Pandemic Support System Modelling and Implementation as Integral Part of Computer Science Courses

Nenad Petrović

Abstract — **In this paper, exercises related to modelling and implementation of pandemic-tackling systems are proposed as integral part of computer science university courses. The presented case study considers the perspective of two bachelor degree courses taught at the third year of Computer Science and Informatics track at University of Niš, Faculty of Electronic Engineering in Serbia covering both hardware and software design: Microcomputer Systems and Information Systems. For the first course, an indoor safety system based on Intel 8086 and additional components (8251, 8255 and 8259) is presented. On the other side, the main topic of the second one is Java Enterprise Edition (JEE)-based information system development, while the presented example shows application providing efficient pandemic-related data management (coronavirus tests and vaccination).**

Index Terms **— COVID-19; coronavirus; education; Intel 8086; Java Enterprise Edition (JEE).**

I. Introduction

The ongoing coronavirus pandemic has brought numerous challenges not only to healthcare and medical science-related personnel, but when it comes to engineering and information sciences professionals as well. Efficient pandemic-related data management in synergy with leveraging the collected information are recognized as key factors when it comes to role of information systems in global battle against COVID-19 [1]. On the other side, IoT and embedded devices are one of crucial means for disease spread reduction (such as mask detection and contactless temperature measurement) [2]. In this paper, we introduce two exercises aiming Information Systems and IoT bachelor degree courses at Faculty of Electronic Engineering, University of Niš in Serbia. The first one covers the design and development of software leveraging pandemic data, while another is a control unit of indoor safety system. The goal is to show how students can become aware of their role as computer engineers among other professionals tackling COVID-19, while studying Java Enterprise Edition software development and programming microcontrollers, on the other side.

The exercise case studies presented in this paper are inspired by author's intensive work in area of pandemic support information technology solutions since the beginning of COVID-19 outbreak. In [3], an affordable IoT-based indoor safety system for mask check and temperature detection leveraging Raspberry Pi and Arduino Uno was introduced. Moreover, the works from [4, 5] show methodology for efficient pandemic resource planning, such as vaccine allocation and tests. Furthermore [6] shows air quality regulation system for coronavirus spread reduction, while [2] addresses green certificates checking relying on blockchain. Finally, [7] shows adoption of data mining techniques in area of post-COVID tourism. On the other side, author also introduced exercises related to coronavirus protection within Logic Design [8, 9] course and microcontroller-related part of Microcomputer Systems [9, 10].

Nenad N. Petrović is with University of Niš, Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Niš, Serbia, (e-mail: nenad.petrovic@elfak.ni.ac.rs), (https://orcid.org/0000-0003-2264-7369)

II. Background

A. Information Systems Course

Information Systems is third year bachelor degree course, mainly covering Java Enterprise Edition (JEE) [11] and related technologies within practical sessions. The implementation-related assignment is homework project scored by 25% of the overall coursework. The underlying architecture is depicted in Fig. 1. In such application, user provides the corresponding input data and selects desired options via web browser-based interface. Moreover, in order to generate the expected results, business logic layer is responsible for calculations and data manipulation. Additionally, at some point it might rely on external microservices deployed within containers for some of the tasks. During this processing within business layer, objects holding the relevant data about events, users and other entities might be persisted within database or retrieved from there Finally, the outcome is shown back to the end-user inside web page. Later, the acquired data can be further leveraged for various types of predictions relevant to business goals.

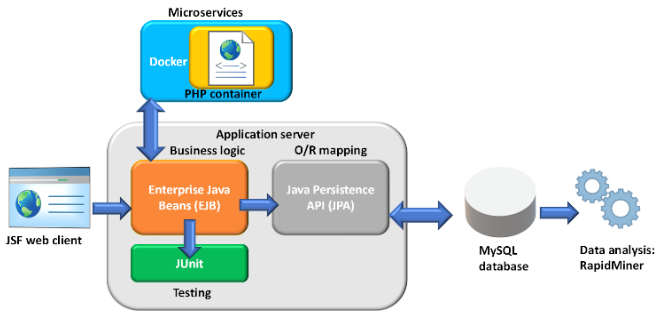


Fig. 1. Information Systems course project architecture.

An overview of components corresponding to the topics covered by the practice-oriented part of Information Systems course is given within Table 1.

TABLE I
INFORMATION SYSTEMS TOPIC OVERVIEW

Component	Type	Role
Java Server Faces (JSF) [11]	Java-based framework for web applications (front-end)	User data input and result visualization
Enterprise Java Beans (EJB) [11]	Business logic layer components for application JEE back-end	Calculations and data manipulation
JUnit [12]	Testing framework	Unit testing of business layer functionalities based on assert checking
Java Persistence API (JPA) [11]	Object-relational (O/R) mapping specification	Persistence of entities (Java class objects) in form of database rows
MySQL [13]	Relational database	Storage of relevant data within tables
Docker [14]	Containerization engine	Microservice implementation and deployment within isolated environment
RapidMiner [15]	Automated machine learning tool	Predictions using classification, regression, clustering and association rules

B. *Microcomputer Systems Course*

Microcomputer Systems are also obligatory third year bachelor degree course, but focused on hardware design instead. It consists of two main parts: 1) Intel 8086-based systems [16] 2) PIC16-family microcontrollers. In this paper, we focus on the first part, which represents around 30% of the overall course work, split into lab session exercises and written exam. When it comes to that topic, several additional components are also covered, as shown in Table 2.

TABLE II
MICROCOMPUTER SYSTEMS TOPIC OVERVIEW

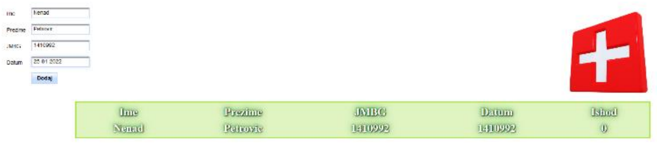
Component	Type	Role
8086	16-bit microprocessor	Control unit and data processing
8255	Parallel data peripheral interface	Output to devices such as 7-segment displays and LEDs or input handling, such as button press
8251	Serial data transfer interface	Receive/transmit data via single line from serial devices and convert it to parallel in order to be usable by 8086
8259	Programmable Interrupt Controller	Detecting changes on interrupt lines and providing mechanism for response to events which occurred in the external environment

III. Case Studies

A. *COVID-19 Healthcare System for Test and Vaccination*

The goal of the project assignment is to model and implement a healthcare information system aiming to support COVID-19 testing and vaccination processes. The complete source course of the project is given within public online GiHub repository: https://github.com/penenadpi/is2022_covid.

It gives the ability to enter the related data using corresponding JSF pages, as shown within screenshot of test-related interface (Fig. 2). On the other side, it enables the storage and retrieval of relevant information about vaccination as well, such as ambulance identifier, date, vaccine type, the number of dose and person identifier.



Ime	Nenad
Prezime	Petrovic
JMBG	1410992
Datum	25 01 2022

Dodaj

Ime	Prezime	JMBG	Datum	Ishod
Nenad	Petrovic	1410992	1410992	0

Fig. 2. JSF page for COVID-19 test records.

We can distinguish between three main types of Java class components within the application (as illustrated in Fig. 3): 1) entity classes – responsible for data persistence about relevant objects (*VaccinationRecord* and *TestRecord*); 2) session beans services – implementation of business logic services together with their corresponding interfaces (*VaccinationService* and *TestService*); 3) controllers - functionalities related to a single JSF page (*VaccinationController* and *TestController*). The underlying flow can be described as follows: controllers are triggered by clicking on buttons within the JSF page, while inside them, the calls to corresponding services leveraging the data (storing or reading) and performing processing are invoked. These operations are executed against entity objects which are permanently persistent within database relying on O/R mapping.

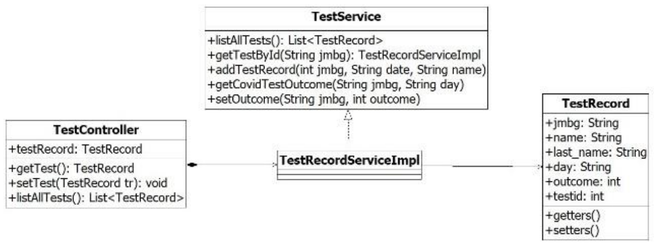
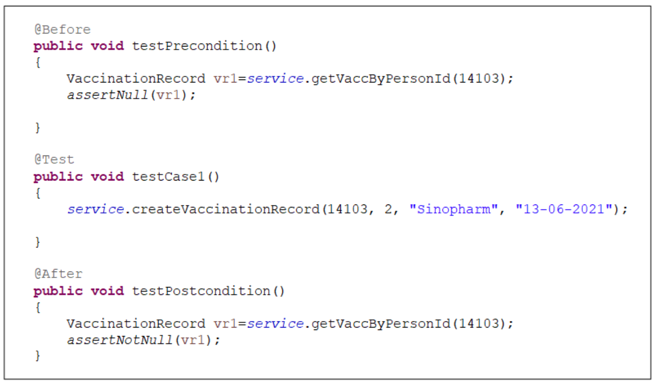

TestService
+listAllTests(): List<TestRecord>
+getTestById(String jmbg): TestRecordServiceImpl
+addTestRecord(int jmbg, String date, String name)
+getCovidTestOutcome(String jmbg, String day)
+setOutcome(String jmbg, int outcome)
TestController
+testRecord: TestRecord
+getTest(): TestRecord
+setTest(TestRecord tr): void
+listAllTests(): List<TestRecord>
TestRecordServiceImpl
TestRecord
+jmbg: String
+name: String
+last_name: String
+day: String
+outcome: int
+testid: int
+getters()
+setters()

Fig. 3. Excerpt of UML class diagram for coronavirus test records.

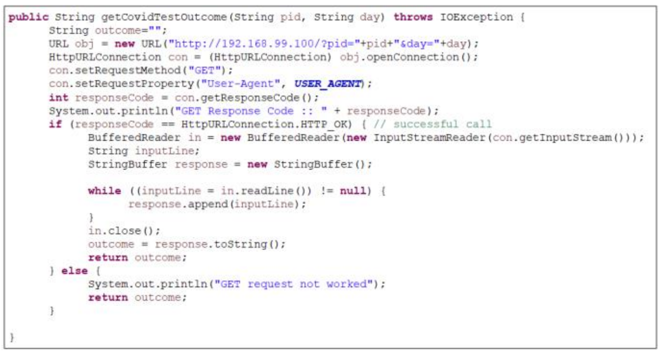
On the other side, when it comes to testing, JUnit framework for unit testing in Java is covered by this course. Unit tests aim to check whether lowest level executable units (such as single method in object-oriented program or function) give correct result. For that purpose, test cases covering relevant features of the project are written. An example which checks whether new vaccination records are inserted correctly is given in Fig. 3. The test case consists of three main parts: 1) pre-condition: assert which needs to hold before execution of the test itself 2) test steps: the functional call to the method being tested 3) post-condition: condition which should be true after the execution of test steps. As it can be seen in Fig. 4, the precondition checks whether vaccination record for given person does not exist. Then, the test itself adds vaccination record. Finally, the post-condition checks if insertion had been successful by retrieving the vaccination record of provided person and testing if it is not null value.



```java
@Before
public void testPrecondition()
{
    VaccinationRecord vr1=service.getVaccByPersonId(14103);
    assertNull(vr1);

}

@Test
public void testCase1()
{
    service.createVaccinationRecord(14103, 2, "Sinopharm", "13-06-2021");

}

@After
public void testPostcondition()
{
    VaccinationRecord vr1=service.getVaccByPersonId(14103);
    assertNotNull(vr1);
}
```

Fig. 4. JUnit test case implementation.

When it comes to integration with microservices, the layer of business logic (*CovidTestService* session bean) interacts with PHP script deployed in a container via HTTP request. In Fig. 5, an example in context of our project assignment is shown. When it comes to insertion of COVID test results, the value of test outcome field is taken as result of PHP script deployed within web server inside Docker container. The script itself takes person identifier and current date as input, while the outcome is randomly generated (0 or 1), simulating the randomness regarding the probability of being infected.



```java
public String getCovidTestOutcome(String pid, String day) throws IOException {
        String outcome="";
        URL obj = new URL("http://192.168.99.100/?pid="+pid+"&day="+day);
        HttpURLConnection con = (HttpURLConnection) obj.openConnection();
        con.setRequestMethod("GET");
        con.setRequestProperty("User-Agent", USER_AGENT);
        int responseCode = con.getResponseCode();
        System.out.println("GET Response Code :: " + responseCode);
        if (responseCode == HttpURLConnection.HTTP_OK) { // successful call
            BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
            String inputLine;
            StringBuffer response = new StringBuffer();

            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
            in.close();
            outcome = response.toString();
            return outcome;
        } else {
            System.out.println("GET request not worked");
            return outcome;
        }

}
```

Fig. 5. Integration with Docker container microservice.

Finally, the last part of the project assignment involves leveraging the data collected by the described information system in order to make predictions which could be relevant to the pandemic support domain. For this purpose, we make use of RapidMiner tool, which gives intuitive and automated visual interface for machine learning. The presented example shows the adoption of regression, whose goal is to predict the value of numerical (continuous-valued) outcome. The layout of the underlying dataset based on the collected data is shown in Table 3. The goal is to predict vaccine demand (number of persons vaccinated) for given day. Relevant factors (input, independent variables) are day number, vaccine type, institution id and number of daily COVID-positive cases. On the other side, the predicted value (dependent variable) is the number of people who decided to take the vaccine.

TABLE III
VACCINE DEMAND PREDICTION DATASET HEADER

Day	Vaccine type	Institution	New cases	Demand

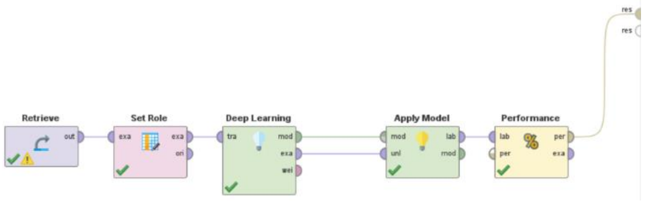


Fig. 6. Vaccination demand prediction workflow in RapidMiner.

The corresponding RapidMiner flow for regression-based prediction is shown in Fig. 6. The first step imports CSV containing all the data. After that, we select the target variable among the available columns. The third step is deep learning module which is trained to make predictions on 80% of the dataset. It contains 2 hidden layers, 30 nodes each and ReLU activation function, performs training in 10 epochs with adaptive learning rate. After that, the model is executed on test data and performance evaluated (mean relative error, which was around 4% in our case).

B. *Control Unit of Coronavirus Indoor Safety System*

On the other side, the goal of Microcomputer Systems assignment is design of control unit within COVID-19 indoor safety system (depicted in Fig. 7). Complete code with Proteus simulation is available on GitHub: https://github.com/penenadpi/8086covidsafety/blob/main/8086_8255_8259_8251_covid_safety.pdsprj

Each time new person arrives, two checks are performed by external devices: 1) temperature check – whether it is in normal range (less than 37°C) 2) mask check – if person wears mask or not. After that, the data is sent via 8251 component to the microprocessor, relying on 8259 for

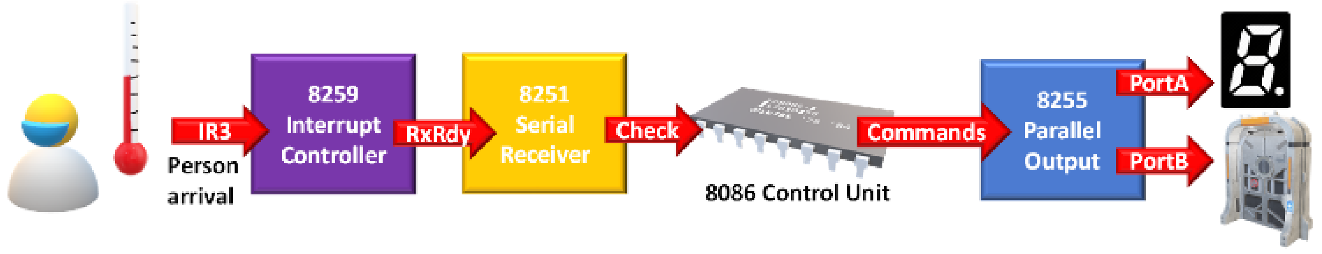


Fig. 7. 8086-based indoor coronavirus safety control system.

interrupt handling (vector 171, line 3). In case that person passes both checks, then the door will be opened only in case that the current number of persons is in the allowed range for that room. Additional output peripheral devices are connected via ports of 8255: 1) green LED – turned on for opening door (PORTB, pin 1); 2) red LED – turned on for closing door (PORTB, pin 0); 3) 7-segment display (PORTA) – showing the current number of persons inside. However, when person leaves room, interrupt which decreases the number of persons inside the room is activated, as button on interrupt line 1, vector 169.

```
PROCED SEGMENT
ENTRANCE_CHECK PROC FAR
ASSUME CS:PROCED, DS:DATA

 MOV DX, CTRL
 IN AL, DX

 MOV DX, DAT
 IN AL, DX

 XOR AL, MASK_TEMP_CHECK
 JNZ CLOSE_DOOR

 CMP SI, 5
 JE CLOSE_DOOR

 INC SI

 MOV DX, PORTA
 MOV AL, NUMBERS[SI]
 OUT DX, AL

 MOV DX, PORTB
 MOV AL, GREEN
 OUT DX, AL

 JMP END

CLOSE_DOOR:
 MOV DX, PORTB
 MOV AL, RED
 OUT DX, AL

END:

 IRET
ENTRANCE_CHECK ENDP
PROCED ENDS
```

Fig. 8. 8086 interrupt procedure activated when 8251 receives data.

An excerpt of the solution containing the crucial part of the assembly code run on Intel 8086, related to interrupt triggered when new person arrives is shown in Fig. 8. As interrupt line activation means that 8251 is ready for receiving data, we set the corresponding data exchange address and receive the outcome of the entrance rules checks (mask and temperature) coming from other devices. After that, the received byte is compared against the only case which allows opening the door (MASK_TEMP_CHECK which is 18 hexadecimal). If it is not the case, the red LED will be turned on (door closed). Additionally, if person passes this check, it is also asked whether the current number of persons inside the room is below the maximum threshold (5 in this case). Only if it is true, the door will be opened (green LED turned on), the current number of persons incremented and shown on 7-segment display.

IV. Evaluation

The proposed approach of including pandemic-related exercises within computer science bachelor degree courses has been evaluated by observing the results before and after their adoption. The following aspects were considered: number of students involved into lab sessions, average lab exam grade and exam pass rate. Fig. 9a and 9b show graphical summary of results compared to previous academic years for Microcomputer and Information Systems courses, respectively.

When it comes to Microcomputer Systems, COVID-19 protection systems were also part of additional student challenges bringing bonus points which can change the whole written exam, where participation was around 15% of the total students involved.

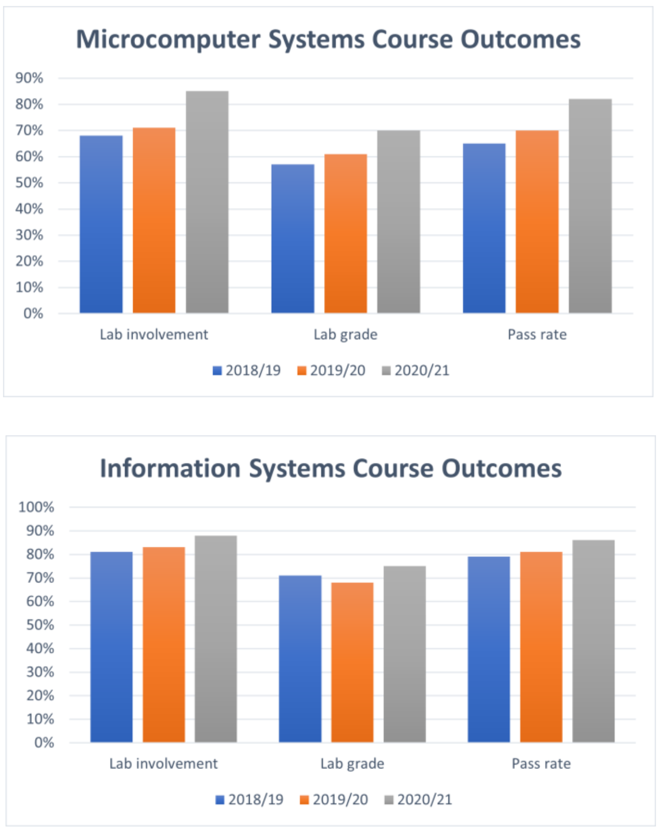


Fig. 9. Course outcomes before and after adoption of the proposed teaching methodology: a) Microcomputer Systems b) Information Systems.

V. Conclusion

In this paper, it was shown how the implementation of pandemic support solutions can be integrated within computer science university cases, considering the case studies of both hardware and software design. According to authors' observations, the adoption of such exercises in previous school year (2020/21) has shown several benefits for both of the considered courses: greater student involvement into lab sessions together with better lab exercise grades and higher exam pass rate. Taking into account all the considerations, the adopted teaching methodology has been slightly more effective in case of Microcomputer Systems course.

Therefore, the approach seems promising, showing that consideration of current real-world problems within computer science courses increases motivation and interest among participants, leading to better overall results.

ACKNOWLEDGMENT

This work has been supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia.

REFERENCES

[1] Q. Wang, M. Su., M. Zhang, R. Li, “Integrating Digital Technologies and Public Health to Fight Covid-19 Pandemic: Key Technologies, Applications, Challenges and Outlook of Digital Healthcare”, International Journal of Environmental Research and Public Health. 2021; 18(11):6053, 2021.

[2] N. Petrović, Đ. Kocić, “Smart technologies for COVID-19 indoor monitoring”, Viruses, Bacteria and Fungi in the Built Environment, ISBN: 9780323852067, Woodhead Publishing, pp. 251-272, 2021. https://doi.org/10.1016/B978-0-323-85206-7.00012-5

[3] N. Petrović, Đ. Kocić, “IoT-based System for COVID-19 Indoor Safety Monitoring”, IcETRAN 2020, pp. 603-608, 2020.

[4] N. Petrović, “Simulation Environment for Optimal Resource Planning During COVID-19 Crisis”, ICEST 2020, pp. 23-26, 2020. https://doi.org/10.1109/ICEST49890.2020.9232908

[5] N. Petrović, I. Al-Azzoni, "Model-Driven Approach to COVID-19 Vaccination Planning Leveraging Multi-Objective Optimization and Deep Learning", SSSS 2022, pp. 19-24, 2022.

[6] N. Petrović and Đ. Kocić, "IoT for COVID-19 Indoor Spread Prevention: Cough Detection, Air Quality Control and Contact Tracing," 2021 IEEE 32nd International Conference on Microelectronics (MIEL), 2021, pp. 297-300. https://doi.org/10.1109/MIEL52794.2021.9569099

[7] N. Petrović, "VHDL Logic Design Exercises Simulating COVID-19 Protection Systems", SSSS 2022, pp. 127-132.

[8] N. Petrović, “COVID-19 Safety Systems Design Exercises in Computer Science University Courses”, YuInfo 2021, pp. 1-6.

[9] N. Petrović, “Prototyping PIC16-based COVID-19 indoor safety solutions within microcomputer systems course”, IEEESTEC – 13th Student Projects Conference, pp. 185–189, 2020.

[10] N. Petrović, V. Roblek, N. Papachashvili, “Decision Support Based on Data Mining for Post COVID-19 Tourism Industry”, SAUM 2021, pp. 29-32, 2021.

[11] D. Heffelfinger, Java EE 8 Application Development, Packt Publishing, 2017.

[12] JUnit 4 [online], available on: https://junit.org/junit4/ , last accessed: 25/03/2022.

[13] MySQL [online], available on: https://www.mysql.com/ , last accessed: 25/03/2022.

[14] Docker [online], available on: https://www.docker.com/ , last accessed: 25/03/2022.

[15] RapidMiner [online], available on: https://rapidminer.com/ , last accessed: 25/03/2022.

[16] B. Brey, The Intel microprocessors: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro processor, Pentium II, Pentium III, Pentium 4, and Core2 with 64-bit extensions: architecture, programming, and interfacing, Pearson Prentice Hall, 2019.