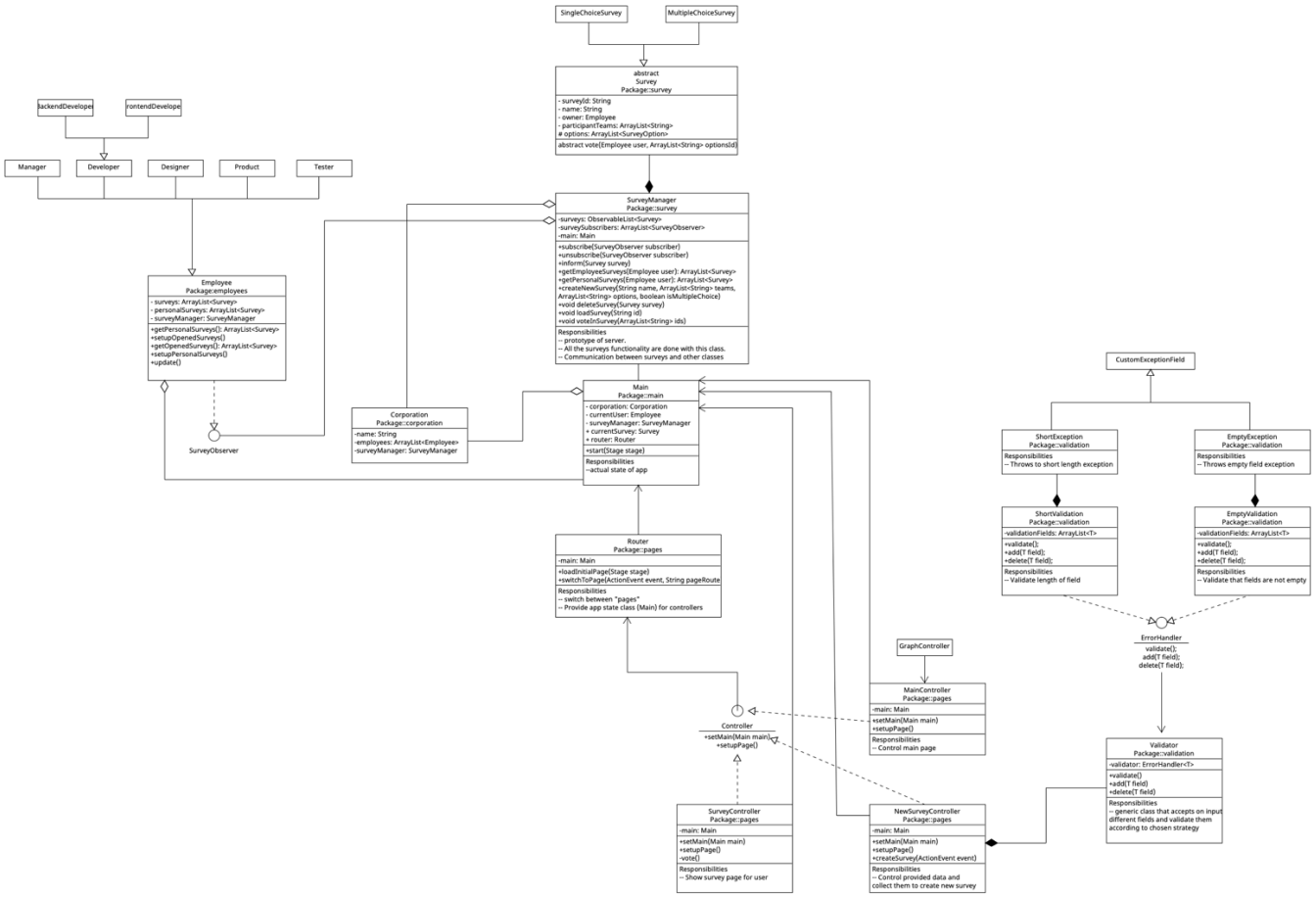
Mariia Portnykh
Názov projektu: TeamSurveyManager

Zámer projektu:
Cieľom môjho projektu je navrhnúť 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 obmedzene 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.

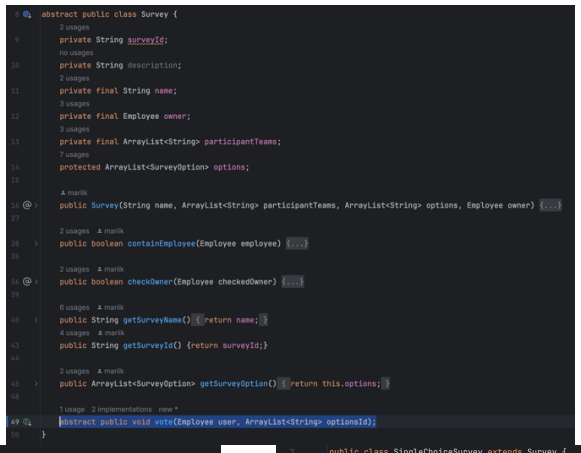
Ako to funguje?
Na začiatku treba vstúpiť do systému vašej spoločnosti, kde pracujete, alebo zaregistrovať svoju vlastnú firmu. Systém obsahuje zoznam všetkých zamestnancov spoločnosti, vrátane ich prax, odbor (či tím) a iné relevantné informácií. Ľubovoľný človek zo zoznamu môže vytvoriť hlasovanie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

UML diagram:

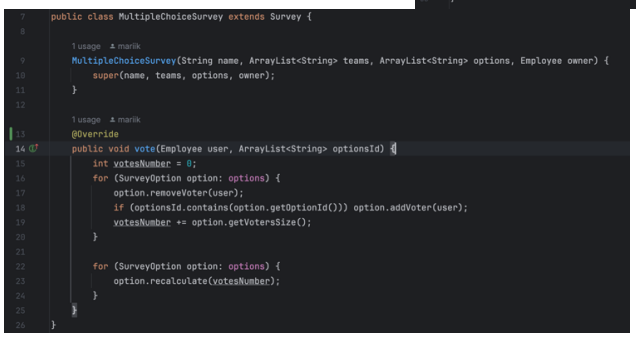


Hlavné kritériá:

- 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.



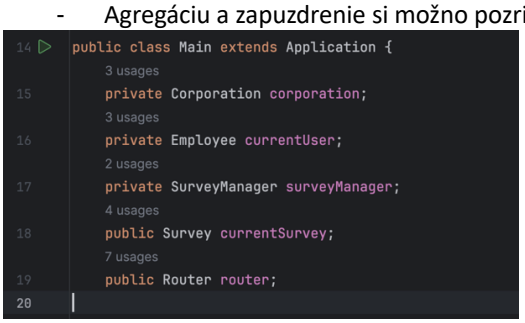
```
abstract public class Survey {
    2 usages
    private String surveyId;
    no usages
    private String description;
    2 usages
    private final String name;
    3 usages
    private final Employee owner;
    3 usages
    private final ArrayList<String> participantTeams;
    7 usages
    protected ArrayList<SurveyOption> options;

    1 usage  2 implementations  new *
    abstract public void vote(Employee user, ArrayList<String> optionsId);
}
```



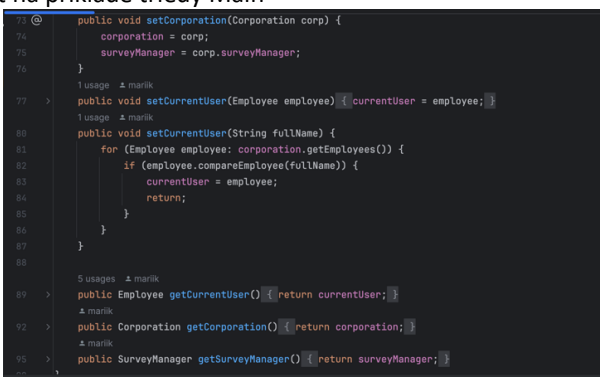
```
public class MultipleChoiceSurvey extends Survey {

    1 usage  mariik
    MultipleChoiceSurvey(String name, ArrayList<String> teams, ArrayList<String> options, Employee owner) {
        super(name, teams, options, owner);
    }

    1 usage  mariik
    @Override
    public void vote(Employee user, ArrayList<String> optionsId) {
        int votesNumber = 0;
        for (SurveyOption option: options) {
            option.removeVoter(user);
            if (optionsId.contains(option.getOptionId())) option.addVoter(user);
            votesNumber += option.getVotersSize();
        }

        for (SurveyOption option: options) {
            option.recalculate(votesNumber);
        }
    }
}
```

```
public class SingleChoiceSurvey extends Survey {
    1 usage  mariik
    SingleChoiceSurvey(String name, ArrayList<String> teams, ArrayList<String> options, Employee owner) {
        super(name, teams, options, owner);
    }

    1 usage  mariik
    @Override
    public void vote(Employee user, ArrayList<String> optionsId) {
        String optionId = optionsId.getFirst();

        int votesNumber = 0;
        for (SurveyOption option: options) {
            if (option.getOptionId().equals(optionId)) option.addVoter(user);
            else option.removeVoter(user);
            votesNumber += option.getVotersSize();
        }

        for (SurveyOption option: options) {
            option.recalculate(votesNumber);
        }
    }
}
```

- Agregáciu a zapuzdrenie si možno pozrieť na príklade triedy Main



```
public class Main extends Application {
    3 usages
    private Corporation corporation;
    3 usages
    private Employee currentUser;
    2 usages
    private SurveyManager surveyManager;
    4 usages
    public Survey currentSurvey;
    7 usages
    public Router router;
```



```
public void setCorporation(Corporation corp) {
    corporation = corp;
    surveyManager = corp.surveyManager;
}
1 usage  mariik
public void setCurrentUser(Employee employee) { currentUser = employee; }
1 usage  mariik
public void setCurrentUser(String fullName) {
    for (Employee employee: corporation.getEmployees()) {
        if (employee.compareEmployee(fullName)) {
            currentUser = employee;
            return;
        }
    }
}

5 usages  mariik
public Employee getCurrentUser() { return currentUser; }
mariik
public Corporation getCorporation() { return corporation; }
mariik
public SurveyManager getSurveyManager() { return surveyManager; }
```

Ďalšie 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.

```java
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;
    mariik
    public SurveyManager(Main main) {
        this.main = main;
    }
    // subscribe a user for surveys update;
    1 usage  mariik
    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  mariik
    public void unsubscribe(SurveyObserver subscriber) { surveySubscribers.remove(subscriber); }
```

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

User:

```java
    public ArrayList<Survey> getPersonalSurveys() { return personalSurveys; }
    1 usage  mariik
    public ArrayList<Survey> getOpenedSurveys() { return surveys; }

    1 usage  mariik
    public void setupOpenedSurveys() { this.surveys = surveyManager.getEmployeeSurveys( user: this); }

    1 usage  mariik
    public void setupPersonalSurveys() { this.personalSurveys = surveyManager.getPersonalSurveys( user: this); }

    2 usages  mariik
    public void update() {
        setupOpenedSurveys();
        setupPersonalSurveys();
    }
}
```

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).

```java
package validation;

4 usages  mariik *
public class Validator<T> {
    4 usages
    private final ErrorHandler<T> validator;
    2 usages  mariik
    public Validator(ErrorHandler<T> handler) { this.validator = handler; }

    2 usages  mariik
    public void validate() { validator.validate(); }

    mariik
    public void add(T field) { this.validator.add(field); }
    2 usages  mariik
    public void delete(T field) { this.validator.delete(field); }
}
```

```java
public void setMain(Main main) {
    this.main = main;
    setupPage();
}

5 usages  mariik *
public void setupPage() {
    this.emptyFieldValidator = new Validator<ValidationComponent>(new EmptyValidation<ValidationComponent>());
    this.shortFieldValidator = new Validator<ValidationComponent>(new ShortValidation<ValidationComponent>());

    ObservableList<String> items = FXCollections.observableArrayList(
            ...es: "product", "manager", "design", "frontend", "backend", "tester"
    );
    teamsBox.setItems(items);
    teamsBox.getSelectionModel().setSelectionMode(javafx.scene.control.SelectionMode.MULTIPLE);
    teamsBox.setName("Teams field");
    emptyFieldValidator.add(teamsBox);

    surveyNameField.setName("Survey field name");
    emptyFieldValidator.add(surveyNameField);
    shortFieldValidator.add(surveyNameField);

    surveyOptionsField.setName("Options box");
    emptyFieldValidator.add(surveyOptionsField);
}

mariik
@FXML
private void createSurvey(ActionEvent event) throws IOException {
    try {
        errorMsg.setText("");
        emptyFieldValidator.validate();
        shortFieldValidator.validate();

        String name = surveyNameField.getText();
```

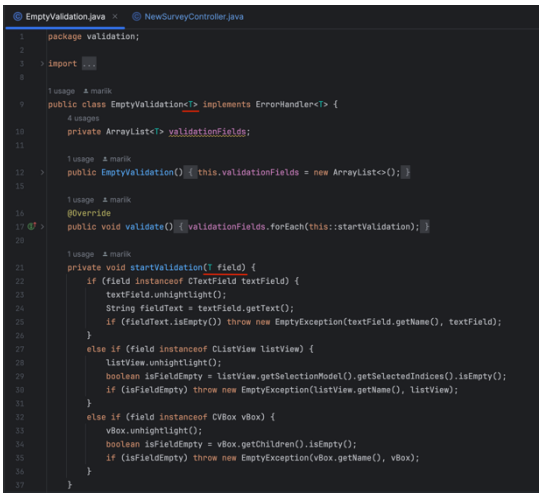
3. Vlastné výnimky: Implementovala som vlastné výnimky, ako ShortException a EmptyException, ktoré sú používané v triedach ShortValidation a EmptyValidation.

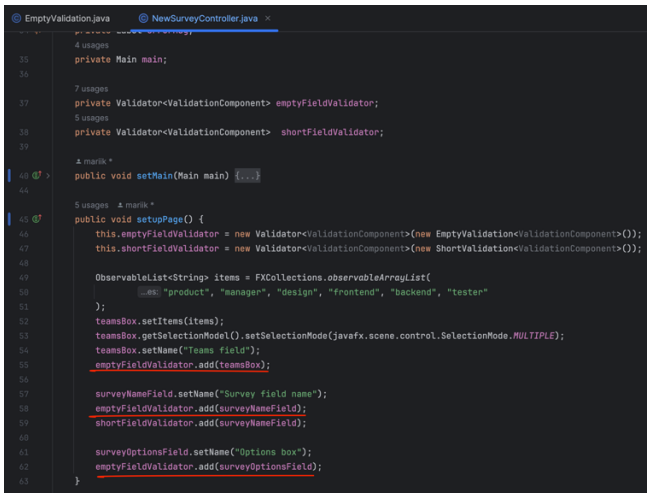
```java
package validation;

import validation.components.ValidationComponent;

3 usages  mariik
public class EmptyException extends CustomFieldException {
    3 usages  mariik
    public EmptyException(String errMsg, ValidationComponent emptyField) { super(errMsg + " is empty", emptyField); }
}
```

```java
package validation;

import ...

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

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

    1 usage  mariik
    @Override
    public void validate() { validationFields.forEach(this::startValidation); }

    1 usage  mariik
    private void startValidation(T field) {
        if (field instanceof CTextField textField) {
            textField.unhightlight();
            String fieldText = textField.getText();
            if (fieldText.isEmpty()) throw new EmptyException(textField.getName(), textField);
        }
        else if (field instanceof CListView listView) {
            listView.unhightlight();
            boolean isFieldEmpty = listView.getSelectionModel().getSelectedIndices().isEmpty();
            if (isFieldEmpty) throw new EmptyException(listView.getName(), listView);
        }
        else if (field instanceof CVBox vBox) {
            vBox.unhightlight();
            boolean isFieldEmpty = vBox.getChildren().isEmpty();
            if (isFieldEmpty) throw new EmptyException(vBox.getName(), vBox);
        }
    }
```

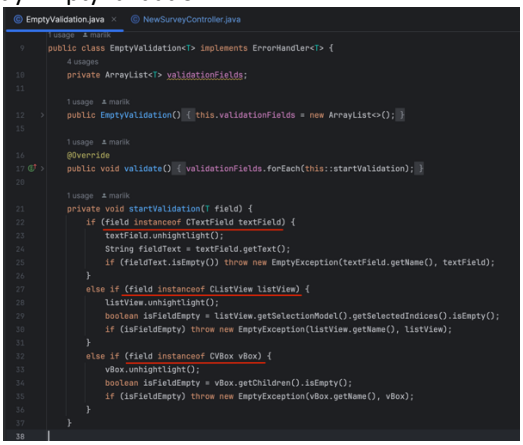
4. Grafické používateľské rozhranie (GUI): Oddelila som grafické používateľské rozhranie od aplikačnej logiky a handlers. Všetko čo patrí do GUI sú v pages package.

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

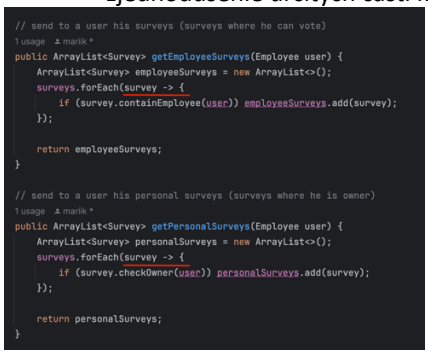


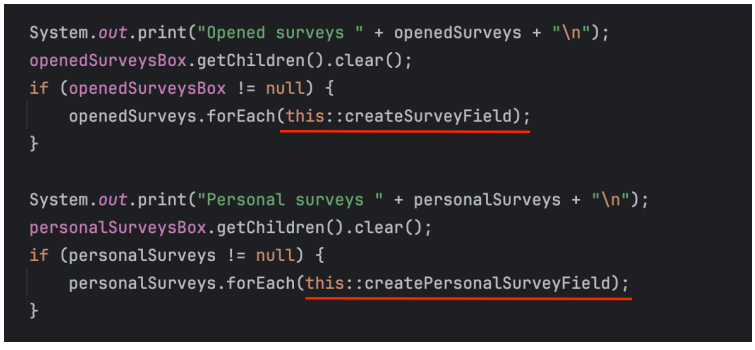


6. RTTI (Run-Time Type Identification): Použila som RTTI na zistenie spôsobu ošetrovania polí, napríklad v metóde startValidation triedy EmptyValidation.



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





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 funkcionalita validácie polí, ako aj možnosť zmazania prieskumov. Okrem toho bola pridaná možnosť zobraziť výsledky prieskumov pomocou grafov pre udržiavateľa. Bola vylepšená funkcionalita 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.