# Software engineering semester project

# QuizApp

Student: Indre Bogdan

Teammate: Bud Bogdan

Group: 30432

# 1)Abstract

This paper focuses on presenting the application developed during this semester for the Software Engineering class. It presents the progress made from the initial mini project and showcases some techniques and tools specific to this class.

For the subject of this project we chose a quiz type application as we saw it as a good candidate to incorporate as many relevant elements of this course as possible. It enables multiple Android devices to connect to a local server and solve a multiple answer quiz independently.

The paper will include all the main references used as guidelines in designing the programs as well as the source code from the initial and final project. Relevant code segments will also be discussed and the contribution of each student will be highlighted.

# 2)Project Description

## 2.1) Initial mini project

The initial application is built on the MVC architectural style and showcases a simple interface that allows the user to insert, update and delete users from a database. The complete source code is the following:

```
@Controller
public class MainController {
    @Autowired

    private UserRepository userRepository;

    @GetMapping("/signup")
    public String showSignUpForm(User user) {
        return "add-user";
    }

    @PostMapping("/adduser")
    public String addUser(@Valid User user, BindingResult result, Model model) {
        if (result.hasErrors()) {
            return "add-user";
        }

        userRepository.save(user);
        model.addAttribute("users", userRepository.findAll());
        return "index";
    }

    @GetMapping("/edit/{id}")
    public String showUpdateForm(@PathVariable("id") int id, Model model) {
        User user = userRepository.findById(id)
```

```
                .orElseThrow(() -> new IllegalArgumentException("Invalid user Id:" +
id));

        model.addAttribute("user", user);
        return "update-user";
    }

    @PostMapping("/update/{id}")
    public String updateUser(@PathVariable("id") int id, @Valid User user,
                             BindingResult result, Model model) {
        if (result.hasErrors()) {
            user.setId(id);
            return "update-user";
        }

        userRepository.save(user);
        model.addAttribute("users", userRepository.findAll());
        return "index";
    }

    @GetMapping("/delete/{id}")
    public String deleteUser(@PathVariable("id") int id, Model model) {
        User user = userRepository.findById(id)
                .orElseThrow(() -> new IllegalArgumentException("Invalid user Id:" +
id));
        userRepository.delete(user);
        model.addAttribute("users", userRepository.findAll());
        return "index";
    }
}
```

```
public interface UserRepository extends CrudRepository<User, Integer> {

}
```

```
@Entity
public class User {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;

    @NotBlank(message = "Name is mandatory")
    private String name;

    @NotBlank(message = "Email is mandatory")
    private String email;

    public Integer getId() {
        return id;
    }
```

```
    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }


}
```

```
@SpringBootApplication
public class AccessingDataMysqlApplication {

    public static void main(String[] args) {
        SpringApplication.run(AccessingDataMysqlApplication.class, args);
    }

}
```

## 2.2) Final project

For the final project we decided to make a server/client quiz application. The server part of the application works mostly as an API (It provides questions and answers from a MS SQL database) through a HTTP protocol in a JSON format, but it also does a computation part to keep the score of the players.

The Client is an application that can run on multiple Android devices connected to the same local server. In order to participate in the game and finish the quiz, each client makes multiple requests to the same local server.

## Code for the server:

```
public class AnswerDto {
    private Long id;
    private String text;
    private boolean correct;
    private Long question_id;
    private String player;

    public AnswerDto(Answer answer) {
        this.id = answer.getId();
        this.text = answer.getText();
        this.correct = answer.isCorrect();
        this.question_id = answer.getQuestion().getId();
        this.player = null;
    }

    public AnswerDto(Long id, String text, boolean correct, Long question_id, String
player) {
        this.id = id;
        this.text = text;
        this.correct = correct;
        this.question_id = question_id;
        this.player = player;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public boolean isCorrect() {
        return correct;
    }

    public void setCorrect(boolean correct) {
        this.correct = correct;
    }

    public Long getQuestion_id() {
        return question_id;
    }
```

```
    public void setQuestion_id(Long question_id) {
        this.question_id = question_id;
    }

    public String getPlayer() {
        return player;
    }

    public void setPlayer(String player) {
        this.player = player;
    }
}
```

```
@Entity
@Table(name = "tbl_answer")
public class Answer {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column
    private String text;

    @Column
    private boolean correct;

    @ManyToOne(fetch = FetchType.LAZY)
    @JsonIgnore
    private Question question;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public boolean isCorrect() {
        return correct;
    }

    public void setCorrect(boolean correct) {
        this.correct = correct;
    }
```

```
    public Question getQuestion() {
        return question;
    }

    public void setQuestion(Question question) {
        this.question = question;
    }
}
```

```
@Service
public class AnswerService {
    private static final Logger LOGGER =
LoggerFactory.getLogger(AnswerService.class);

    @Async
    public CompletableFuture<Boolean> checkIfAnswerIsRight(AnswerDto answerDto){
        LOGGER.info("Answer is " + (answerDto.isCorrect() ? "correct" : "false"));
        return CompletableFuture.completedFuture(answerDto.isCorrect());
    }
}
```

```
@RestController
public class AnswerController {

    @Autowired
    private PlayerService playerService;
    @Autowired
    private AnswerService answerService;
    @PostMapping("/answers")
    public ResponseEntity submitAnswer(@RequestBody AnswerDto answerDto){
        try {
            CompletableFuture<Player>
fetchPlayer=playerService.getPlayerByNameAsync(answerDto.getPlayer());
            CompletableFuture<Boolean> correct =
answerService.checkIfAnswerIsRight(answerDto);
            Player player = fetchPlayer.get();

            CompletableFuture.allOf(fetchPlayer, correct).join();
            if (correct.get())
                player.setScore(player.getScore() + 1);
            playerService.updateScore(player);
        } catch (Exception e) {
            e.printStackTrace();
            return new ResponseEntity(HttpStatus.INTERNAL_SERVER_ERROR);
        }

        return new ResponseEntity(HttpStatus.OK);
    }

}
```

```java
@Configuration
@EnableAsync
public class AsyncConfiguration {
    private static final Logger LOGGER =
LoggerFactory.getLogger(AsyncConfiguration.class);
    @Bean(name = "taskExecutor")
    public Executor taskExecutor() {
        LOGGER.debug("Creating Async Task Executor");
        final ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(10);
        executor.setMaxPoolSize(10);
        executor.setQueueCapacity(100);
        executor.setThreadNamePrefix("ScoringThread-");
        executor.initialize();
        return executor;
    }
}
```

```java
@RestController
public class PlayerController {
    @Autowired
    private PlayerService playerService;

    @PostMapping("player")
    public ResponseEntity postPlayer(@RequestParam String name){
        if(name != null && playerService.getPlayerByName(name) == null) {
            Player player = new Player(name, (long) 0);
            playerService.savePlayer(player);
            return new ResponseEntity(HttpStatus.OK);
        }
        else
            return new ResponseEntity(HttpStatus.BAD_REQUEST);

    }

    @GetMapping("player/{name}/score")
    public ResponseEntity<Long> getScoreForPlayer(@PathVariable String name){
        return new
ResponseEntity<Long>(playerService.getPlayerByName(name).getScore(), HttpStatus.OK);
    }


}
```

```java
@Entity
@Table(name = "tbl_player")
public class Player {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
```

```
    @Column
    private String name;

    @Column
    private Long score;

    public Player(String name, Long score) {
        this.name = name;
        this.score = score;
    }
    public Player() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Long getScore() {
        return score;
    }

    public void setScore(Long score) {
        this.score = score;
    }
}
```

```
@Repository
public interface PlayerRepository extends JpaRepository<Player, Long> {
    Optional<Player> findPlayerByName(String name);
}
```

```
@RestController
public class QuestionController {
    @Autowired
    private QuestionService questionService;

    @GetMapping("question/{id}")
    public QuestionDto findQuestionById(@PathVariable Long id){
        return new QuestionDto(questionService.findById(id));
```

```
    }

    @GetMapping("question/{id}/answers")
    public List<AnswerDto> getAnswersForQuestion(@PathVariable Long id){
        return
StreamSupport.stream(questionService.findAnswersForQuestion(id).spliterator(),
false).map(AnswerDto::new).collect(Collectors.toList());

    }

}
```

```
public class QuestionDto {
    private Long id;
    private String text;

    public QuestionDto(Question question) {
        this.id = question.getId();
        this.text = question.getText();
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }
}
```

```
@Entity
@Table(name = "tbl_question")
public class Question {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column
    private String text;
    @OneToMany(mappedBy = "question",
    cascade = CascadeType.ALL,
    orphanRemoval = true)
    @JsonIgnore
    private List<Answer> answers;
```

```
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    public List<Answer> getAnswers() {
        return answers;
    }

    public void setAnswers(List<Answer> answers) {
        this.answers = answers;
    }

    public void addAnswer(Answer answer){
        answers.add(answer);
        answer.setQuestion(this);
    }

    public void removeAnswer(Answer answer){
        answers.remove(answer);
        answer.setQuestion(null);
    }
}
```

```
@Repository
public interface QuestionRepository extends JpaRepository<Question, Long> {
}
```

```
@Service
public class QuestionService {
    @Autowired
    private QuestionRepository questionRepository;

    public Question findById(Long id){
        return questionRepository.findById(id).get();
    }

    public List<Answer> findAnswersForQuestion(Long qId){
        Question question = findById(qId);
```

```
        if(question.getAnswers().isEmpty())
            return Collections.emptyList();
        else
            return question.getAnswers();
    }
}
```

```
@SpringBootApplication
public class QuizappApplication {

    public static void main(String[] args) {
        SpringApplication.run(QuizappApplication.class, args);
    }

}
```

## Code for the client:

```
public class MainActivity extends AppCompatActivity {

    private static final int MAX = 3;
    private int questionCount=1;
    private TextView mTextViewResult;
    private TextView mTextViewAnswers;
    private TextView mPleaseEnter;
    private TextView mFinalText;
    OkHttpClient client = new OkHttpClient();

    String url = "http://192.168.16.101:8080/question/";

    private EditText mConfirmedName;
    private String savedName;
    private Button nameButton;
    private TextView mTextShowName;

    //ABCD buttons
    private Button a;
    private Button b;
    private Button c;
    private Button d;
    //Answers
    private int globalID[] ={1,2,3,4};
    private String globalText[]={"","","",""};
    private boolean globalCorrect[]={false,false,false,false};

    private int globalQID;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mTextViewResult = findViewById(R.id.text_view_result);
        mConfirmedName = findViewById(R.id.name_edittext);
```

```
        nameButton = findViewById(R.id.button5);
        mPleaseEnter=findViewById(R.id.textView);
        mTextShowName=findViewById(R.id.textView2);
        mTextViewAnswers=findViewById(R.id.textView3);
        mFinalText=findViewById(R.id.textView4);
        a=findViewById(R.id.button4);
        b=findViewById(R.id.button3);
        c=findViewById(R.id.button2);
        d=findViewById(R.id.button);




    }

    public void sendAnswer(int ID, String Text,boolean correct,int QID ){

        Judge judge = new Judge();
        if(!judge.start())
            correct = !correct;

        final MediaType JSON = MediaType.get("application/json; charset=utf-8");
        String json="{\n" +
                "        \"id\":" +ID+ ",\n" +
                "        \"text\":"+"\""+Text+"\""+ ",\n" +
                "        \"correct\":" +correct+",\n" +
                "        \"question_id\":" +QID+",\n" +
                "        \"player\":"+"\""+savedName+"\""+ "\n" +
                "}";
        RequestBody body = RequestBody.create(json, JSON);

        Request request2 = new Request.Builder()
                .url("http://192.168.16.101:8080/answers")
                .post(body)
                .build();

        client.newCall(request2).enqueue(new Callback() {
            @Override
            public void onFailure(@NotNull Call call, @NotNull IOException e) {

            }

            @Override
            public void onResponse(@NotNull Call call, @NotNull Response response)
throws IOException {

            }
        });




    }
    public void nextQuestion()  {

        Request request = new Request.Builder()
                .url(url+questionCount)
                .build();

        client.newCall(request).enqueue(new Callback() {
            @Override
```

```
            public void onFailure(Call call, IOException e) {
                e.printStackTrace();
            }

            @Override
            public void onResponse(Call call, Response response) throws IOException {
                if (response.isSuccessful()) {


                    String str=response.body().string();
                    String k=str.substring(6,7);
                    globalQID= Integer.parseInt(k);
                    System.out.println(k);

                    str = str.substring(16,str.length() - 2);

                    final String myResponse=str;


                        MainActivity.this.runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                mTextViewResult.setText(myResponse);
                                try {
                                    Thread.sleep(5);
                                } catch (InterruptedException e) {
                                    e.printStackTrace();
                                }
                            }
                        });




                }
            }
        });
    }







    public void nextAnswers()  {

        Request request = new Request.Builder()
                .url(url+questionCount+"/answers")
                .build();
        questionCount++;
        client.newCall(request).enqueue(new Callback() {
            @Override
            public void onFailure(Call call, IOException e) {
                e.printStackTrace();
            }

            @Override
            public void onResponse(Call call, Response response) throws IOException {
                if (response.isSuccessful()) {


                    String str=response.body().string();
```

```
                    try {

                        JSONArray Jarray = new JSONArray(str);
                        str="";
                        String lett[]={"A.","B.","C.","D."};
                        for (int i = 0; i < Jarray.length(); i++) {
                            JSONObject object     = Jarray.getJSONObject(i);


                            String k = (String) object.get("text");

                            globalID[i]= (int) object.get("id");
                            globalCorrect[i]= (boolean) object.get("correct");
                            globalText[i] = (String) object.get("text");
                            str=str+lett[i]+k+"\n";

                            System.out.println(k+"  "+globalID[i]+"
"+globalCorrect[i]+"  "+globalText[i]+"  ");
                        }

                    }catch (JSONException err){
                        Log.d("Error", err.toString());
                    }


                    final String myResponse=str;
                    MainActivity.this.runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            mTextViewAnswers.setText(myResponse);
                            try {
                                Thread.sleep(5);
                            } catch (InterruptedException e) {
                                e.printStackTrace();
                            }
                        }
                    });




                }
            }
        });
    }


    public void clearScreen(){

        mTextViewAnswers.setVisibility(View.GONE);
        mTextShowName.setVisibility(View.GONE);
        mConfirmedName.setVisibility(View.GONE);
        mTextViewResult.setVisibility(View.GONE);
        a.setVisibility(View.GONE);
        b.setVisibility(View.GONE);
        c.setVisibility(View.GONE);
        d.setVisibility(View.GONE);
        mPleaseEnter.setVisibility(View.GONE);

    }
```

```
    public void finishGame(){
        Request request = new Request.Builder()
                .url("http://192.168.16.101:8080/player/"+savedName+"/score")
                .build();

        client.newCall(request).enqueue(new Callback() {
            @Override
            public void onFailure(Call call, IOException e) {
                e.printStackTrace();
            }

            @Override
            public void onResponse(Call call, Response response) throws IOException {
                if (response.isSuccessful()) {


                    final String myResponse=response.body().string();


                    MainActivity.this.runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            mFinalText.setText("Congratulations, "+savedName+", your
score is "+myResponse+"!");
                            try {
                                Thread.sleep(5);
                            } catch (InterruptedException e) {
                                e.printStackTrace();
                            }
                        }
                    });

                }
            }
        });



    }

    public void sendD(View view) {

        if(questionCount>MAX)
        {
            clearScreen();
            finishGame();
        }else {
            sendAnswer(globalID[3], globalText[3], globalCorrect[3], globalQID);
            nextQuestion();
            nextAnswers();
        }
    }

    public void sendC(View view) {
        if(questionCount>MAX)
        {
            clearScreen();
            finishGame();
        }else {
            sendAnswer(globalID[2], globalText[2], globalCorrect[2], globalQID);
            nextQuestion();
            nextAnswers();
```

```
        }
    }

    public void sendB(View view) {
        if(questionCount>MAX)
        {
            clearScreen();
            finishGame();
        }else {
            sendAnswer(globalID[1], globalText[1], globalCorrect[1], globalQID);
            nextQuestion();
            nextAnswers();
        }
    }

    public void sendA(View view) {
        if(questionCount>MAX)
        {
            clearScreen();
            finishGame();
        }else {
            sendAnswer(globalID[0], globalText[0], globalCorrect[0], globalQID);
            nextQuestion();
            nextAnswers();
        }
    }

    public void confirmName(View view) {
        savedName=mConfirmedName.getText().toString();
        nameButton.setVisibility(View.GONE);
        mTextShowName.setText(savedName+"  GO GO GO!");

        final MediaType JSON = MediaType.get("application/json; charset=utf-8");
        String json="";
        RequestBody body = RequestBody.create(json, JSON);

        Request request2 = new Request.Builder()
                .url("http://192.168.16.101:8080/player?name="+savedName)
                .post(body)
                .build();

        client.newCall(request2).enqueue(new Callback() {
            @Override
            public void onFailure(@NotNull Call call, @NotNull IOException e) {

            }

            @Override
            public void onResponse(@NotNull Call call, @NotNull Response response)
throws IOException {

            }
        });



        nextQuestion();
        nextAnswers();
    }
}
public class ThreadA extends Thread {
```

```
    protected BlockingQueue blockingQueue;
    private int ma = 1;


    public ThreadA(BlockingQueue blockingQueue) {
        this.blockingQueue = blockingQueue;
    }

    @Override
    public void run() {
        try {
            while (!interrupted() && ma <= 10) {

                blockingQueue.put(ma);
                System.out.println("ThreadA reached" + ma);
                Thread.sleep(200);
                ma++;
                blockingQueue.take();
            }
        } catch (InterruptedException e) {
        }

    }

    public int cancel() {
        interrupt();
        return ma;
    }
}
```

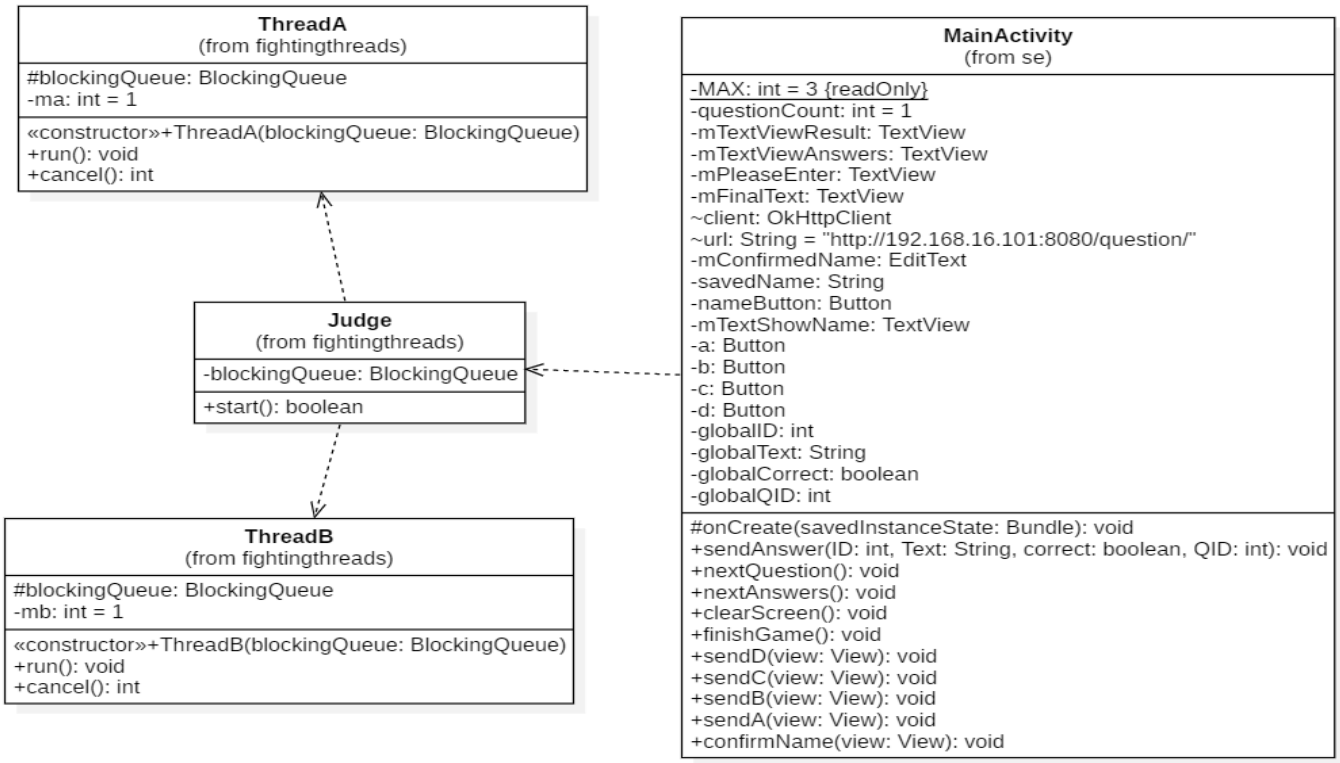
```
public class ThreadB extends Thread {
    protected BlockingQueue blockingQueue;
    private int mb = 1;


    public ThreadB(BlockingQueue blockingQueue) {
        this.blockingQueue = blockingQueue;
    }

    @Override
    public void run() {
        try {
            while (!interrupted() && mb <= 10) {
                blockingQueue.take();
                System.out.println("ThreadB reached" + mb);
                Thread.sleep(200);
                blockingQueue.put(mb);
                mb++;
            }
        } catch (InterruptedException e) {
        }
    }

    public int cancel() {
        interrupt();
        return mb;
    }
}
public class Judge {

    private BlockingQueue blockingQueue = new SynchronousQueue();
```

```
    public boolean start() {
        ThreadA threadA = new ThreadA(blockingQueue);
        ThreadB threadB = new ThreadB(blockingQueue);
        try {
            threadA.start();
            threadB.start();

            Random random = new Random();
            Thread.sleep(random.nextInt(2000));
            int ma =threadA.cancel();
            int mb =threadB.cancel();

            if(random.nextInt(9) + 1 < ma) {

                System.out.println("The elders have spoken, the exchange is fair");
                return true;
            }else{
                System.out.println("The elders have spoken, the exchange is NOT
fair");
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return false;
    }

}
```

## 2.2.1 Design documents:

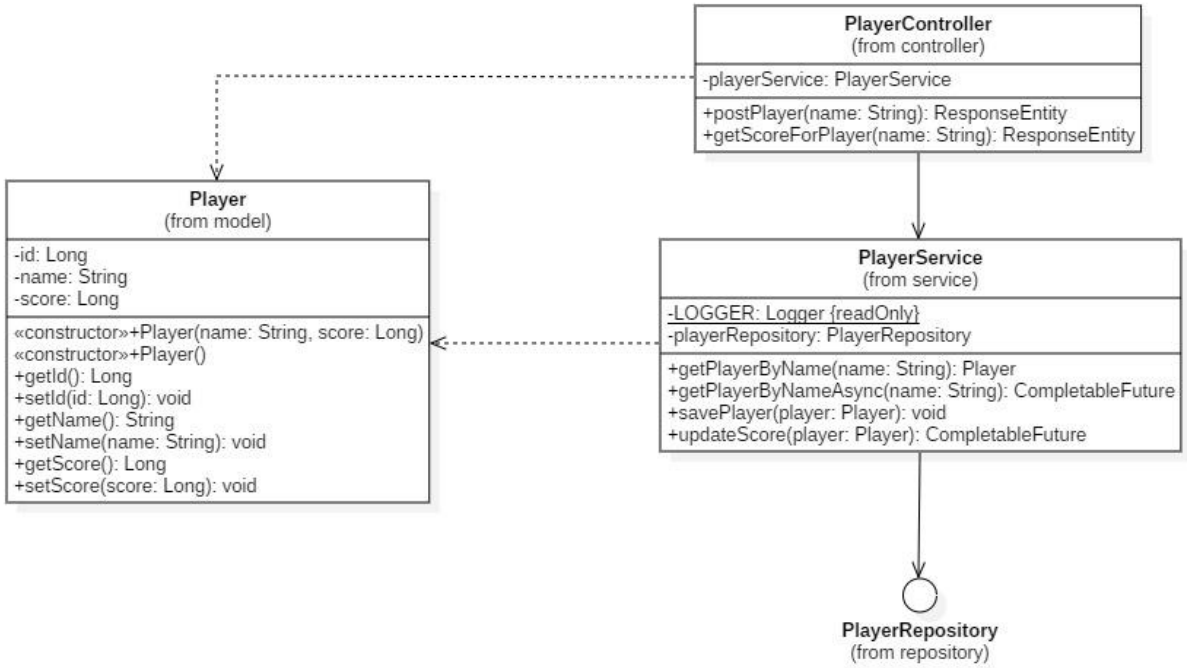
### 1) Uml class diagrams:

#### -client:

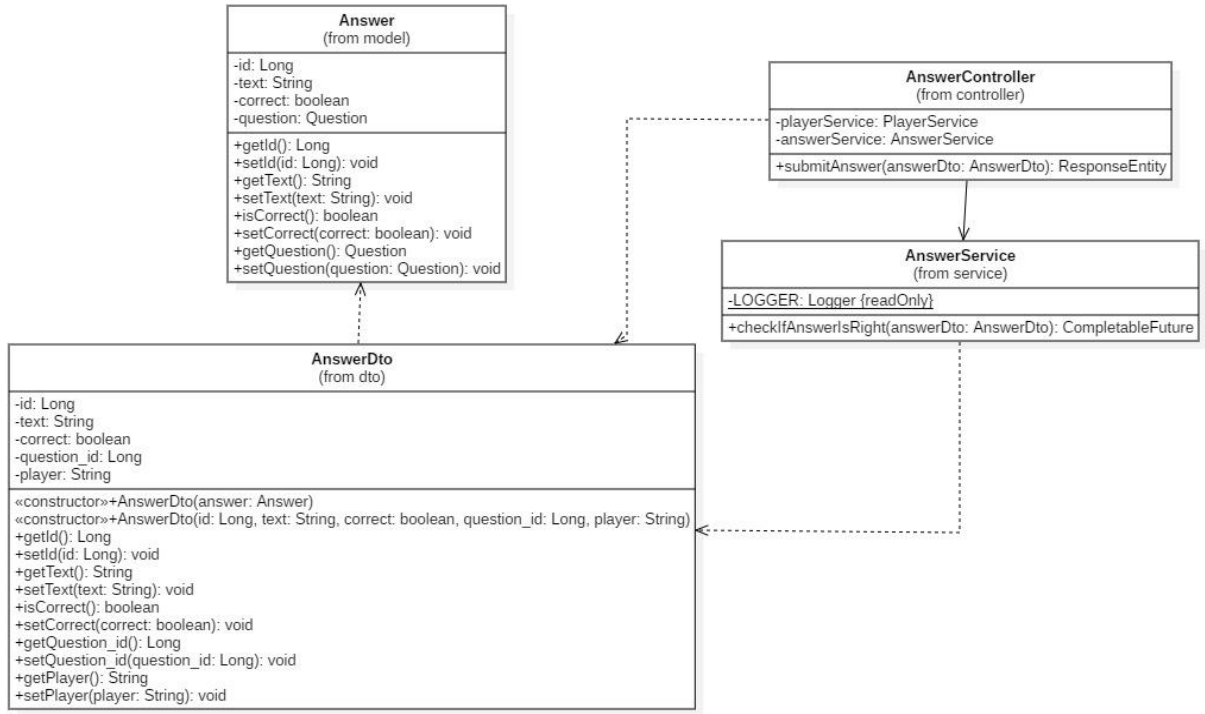


**-server:**

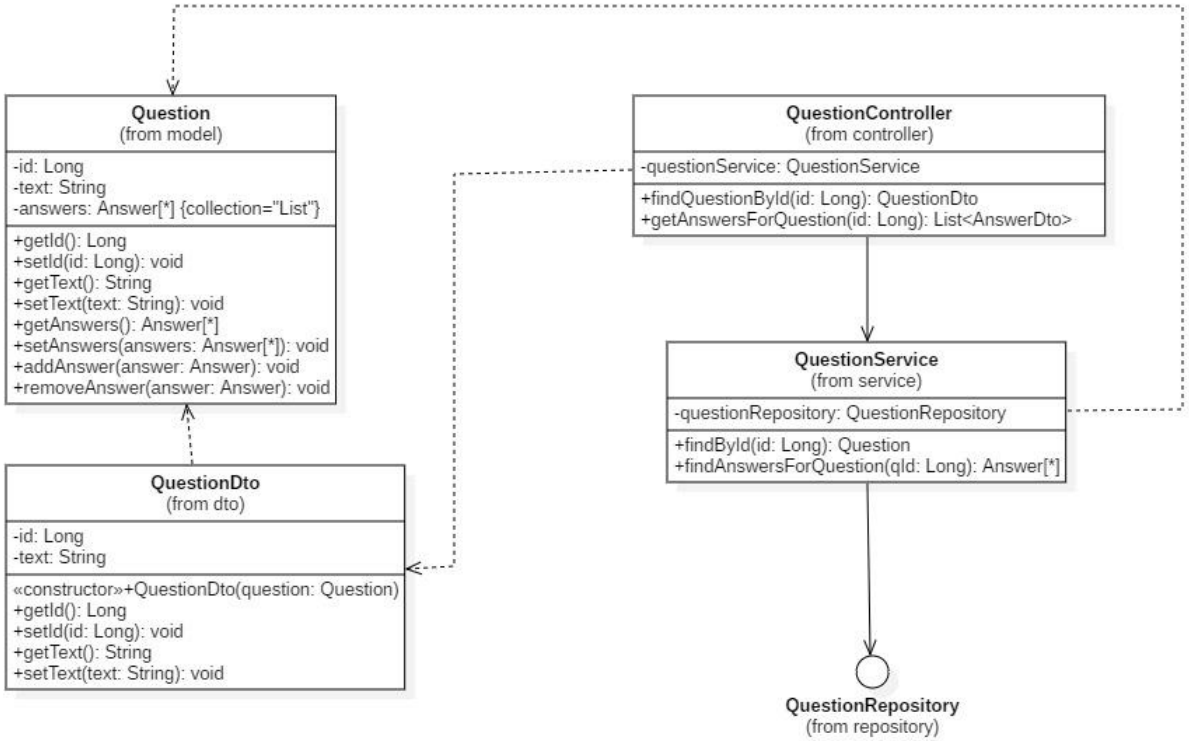
## a) Player



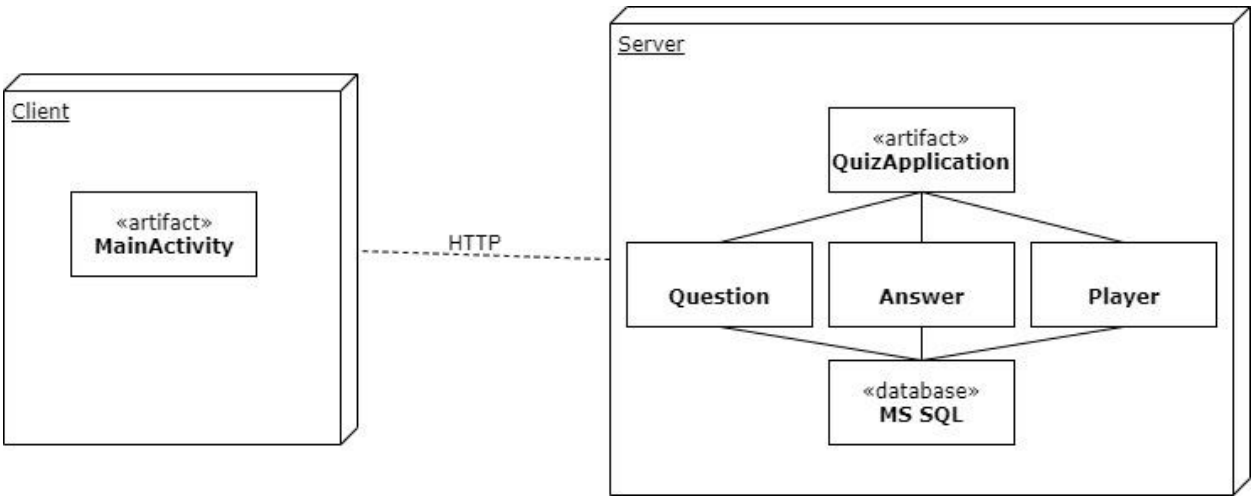

## b) Answer



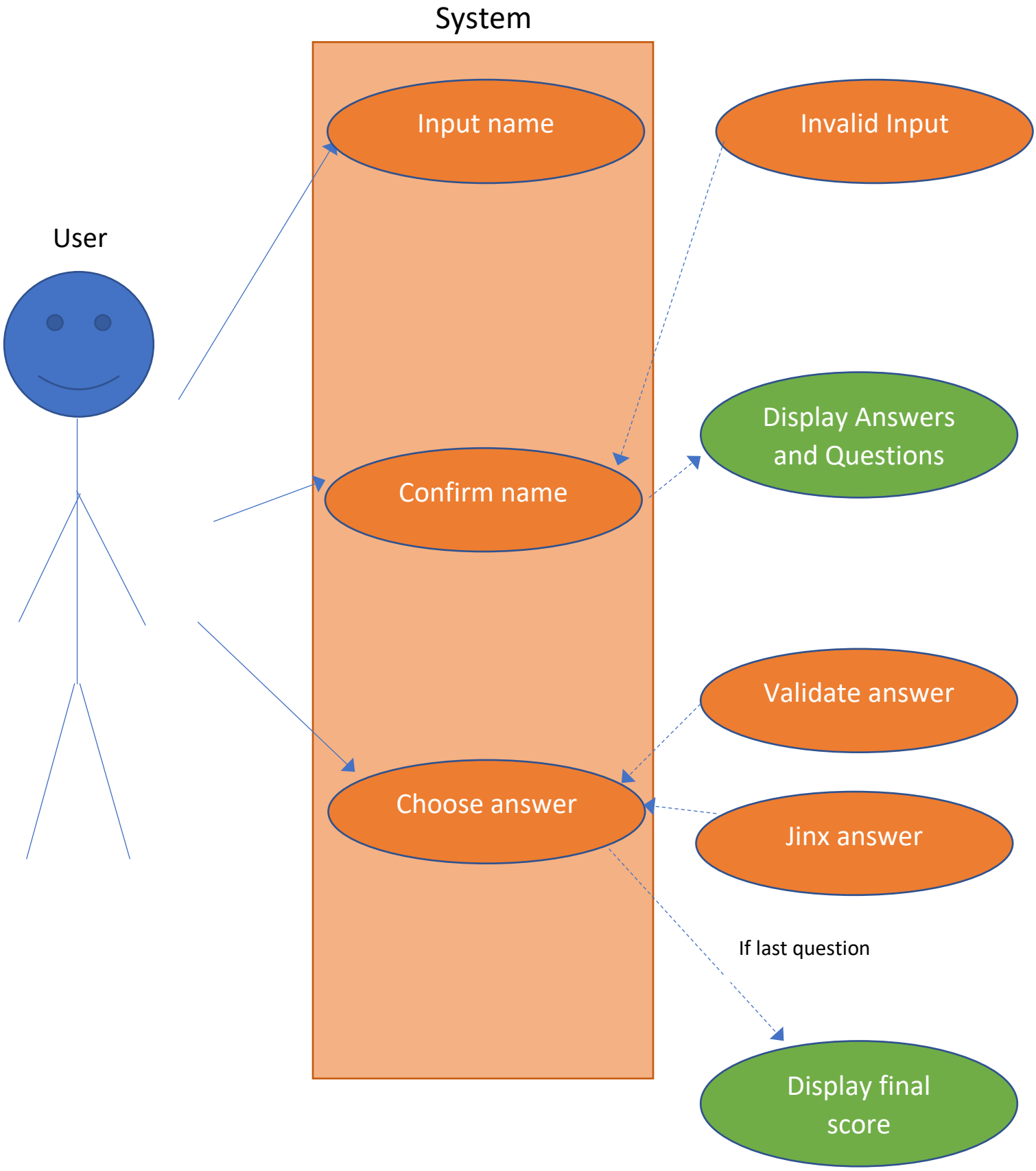
## c) Question




## 2) Deployment diagram



**3)Use case diagram**



# 3. Personal contribution

My contribution has been mainly the server part. Several microservices are used: Question, Answer and Player together with the MVC pattern. The View is handled by the client, while the Controller (service, persistence) and Model (Dto) are handled by the server.

Not all CRUD operation can be done on these microservices. Questions and Answers are added at run-time on the database and they need not be modified by the client. On the other hand, Players are created at the beginning of the quiz, so the clients can persist them in the database and find their scores.

The updating of the scores is done by the server part if the answer submitted by the player is correct. To add concurrency to our project, CompletableFuture is used and on a submitted answer, two separate threads are started: one that fetches the player from the database and one that computes the score. A third thread waits for the other two to finish in order to persist the new score of the player.

There are several endpoints available for clients to work with: "/question/{id}" provides the client with the question with id={id}, "question/{id}/answers" provides the client with the answers of that question.

Players can be persisted by providing a name as a PathParam on "/player?name=" and their score can be obtained with a get on "/player/{name}/score".

Answers are submitted to "/answers" where they are scored.

# 4. References


- https://www.prismmodelchecker.org/casestudies/fairexchange.php
- https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html
- https://square.github.io/okhttp/
- https://developer.android.com/training/volley/simple
- https://docs.oracle.com/javaee/7/api/javax/json/JsonArray.html
- https://docs.oracle.com/javaee/7/api/javax/json/JsonObject.html