

Mariia Portnykh

Názov projektu: TeamSurveyManager**Zámer projektu:**

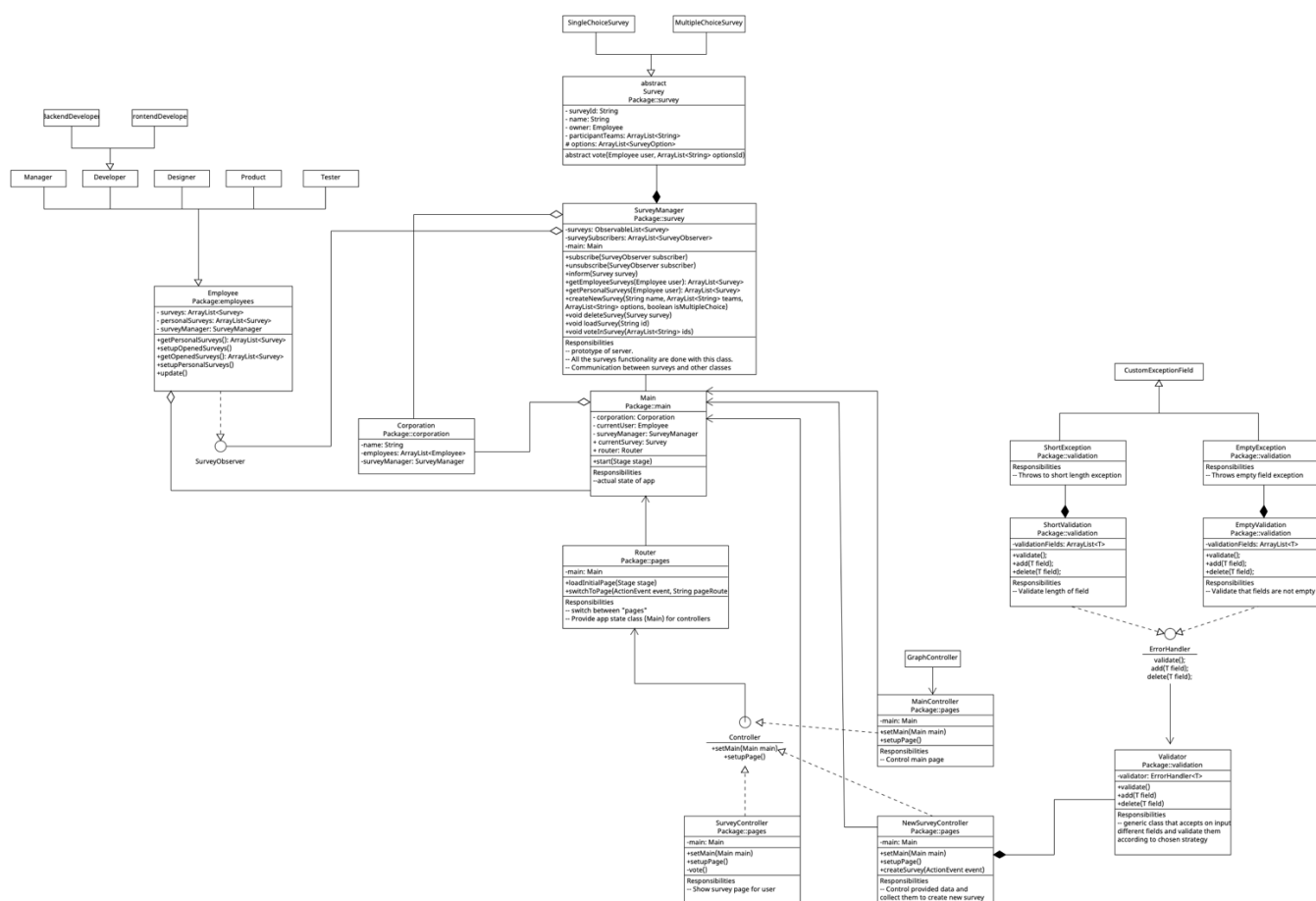
Cieľom môjho projektu je navrhnuť inovatívny systém hlasovania na riešenie problémov pre spoločnosti. Hlavnou funkciou aplikácie je vytváranie prieskumov, obmedzené na konkrétne skupiny zamestnancov.

Aké sú výhody?

Tento systém poskytuje efektívne riešenie problémov prostredníctvom hlasovania medzi rôznymi kategóriami zamestnancov. Povedzme, že je problém alebo otázka týkajúca sa propagácie produktu, ktorý firma vyrába. V tomto prípade môže člen tímu "products" vytvoriť prieskum a priradiť všetkých zamestnancov patriacich do tohto tímu. Ďalším príkladom môže byť výber medzi rôznymi variantmi dizajnu. V takejto situácii, je možnosť vytvoriť hlasovanie obmedzené na 2 skupiny tímov – frontend developeri a dizajnéri. Lebo nie je to len otázka toho, ako to bude vyzerať, ale aj otázka, či sa to dá implementovať. Druhou výhodou aplikácie je, že výsledok prieskumu bude závisieť najmä od hodnotenia účastníkov. Týmto spôsobom zamestnanci s bohatšími pracovnými skúsenosťami budú mať vážnejší hlas pri rozhodovaní.

Ako to funguje?

Na začiatku treba vstúpiť do systému vašej spoločnosti, kde pracujete, alebo zaregistrovať svoju vlastnú firmu. Systém obsahuje zoznam všetkých zamestnancov spoločnosti, vrátane ich prax, odbor (či tím) a iné relevantné informácií. Ľubovoľný človek zo zoznamu môže vytvoriť hlasovanie k nejakej otázke a priradiť do neho akékoľvek skupiny pracovníkov, podľa potrieb. Systém upozorní všetkých priradených účastníkov. Výsledkom bude graf.

UML diagram:

Hlavné kritériá:

- Dedenie: SingleChoiceSurvey a MultipleChoiceSurvey sa dedia od triedy Survey. Trieda Employee má deti ako Tester, Product, Developer atď.
- Polymorfizmus: Polymorfizmus bol využitý v príklade s prieskumami, teda vytvorila som overriding metódy vote v triedach SingleChoiceSurvey a MultipleChoiceSurvey.

```

1 abstract public class Survey {
2     2 usages
3     private String surveyId;
4     no usages
5     private String description;
6     2 usages
7     private final String name;
8     3 usages
9     private final Employee owner;
10    3 usages
11    private final ArrayList<String> participantTeams;
12    7 usages
13    protected ArrayList<SurveyOption> options;
14
15    A mark
16    1 usage 1 mark
17    public Survey(String name, ArrayList<String> participantTeams, ArrayList<String> options, Employee owner) {
18    }
19    2 usages 1 mark
20    public boolean containEmployee(Employee employee) {
21    }
22    2 usages 1 mark
23    public boolean checkOwner(Employee checkedOwner) {
24    }
25
26    6 usages 1 mark
27    public String getSurveyName() { return name; }
28    4 usages 1 mark
29    public String getSurveyId() { return surveyId; }
30
31    2 usages 1 mark
32    public ArrayList<SurveyOption> getSurveyOptions() { return this.options; }
33
34    1 usage 2 implementations 1 mark
35    abstract public void vote(Employee user, ArrayList<String> optionsId);
36 }

7 public class MultipleChoiceSurvey extends Survey {
8
9     1 usage 1 mark
10    MultipleChoiceSurvey(String name, ArrayList<String> teams, ArrayList<String> options, Employee owner) {
11        super(name, teams, options, owner);
12    }
13
14    1 usage 1 mark
15    @Override
16    public void vote(Employee user, ArrayList<String> optionsId) {
17        int votesNumber = 0;
18        for (SurveyOption option: options) {
19            option.removeVoter(user);
20            if (optionsId.contains(option.getId())) option.addVoter(user);
21            votesNumber += option.getVotersSize();
22        }
23
24        for (SurveyOption option: options) {
25            option.calculateVotesNumber();
26        }
27    }
28 }

7 public class SingleChoiceSurvey extends Survey {
8     1 usage 1 mark
9     SingleChoiceSurvey(String name, ArrayList<String> teams, ArrayList<String> options, Employee owner) {
10         super(name, teams, options, owner);
11     }
12
13     1 usage 1 mark
14     @Override
15     public void vote(Employee user, ArrayList<String> optionsId) {
16         String optionsId = optionsId.getFirst();
17
18         int votesNumber = 0;
19         for (SurveyOption option: options) {
20             if (option.getId().equals(optionsId)) option.addVoter(user);
21             else option.removeVoter(user);
22             votesNumber += option.getVotersSize();
23         }
24
25         for (SurveyOption option: options) {
26             option.calculateVotesNumber();
27         }
28     }
29 }

```

- Agregáciu a zapuzdrenie si možno pozrieť na príklade triedy Main

```

14 public class Main extends Application {
15     3 usages
16     private Corporation corporation;
17     3 usages
18     private Employee currentUser;
19     2 usages
20     private SurveyManager surveyManager;
21     4 usages
22     public Survey currentSurvey;
23     7 usages
24     public Router router;
25 }

73 @
74 public void setCorporation(Corporation corp) {
75     corporation = corp;
76     surveyManager = corp.getSurveyManager();
77 }
78
79 1 usage 1 mark
80 public void setCurrentUser(Employee employee) { currentUser = employee; }
81 1 usage 1 mark
82 public void setCurrentUser(String fullName) {
83     for (Employee employee: corporation.getEmployees()) {
84         if (employee.compareEmployee(fullName)) {
85             currentUser = employee;
86             return;
87         }
88     }
89 }
90
91 5 usages 1 mark
92 public Employee getCurrentUser() { return currentUser; }
93 1 mark
94 public Corporation getCorporation() { return corporation; }
95 1 mark
96 public SurveyManager getSurveyManager() { return surveyManager; }

```

Dalšie kritériá a ich implementácia:

1. Observer návrhový vzor: Použila som Observer návrhový vzor v triede SurveyManager, ktorá udržiava zoznam všetkých odberateľov (subscribers). Keď sa zmení stav prieskumov, SurveyManager informuje každého usera o týchto zmenách.

```
public class SurveyManager {
    1 usage
    private final ArrayList<Survey> surveysArr = new ArrayList<>();
    5 usages
    private final ObservableList<Survey> surveys = FXCollections.observableList(surveysArr);
    3 usages
    private final ArrayList<SurveyObserver> surveySubscribers = new ArrayList<>();
    5 usages
    private final Main main;
    1 marik
    public SurveyManager(Main main) {
        this.main = main;
    }
    // subscribe a user for surveys update;
    1 usage 1 marik
    public void subscribe(SurveyObserver subscriber) { surveySubscribers.add(subscriber); }
    // unsubscribe a user for surveys update;
    // this method is not used now due to non-existence of user deletion.
    no usages 1 marik
    public void unsubscribe(SurveyObserver subscriber) { surveySubscribers.remove(subscriber); }
}
```

```
2 usages 1 marik
private void inform(Survey survey) {
    for (SurveyObserver subscriber: surveySubscribers) {
        if (survey.containsEmployee((Employee) subscriber) || survey.checkOwner((Employee) subscriber)) {
            ((Employee) subscriber).update();
        }
    }
}
}
```

User:

```
87 > public ArrayList<Survey> getPersonalSurveys() { return personalSurveys; }
1 usage 1 marik
90 public ArrayList<Survey> getOpenedSurveys() { return surveys; }
91
92 1 usage 1 marik
93 public void setupOpenedSurveys() { this.surveys = surveyManager.getEmployeeSurveys((User) this); }
94
95 1 usage 1 marik
96 public void setupPersonalSurveys() { this.personalSurveys = surveyManager.getPersonalSurveys((User) this); }
97
2 usages 1 marik
100 @
101 public void update() {
102     setupOpenedSurveys();
103     setupPersonalSurveys();
104 }
}
```

2. Strategy návrhový vzor: Strategy návrhový vzor bol implementovaný v triede Validator. Validator prijíma dve stratégie pre validáciu polí na prázdne hodnoty a na dĺžku. Vybraná stratégia sa potom používa na validáciu všetkých polí v zozname. (Validator sa využíva v triede NewSurveyController v metóde createNewSurvey).

```
Validator.java x NewSurveyController.java
1 package validation;
2
3 4 usages 1 marik *
4 public class Validator<T> {
5     4 usages
6     private final ErrorHandler<T> validator;
7     2 usages 1 marik
8     public Validator(ErrorHandler<T> handler) { this.validator = handler; }
9
10    2 usages 1 marik
11    public void validate() { validator.validate(); }
12
13    1 marik
14    public void add(T field) { this.validator.add(field); }
15    2 usages 1 marik
16    public void delete(T field) { this.validator.delete(field); }
17
18 }
19
20
```

```
Validator.java x NewSurveyController.java
1 public void setupPage() {
2     this.emptyFieldValidator = new Validator<ValidationComponent>(new EmptyValidationValidationComponent());
3     this.shortFieldValidator = new Validator<ValidationComponent>(new ShortValidationValidationComponent());
4
5     ObservableList<String> items = FXCollections.observableArrayList(
6         "product", "manager", "design", "frontend", "backend", "tester"
7     );
8     teamsBox.setItems(items);
9     teamsBox.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
10    teamsBox.setName("Teams field");
11    emptyFieldValidator.add(teamsBox);
12
13    surveyNameField.setName("Survey field name");
14    emptyFieldValidator.add(surveyNameField);
15    shortFieldValidator.add(surveyNameField);
16
17    surveyOptionsField.setName("Options box");
18    emptyFieldValidator.add(surveyOptionsField);
19 }
20
21 1 marik
22 @FXML
23 private void createSurvey(ActionEvent event) throws IOException {
24     try {
25         emptyFieldValidator.validate();
26         shortFieldValidator.validate();
27
28         String name = surveyNameField.getText();
29     }
30 }
```

3. Vlastné výnimky: Implementovala som vlastné výnimky, ako ShortException a EmptyException, ktoré sú používané v triedach ShortValidation a EmptyValidation.

```
EmptyException.java x EmptyValidation.java
1 package validation;
2
3 import validation.components.ValidationComponent;
4
5 3 usages 1 marik
6 public class EmptyException extends CustomFieldException {
7     3 usages 1 marik
8     public EmptyException(String errMsg, ValidationComponent emptyField) { super(errMsg + " is empty", emptyField); }
9 }
10
```

```
EmptyException.java x EmptyValidation.java
1 package validation;
2
3 import
4
5 1 usage 1 marik
6 public class EmptyValidation<T> implements ErrorHandler<T> {
7     4 usages
8     private ArrayList<T> validationFields;
9
10    1 usage 1 marik
11    public EmptyValidation() { this.validationFields = new ArrayList<>(); }
12
13    1 usage 1 marik
14    @Override
15    public void validate() { validationFields.forEach(this::startValidation); }
16
17    1 usage 1 marik
18    private void startValidation(T field) {
19        if (field instanceof TextField textField) {
20            textField.unhighlight();
21            String fieldText = textField.getText();
22            if (fieldText.isEmpty()) throw new EmptyException(textField.getName(), textField);
23        }
24        else if (field instanceof ListView listView) {
25            listView.unhighlight();
26            boolean isEmpty = listView.getSelectionModel().getSelectedIndices().isEmpty();
27            if (isEmpty) throw new EmptyException(listView.getName(), listView);
28        }
29        else if (field instanceof VBox vbox) {
30            vbox.unhighlight();
31            boolean isEmpty = vbox.getChildren().isEmpty();
32            if (isEmpty) throw new EmptyException(vbox.getName(), vbox);
33        }
34    }
35 }
```

4. Grafické používateľské rozhranie (GUI): Oddelila som grafické používateľské rozhranie od aplikačnej logiky a handlers. Všetko čo patrí do GUI sú v pages package.

5. Generické triedy: Validátory sú príkladom generických tried, ktoré umožňujú ošetrovať rôzne typy polí implementované pomocou rozhrania ValidationComponent.

```

package validation;

import java.util.*;

1 usage & mark
public class EmptyValidation<T> implements ErrorHandler<T> {
    4 usages
    private ArrayList<T> validationFields;

    1 usage & mark
    public EmptyValidation() { this.validationFields = new ArrayList<>(); }

    1 usage & mark
    @Override
    17 <?> public void validate() { validationFields.forEach(this::startValidation); }

    1 usage & mark
    private void startValidation(T field) {
        22
        if (field instanceof CTextField textField) {
            23
            textField.unhighlight();
            24
            String fieldText = textField.getText();
            25
            if (fieldText.isEmpty()) throw new EmptyException(textField.getName(), textField);
        }
        26
        else if (field instanceof CListView listView) {
            27
            listView.unhighlight();
            28
            boolean isEmpty = listView.getSelectionModel().getSelectedIndices().isEmpty();
            29
            if (isEmpty) throw new EmptyException(listView.getName(), listView);
        }
        30
        else if (field instanceof CVBox vbox) {
            31
            vbox.unhighlight();
            32
            boolean isEmpty = vbox.getChildren().isEmpty();
            33
            if (isEmpty) throw new EmptyException(vbox.getName(), vbox);
        }
    }
}

```

```

4 usages
private Main main;

7 usages
private Validator<ValidationComponent> emptyFieldValidator;
5 usages
private Validator<ValidationComponent> shortFieldValidator;

4 mark *
40 <?> public void setMain(Main main) { }

5 usages & mark *
45 <?> public void setupPage() {
    46
    this.emptyFieldValidator = new Validator<ValidationComponent>(new EmptyValidation<ValidationComponent>());
    47
    this.shortFieldValidator = new Validator<ValidationComponent>(new ShortValidation<ValidationComponent>());

    48
    ObservableList<String> items = FXCollections.observableArrayList(
        49
        "product", "manager", "design", "frontend", "backend", "tester"
    );
    50
    teamsBox.setItems(items);
    51
    teamsBox.getSelectionModel().setSelectionMode(javafx.scene.control.SelectionMode.MULTIPLE);
    52
    teamsBox.setName("Teams field");
    53
    emptyFieldValidator.add(teamsBox);
    54
    surveyNameField.setName("Survey field name");
    55
    emptyFieldValidator.add(surveyNameField);
    56
    shortFieldValidator.add(surveyNameField);
    57
    surveyOptionsField.setName("Options box");
    58
    emptyFieldValidator.add(surveyOptionsField);
    59
}

```

6. RTTI (Run-Time Type Identification): Použila som RTTI na zistenie spôsobu ošetrovania polí, napríklad v metóde startValidation triedy EmptyValidation.

```

1 usage & mark
public class EmptyValidation<T> implements ErrorHandler<T> {
    10
    private ArrayList<T> validationFields;

    11
    1 usage & mark
    public EmptyValidation() { this.validationFields = new ArrayList<>(); }

    12
    1 usage & mark
    @Override
    17 <?> public void validate() { validationFields.forEach(this::startValidation); }

    18
    1 usage & mark
    private void startValidation(T field) {
        21
        if (field instanceof CTextField textField) {
            22
            textField.unhighlight();
            23
            String fieldText = textField.getText();
            24
            if (fieldText.isEmpty()) throw new EmptyException(textField.getName(), textField);
        }
        25
        else if (field instanceof CListView listView) {
            26
            listView.unhighlight();
            27
            boolean isEmpty = listView.getSelectionModel().getSelectedIndices().isEmpty();
            28
            if (isEmpty) throw new EmptyException(listView.getName(), listView);
        }
        29
        else if (field instanceof CVBox vbox) {
            30
            vbox.unhighlight();
            31
            boolean isEmpty = vbox.getChildren().isEmpty();
            32
            if (isEmpty) throw new EmptyException(vbox.getName(), vbox);
        }
    }
}

```

7. Lambda výrazy a referencie na metódy: Využila som lambda výrazy a referencie na metódy na zjednodušenie určitých častí kódu.

```

// send to a user his surveys (surveys where he can vote)
1 usage & mark *
public ArrayList<Survey> getEmployeeSurveys(Employee user) {
    ArrayList<Survey> employeeSurveys = new ArrayList<>();
    surveys.forEach(survey -> {
        if (survey.containsEmployee(user)) employeeSurveys.add(survey);
    });
    return employeeSurveys;
}

// send to a user his personal surveys (surveys where he is owner)
1 usage & mark *
public ArrayList<Survey> getPersonalSurveys(Employee user) {
    ArrayList<Survey> personalSurveys = new ArrayList<>();
    surveys.forEach(survey -> {
        if (survey.checkOwner(user)) personalSurveys.add(survey);
    });
    return personalSurveys;
}

```

```

System.out.print("Opened surveys " + openedSurveys + "\n");
openedSurveysBox.getChildren().clear();
if (openedSurveysBox != null) {
    openedSurveys.forEach(this::createSurveyField);
}

System.out.print("Personal surveys " + personalSurveys + "\n");
personalSurveysBox.getChildren().clear();
if (personalSurveys != null) {
    personalSurveys.forEach(this::createPersonalSurveyField);
}

```

8. Použitie rozhraní: V projekte som použila rozhrania ako Controller v triede Router, SurveyObserver v triede SurveyManager a ErrorHandler v stratégii Validator.

Verzií projektu

V prvej pracovnej verzii projektu sa zaoberala implementáciou základnej funkcionality, ako je vytváranie používateľov, prepínač medzi nimi a možnosť vytvárania rôznych typov prieskumov, ako sú jedno a viacnásobné výbery. Tiež bola pridaná funkcia hlasovania v prieskumoch.

Druhá verzia projektu sa zamerala na náročnejšie funkcie a vylepšenia. Bola pridaná funkcionality validácie polí, ako aj možnosť zmazania prieskumov. Okrem toho bola pridaná možnosť zobrazit' výsledky prieskumov pomocou grafov pre udržiavateľa. Bola vylepšená funkcionality pomocou dvoch design patternov: strategy a observer. Nakoniec som vymazala všetky statické metódy z triedy Main, a rozhodla som inštanciu main preposielať medzi triedami pomocou Router.