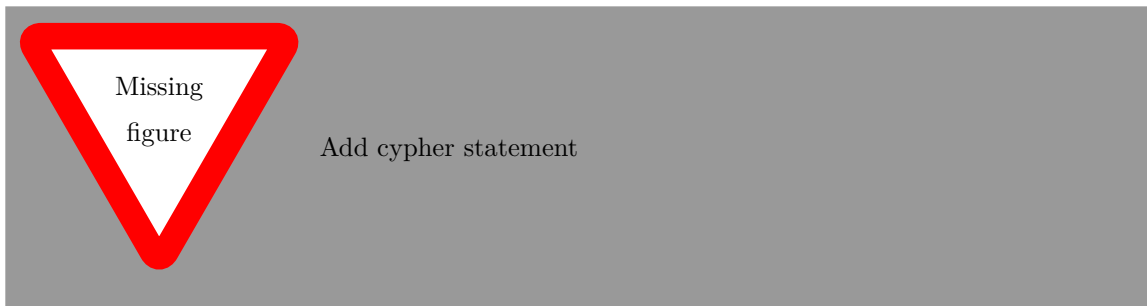
.....

.....

(b) Here is the visualization of a graph.



When you send the following cypher statement to the database:



3

i. How does the result set look like?

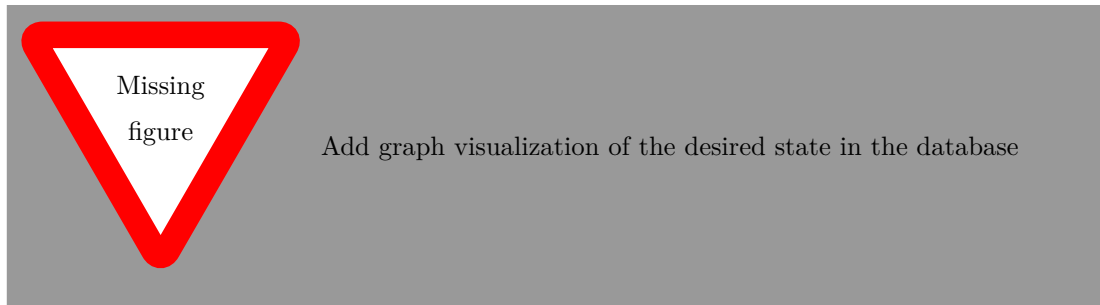
3

ii. How many records are in the result set?

.....

3

iii. Here is a graph visualization of a desired state in the database.



Write down a single cypher statement that produces the desired result.

.....

6. VueJS and NuxtJS

VueJS: component composition pattern

VueJS: data bindings and events

VueJS: computed properties

VueJS: watchers

Nuxt: asyncData

Nuxt: page components and routing

SSR: JS globals e.g. 'document' or 'window'

2

- (a) What are the pros and cons of a single page application and client-side-rendering in comparison with a traditional multi-page and server-side-rendered application?

.....

.....

.....

.....

1

- (b) What is isomorphic code?

.....

.....

2

- (c) NuxtJS provides server-side-rendering for VueJS. What is the motivation to do both server-side-rendering and client-side-rendering?

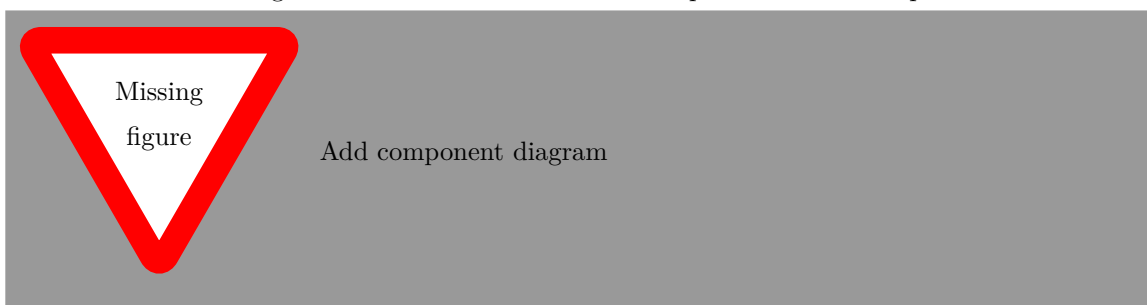
.....

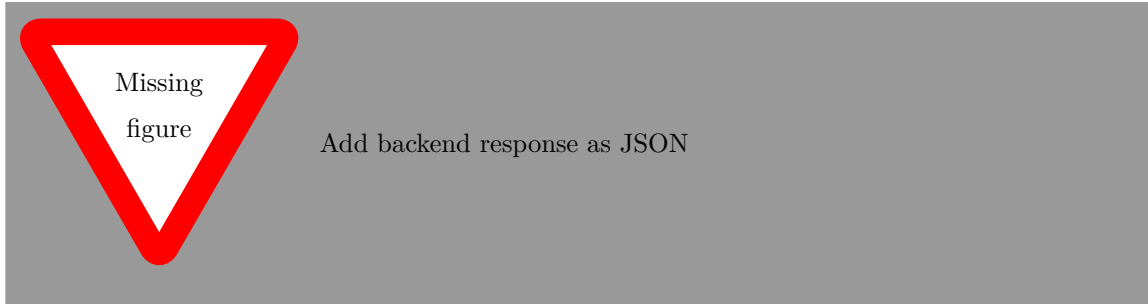
.....

.....

.....

- (d) Have a look at this diagram which shows the different components and its responsibilities.





3

i. What is the minimal data that needs to get passed as property down to the subcomponents respectively? Write your answer as an annotation in the diagram.

3

ii. What are the events that should get emitted from the subcomponents to the parent components respectively? Write your answer as an annotation in the diagram.

7. Fullstack testing and Requirements Engineering

Benefits and drawbacks of fullstack testing

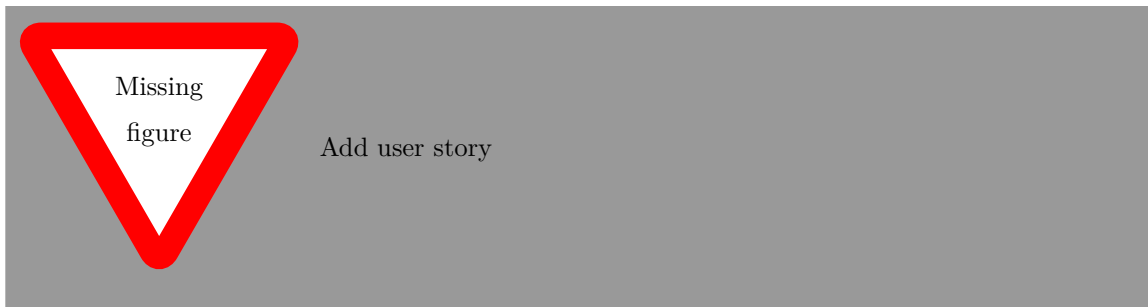
Common pitfalls of fullstack testing, e.g. timeouts and checking non-existing HTML elements

User story template - who, what, why?

Cucumber scenarios and readability

2

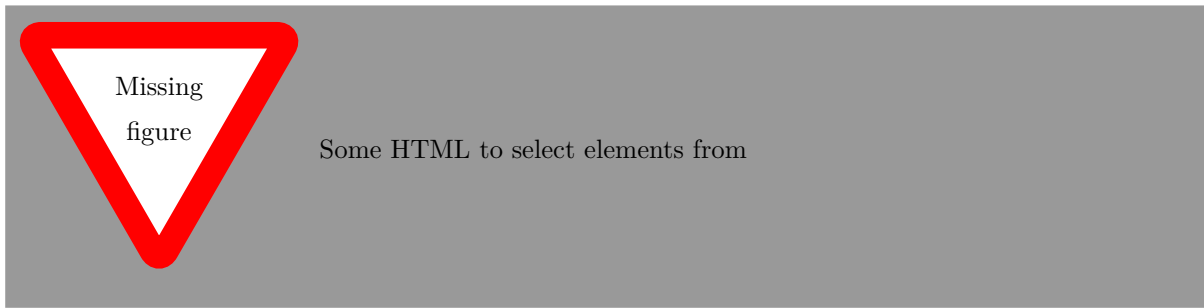
(a) What are the issues with the following user story?



2

(b) Write down a cucumber scenario which could be a valid acceptance criteria for the user story.

8. CSS



3

(a) Which element(s) are selected with this CSS selector?

Three different CSS selectors here

1

(b) Write down a CSS selector to select all input fields in the above example

9. Code Review

I will use a VueJS component or an apollo server test as a code example for the code review

10

- (a) Give a code review of the following pull request. Find and annotate issues in the code. You will get a point only if you can explain what the problem is. You can get another point if you can give a suggestion how to fix the problem. There are about 20 issues in the code example. It is enough if you can find 10 issues and explain the problem. Alternatively you can also find 5 issues, explain the problem **and** write down the source code as a suggestion to fix it. Anything in between is also possible.

Non-atomic test cases

Mixing async/await with then/catch

Not sanitizing strings and having a cypher injection vulnerability

C-style for loop

Writing 'should' in test case descriptions

Not closing sessions

Multiline string without backticks

Using index literals or keys instead of array/object destructuring

Merge objects with ... spread operator

Secrets (e.g. API keys or SSH keys) checked into source code

Not waiting for a promise to finish - tests pass before even checking a result

Nested if-clauses instead of guard clauses

Expose user listing in login resolver - slight security issue

Clients cannot update their cache because types have no ID in type definitions

Not using hard-coded values for expectations - danger of false negatives