**Main Report**

for

**VWS: Volunteer Work System**

For the Department of Voluntary at IUG

*Submitted to*

Dr. Abdelkareem Alashqar

*by*

Mahmoud M. K. Abusaqer – 120180194

Amro Nasser Okasha – 120181573

October 2020

**Abstract**

This report presents product software requirements, system design, implementation, and testing, for developing a Volunteer work system – VWS, for the Islamic University of Gaza. The system was made to handle all the volunteer work online, it is especially for students to help them finish all the hours required for them from the Department of Volunteer (DOV) in an easy way. The DOV will control the system and approve or deny all the requests from students or institutions according to their rules.

The VWS system will provide many capabilities to the student and institution. For the student, he/she can choose the place that he/she wants to volunteer int it, create the initiative, and suggest institutions. For institutions, it can see all the statistics about the volunteer work in their system, see all the new volunteers, and end reports to the university when a student finishes the volunteer work.

The intended audience for this report includes all the IUG Student, DOV, and Institutions that are interested or does volunteer work.

**Table of Contents**

1. Chapter 1. Document Software Requirements

1.1 System Overall Description

1.2 System Use Case Diagram

1.3 System Specific Requirements

1.4 List of classes

1.5 Non-Functional Requirements

2. Chapter 2. System Design and Architecture

2.1 Application Architecture

2.2 Inter-Package Dependencies

2.3 Data Decomposition

2.4 Packages Detailed Design

3. Chapter 3. System Implementation and Testing

3.1 Implementation and Coding Standards

3.2 Software Testing

4. References

**Chapter 1. Document Software Requirements for Volunteer Work System**

**1.1 System Overall Description**

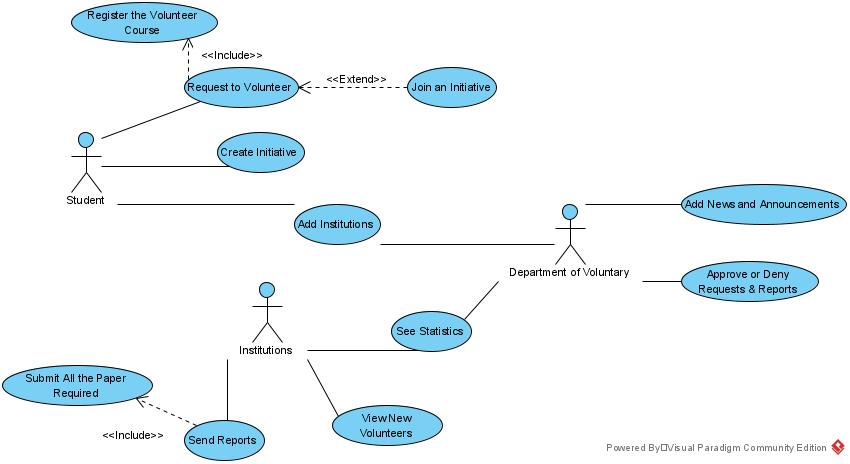
The Volunteer Work System is a system for students, the Department of Voluntary at IUG, and Institutions. To help get and finish the volunteer work in an easy way and online, without going to the Department of Voluntary Work at the university and using papers and letters to communicate.

The student will be able to choose the place to volunteer in and send a request to that institution, he/she can create initiative and also will be able to suggest institutions too.

Institutions will see all the new forms form students who requested to volunteer in it, send reports to the university when a student finishes all the hours required to pass the volunteer work course, and it will have statistics about the number of students who are still volunteering, finished, and requesting to volunteer there.

The University will keep tracking of all the operation that happens in the system, it will approve or deny the requests for students to volunteer in an institution, the requests of institutions to be in the system, and the reports of the students who finished the volunteer work, it will have statistics for all the number of students who still volunteering, finished, requesting to volunteer and need approve of finishing the volunteer work, put news and announcements about the department, and it can be able to add new institutions too.

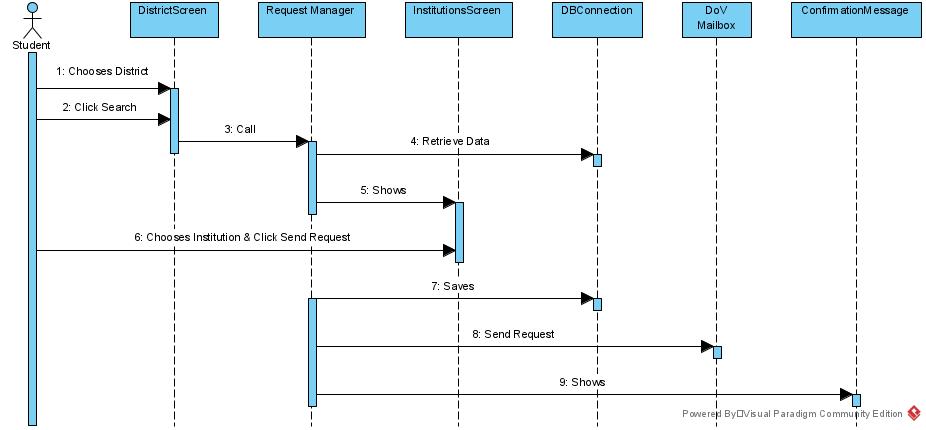
**1.2 System Use Case Diagram**



**1.3 System Specific Requirements**

**1.3.1 Function Scenario for Request to Volunteer**

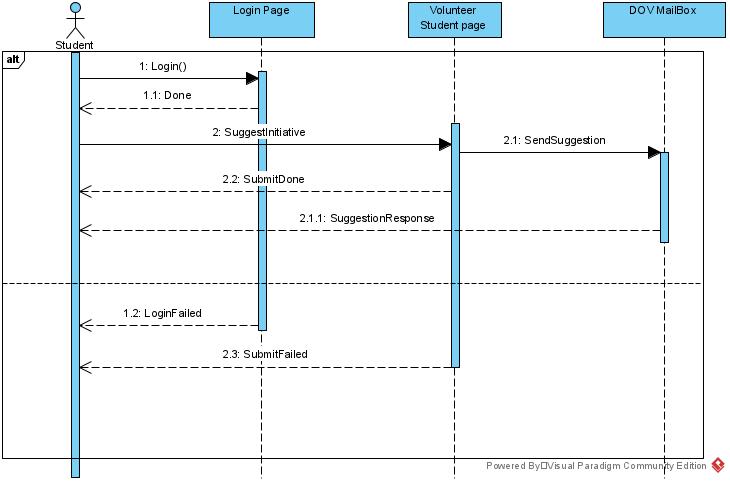
|  |  |  |
| --- | --- | --- |
| ID | 1 |  |
| Name | | Request to Volunteer |
| Description | | The student chooses a place to volunteer in and send a request to the DoV then if the DoV accepts the request it sends a notification to the institution |
| **Priority** | | Must have (1) |
| **Actors** | | Student and DoV |
| **Pre-Conditions** | | * Student login to the system |
| **Flow of Events** | |  |
| Normal flow | | 1. The student chooses the district click the search request button 2. The system shows all the institution in that district 3. The student selects the institution and click the send request button 4. The system saves the selected institution to the database 5. The system sends the request to the DoV mailbox 6. The system shows a success message that the request did reach the DoV |
| Alternative flow | | 1.A The system shows an alert “No district was selected”  2.A The system shows a blank page if there is no connection with the database  3.A The system shows an alert “No institution was selected”  4.A The system shows an error message no connection with the database  5.A The system shows an error message that the request did not reach the DoV |
| **Post-Conditions** | | * The request was successfully sent to the DoV * The student will wait for the approval or denial of the request |

**1.3.1.1 Sequence Diagram for Request to Volunteer**

**1.3.2 Function Scenario for Create Initiative.**

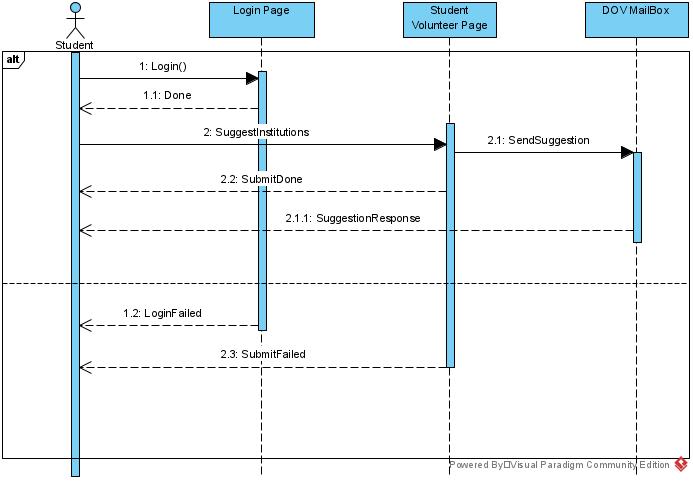
|  |  |  |
| --- | --- | --- |
| ID | 2 |  |
| Name | | Create Initiative. |
| Description | | The student can suggest to add a new Initiative to the system and Department of voluntary who will approve or deny the request. |
| **Priority** | | Could have (3) |
| **Actors** | | Student. |
| **Pre-Conditions** | | * Student must be logged in the system. |
| **Flow of Events** | |  |
| Normal flow | | 1. The student will log in into system. 2. The system will let him in. 3. The student will press the Create Initiative button and enter their Initiative information then press submit. 4. The system will receive the application. |
| Alternative flow | | 1. The student fails to log in due wrong info or server problem. 2. Submit button doesn't work. |
| **Post-Conditions** | | * The submission being sent to Department of voluntary. |

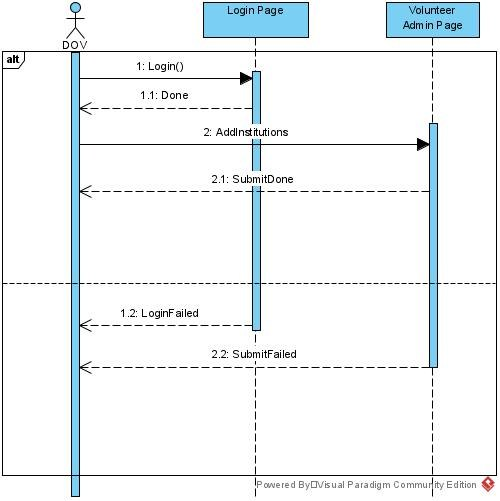
**1.3.2.1 Sequence Diagram for Create Initiative.**



**1.3.3 Function Scenario for Add Institutions.**

|  |  |  |
| --- | --- | --- |
| ID | 3 |  |
| Name | | Add Institutions. |
| Description | | The student or DOV can suggest to add a new volunteer Institution to the system and DOV Suggestions Center who will approve or deny the request. |
| **Priority** | | Could have (3) |
| **Actors** | | Student and Department of voluntary. |
| **Pre-Conditions** | | * Student must be logged in the system. |
| **Flow of Events** | |  |
| Normal flow | | 1. The student or DOV admin will log in into system. 2. The system will let him in. 3. The student or DOV will press the add institutions button and enter institution information then press submit. 4. The system will receive the application. |
| Alternative flow | | 1. The student fails to log in due wrong info or server problem. 2. Submit button doesn't work. |
| **Post-Conditions** | | * The submission by student should being sent to Department of voluntary. * The submission by DOV should let the new institution appear to student page. |

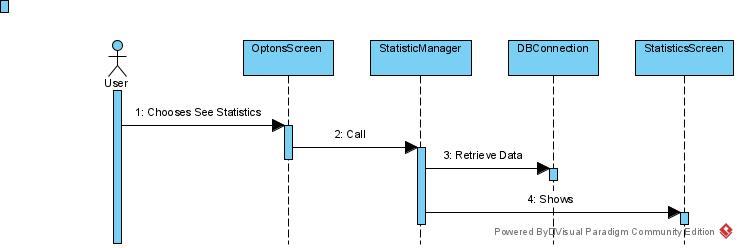
**1.3.3.1 Sequence Diagram for Add Institutions.**



**1.3.4 Function Scenario for See Statistics**

|  |  |  |
| --- | --- | --- |
| ID | 4 |  |
| Name | | See Statistics |
| Description | | It shows all the statistics that the DoV or the institution will need like the number of new volunteers, the number of who still volunteering, … etc. |
| **Priority** | | Could have (3) |
| **Actors** | | DoV and Institutions |
| **Pre-Conditions** | | * Go to the statistics page |
| **Flow of Events** | |  |
| Normal flow | | 1. The user selects see statistics from the options screen 2. The system shows the results |
| Alternative flow | | 2.A The system shows a blank page if there is no connection with the database |
| **Post-Conditions** | | * The user can see all statistics |

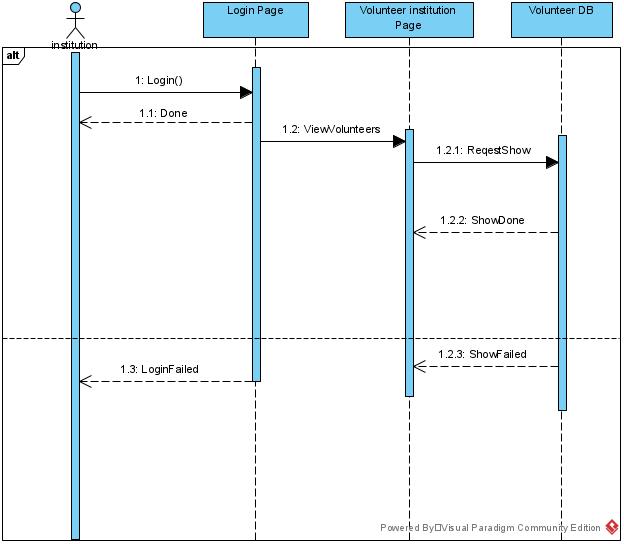
**1.3.4.1 Sequence Diagram for See Statistics**



**1.3.5 Function Scenario for View New Volunteers.**

|  |  |  |
| --- | --- | --- |
| ID | 5 |  |
| Name | | View New Volunteers. |
| Description | | The institutions will have the ability to view their new volunteers and make records to every one of them. |
| **Priority** | | Should have (2) |
| **Actors** | | Institutions. |
| **Pre-Conditions** | | * Institution user must be logged in Institutions volunteer page. |
| **Flow of Events** | |  |
| Normal flow | | 1. The institution will log in into system. 2. The system will let him in. 3. The institution will request to view the new volunteers by clicking on View new volunteers' button. 4. All new volunteers will appear in the page so he can manage them. |
| Alternative flow | | 1. The institution fails to log in due wrong info or server problem. 2. View button doesn't work. 3. View button work but no volunteers showed. |
| **Post-Conditions** | | * Volunteers will have records and information in institution own system. |

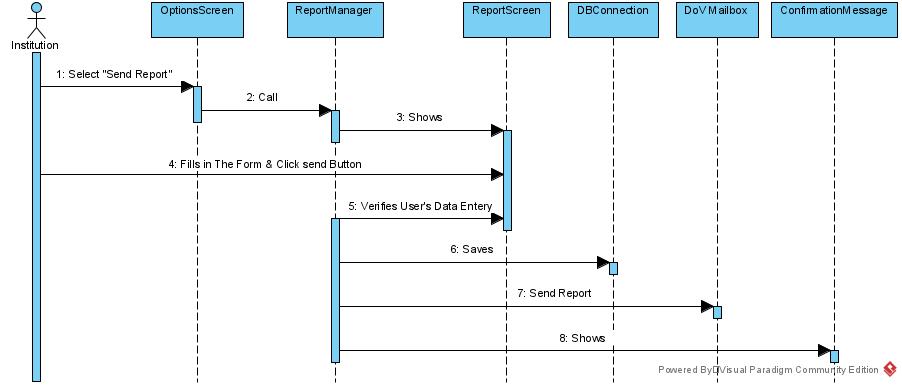
**1.3.5.1 Sequence Diagram for View New Volunteers.**



**1.3.6 Function Scenario for Send Reports**

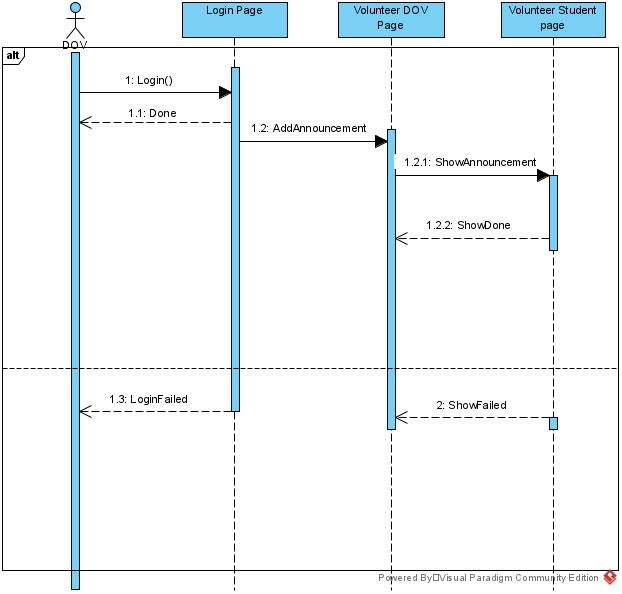
|  |  |  |
| --- | --- | --- |
| ID | 6 |  |
| Name | | Send Reports |
| Description | | Institutions will fill the end of the volunteer work form for students who finished it and send the report to the DoV. |
| **Priority** | | Must have (1) |
| **Actors** | | Institutions, DoV and Student |
| **Pre-Conditions** | | * Go to the send reports page * Having all the information that the form will need. * Making sure that the student has successfully finished all the hours required to pass the course. |
| **Flow of Events** | |  |
| Normal flow | | 1. The user selects “Send Report” from the option screen 2. The system shows the form 3. The user fills in the form then click the send button 4. The system verifies the user’s data entry. 5. The system saves the information into the database 6. The system sends the report to the DoV mailbox 7. The system shows a success message that the request did reach the DoV |
| Alternative flow | | 3.A The system shows an alert “Some filed not filled up”  4.A & 5.A The system shows an error message no connection with the database  6.A The system shows an error message that the request did not reach the DoV |
| **Post-Conditions** | | * The report was successfully sent to the DoV * The institution will wait for the approval or denial of the report * The Student will be notified if report was accepted |

**1.3.6.1 Sequence Diagram for Send Reports**



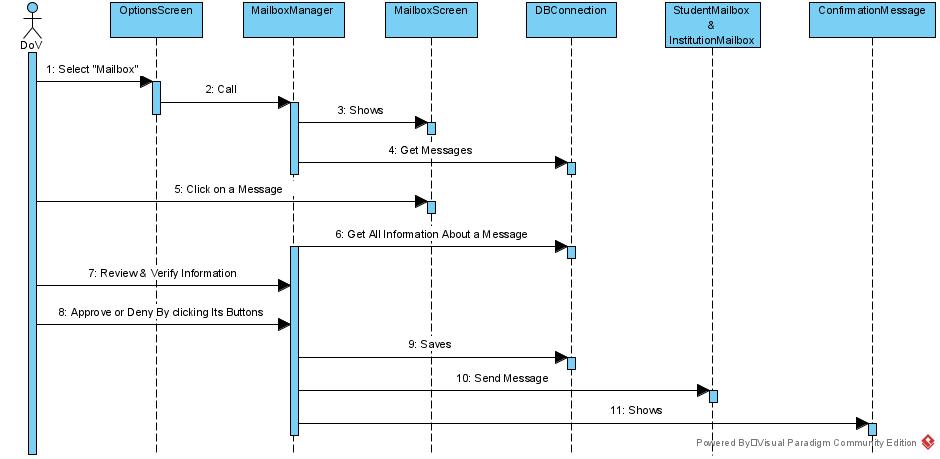
**1.3.7 Function Scenario for Add News and Announcement.**

|  |  |  |
| --- | --- | --- |
| ID | 7 |  |
| Name | | Add News and Announcement. |
| Description | | DOV can add a volunteer Announcement and news for a new volunteer opportunity. |
| **Priority** | | Should have (2) |
| **Actors** | | Department of voluntary. |
| **Pre-Conditions** | | * User (Admin) must be logged in DOV Volunteer admin page. |
| **Flow of Events** | |  |
| Normal flow | | 1. The admin will log in into system. 2. The system will let him in. 3. The admin will press the add new button Announcement button and enter needed information then press Add. 4. The Announcement should appear in student's volunteer page. |
| Alternative flow | | 1. The admin fails to log in due wrong info or server problem. 2. Submit button doesn't work. 3. The Announcement or news will not appear in the student page. |
| **Post-Conditions** | | * The new announcement and news should appear to student page. |

**1.3.7.1 Sequence Diagram for Add News and Announcement.**

**1.3.8 Function Scenario for Approve or Deny Requests & Reports**

|  |  |  |
| --- | --- | --- |
| ID | 8 |  |
| Name | | Approve or Deny Requests & Reports |
| Description | | The DoV will see all the request and reports from students and institutions, it will see all new request to start the volunteer work, and where it will see all the reports from institutions and make sure that the form is correct before giving the student the pass on the volunteer work course, and both the student and institution will get notified if the request or the report was approved or denied. |
| **Priority** | | Must have (1) |
| **Actors** | | DoV, Student and Institutions |
| **Pre-Conditions** | | * Go to the mailbox |
| **Flow of Events** | |  |
| Normal flow | | 1. The user selects “Mailbox” from the option screen 2. The system shows all requests and reports 3. The user clicks on one of the messages 4. The system shows full information about the message and the user can see and review the information that is provided with the message 5. The user can approve if every thing is good by clicking the approve button or deny if there is something wrong by clicking the deny button 6. The system saves the information into the database 7. The system sends a message to the student, institution, or both mailbox 8. The system shows a success alert that the message was delivered |
| Alternative flow | | 2.A, 3A, 4A, 5A & 6A The system shows a blank page if there is no connection with the database  7.A The system shows an error message that the message wasn’t delivered |
| **Post-Conditions** | | * The student will begin the volunteer work or add another request * The student will pass the volunteer work course * The institution will review the message and make sure that the form is complete and the student finished every thing if the report was denied |

**1.3.8.1 Sequence Diagram for Approve or Deny Requests & Reports**

**1.4 List of classes**

**1.4.1 Interface (Boundary) classes**

DistrictScreen, InstitutionsScreen, DBConnection, CreateInitiative, AddInstitutions, OptionsScreenDoV, StatisticsScreen, OptionsScreenInstitution, ViewNewVolunteersScreen, ReportScreen, AddNewsAndAnnouncementScreen, DoVMailbox, MailboxScreen, StudentMailbox, InstitutionMailbox, and LoginPage.

**1.4.2 Entity classes**

Student, DoV, Institutions, District, StudentRequest, CreateInitiativeDB, AddInstitutionsDB, Statistics, ViewNewVolunteersDB, SendReportsDB, AddNewsAndAnnouncementDB, StudentsWhoFinishedVolunteer.

**1.4.3 Control (Manager) classes**

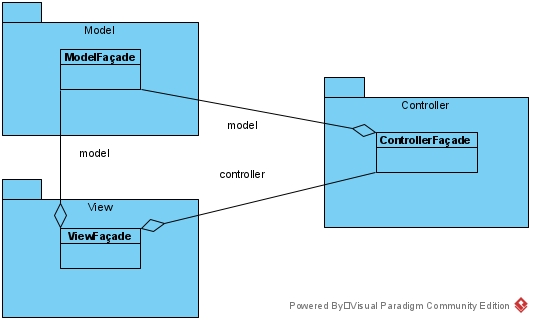
RequestManager, CreateInitiativeManager, AddInstitutionsManager, StatisticManager, ViewNewVolunteersManager, ReportManager, AddNewsAndAnnouncementManager, and MailboxManager.

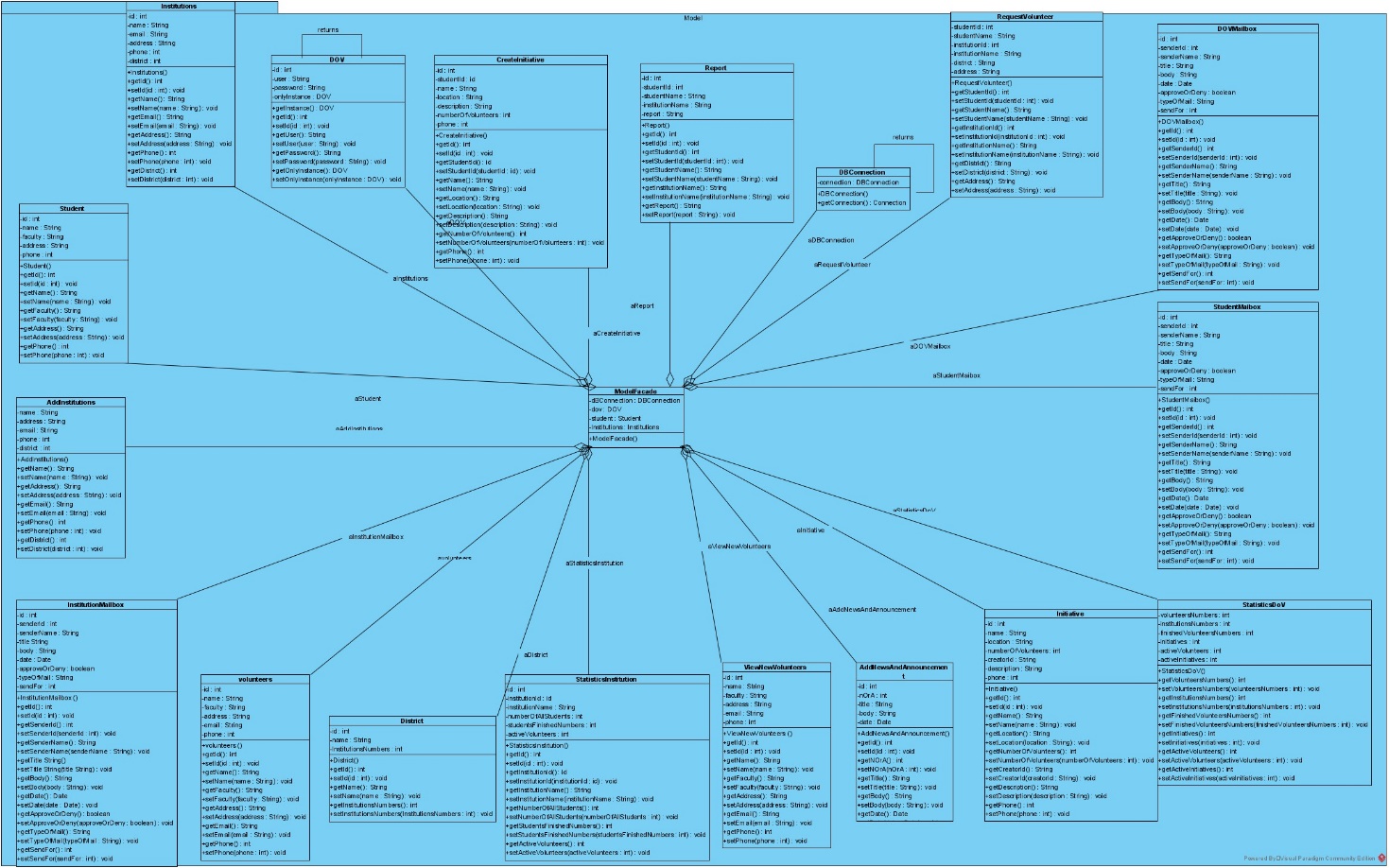
**1.5 Non-Functional Requirements**

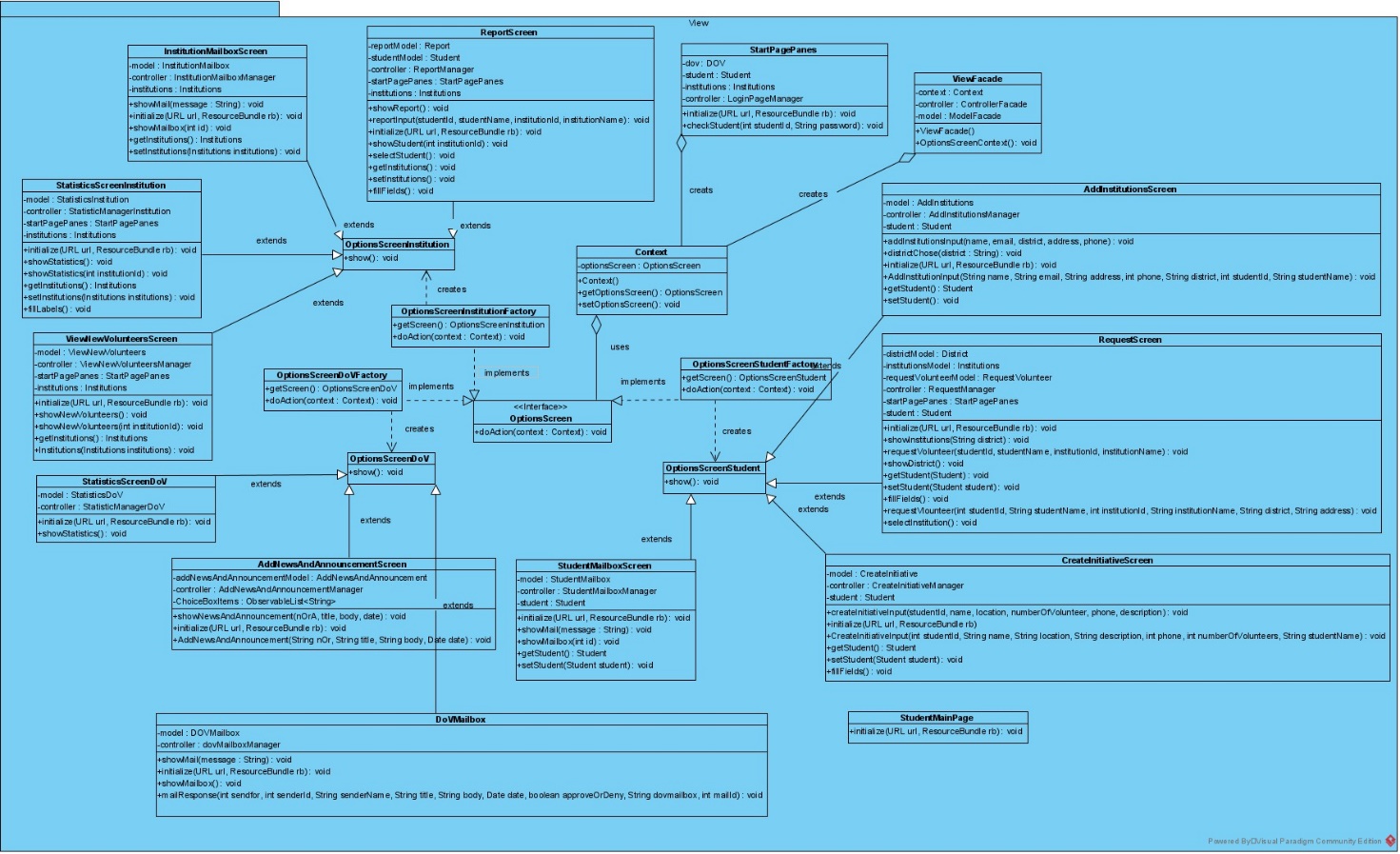
Performance, Accessibility, Usability, Efficiency, Maintainability, Portability, Security, Reliability, Resilience, and Adaptability.

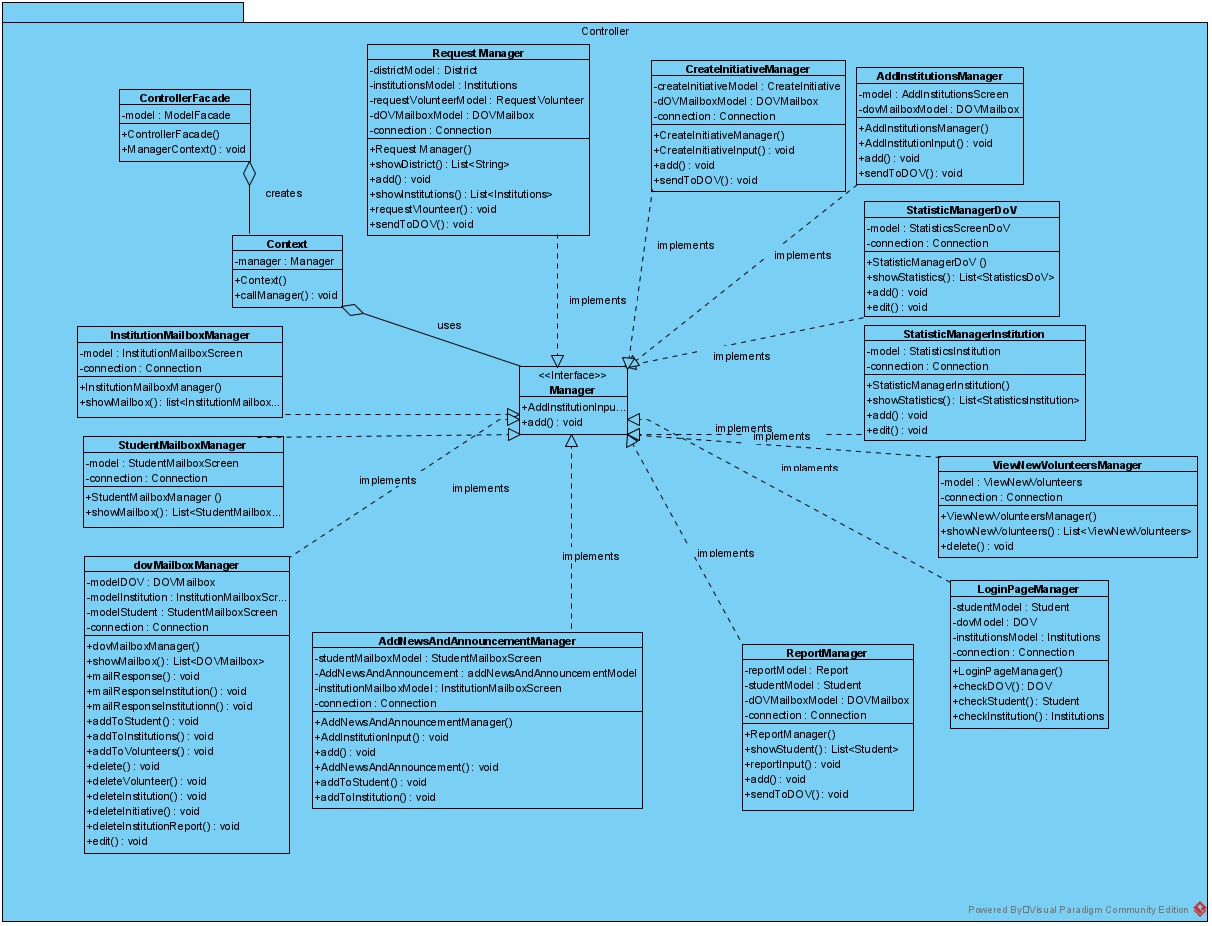
**Chapter 2. System Design and Architecture for Volunteer Work System**

**2.1 Application Architecture**

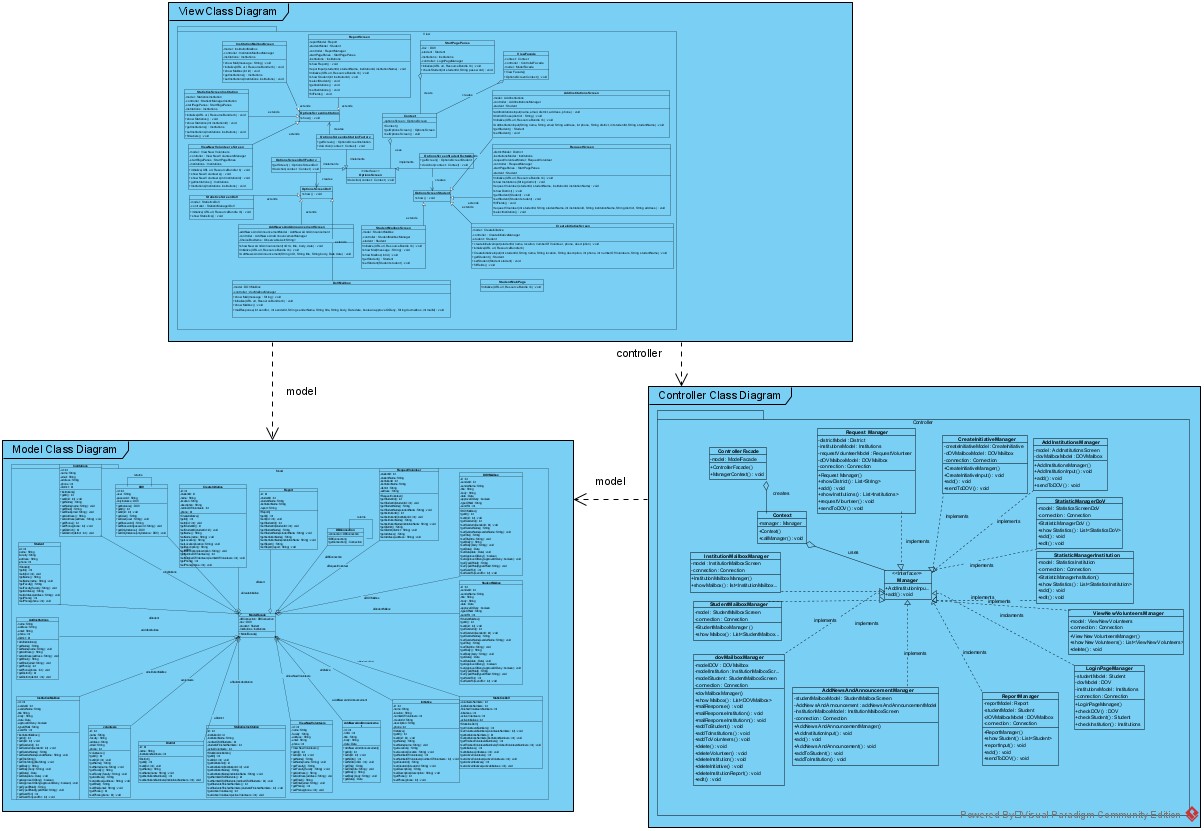


**2.1.1 Model Package**

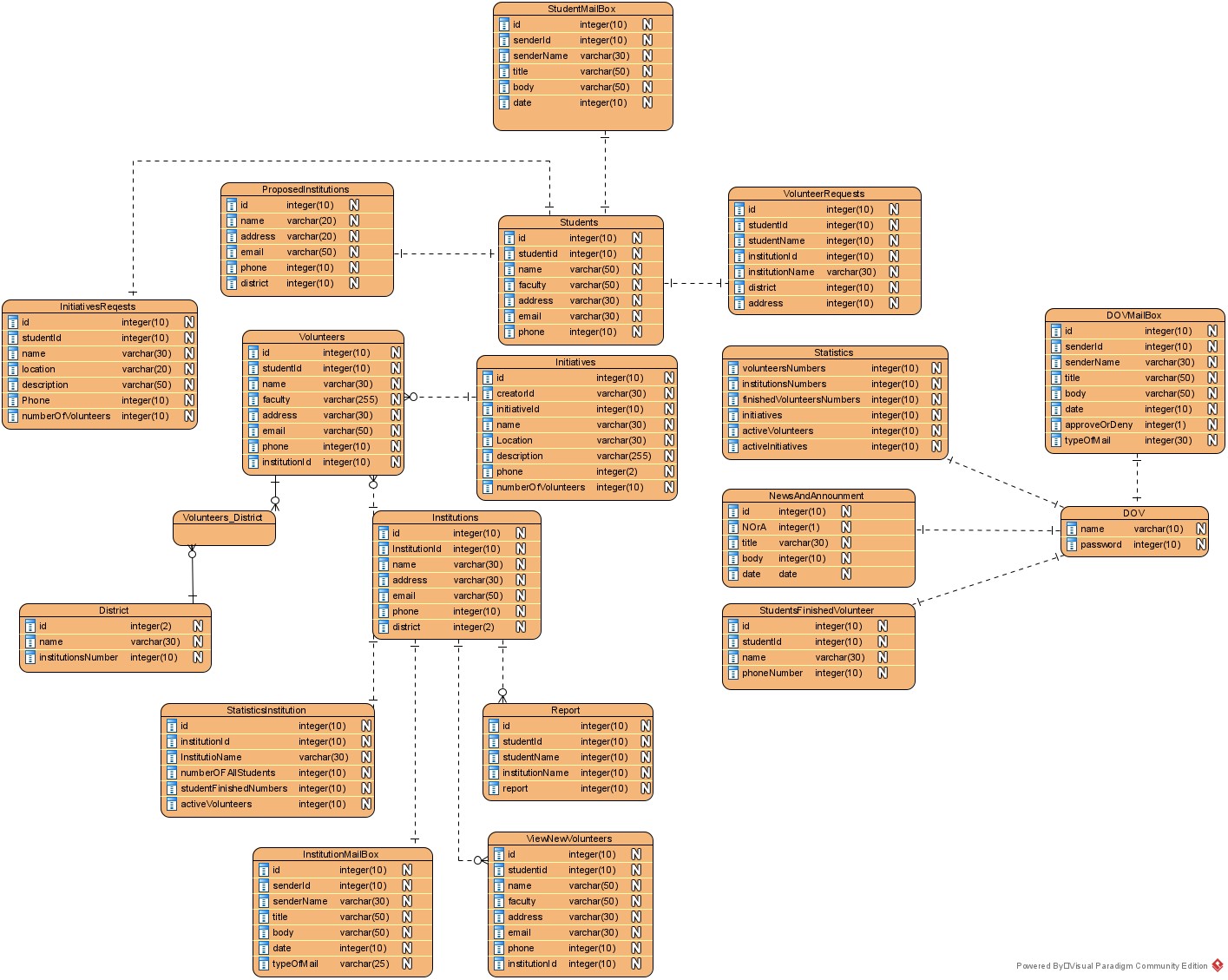
**2.1.2 View Package**

**2.1.3 Controller Package**

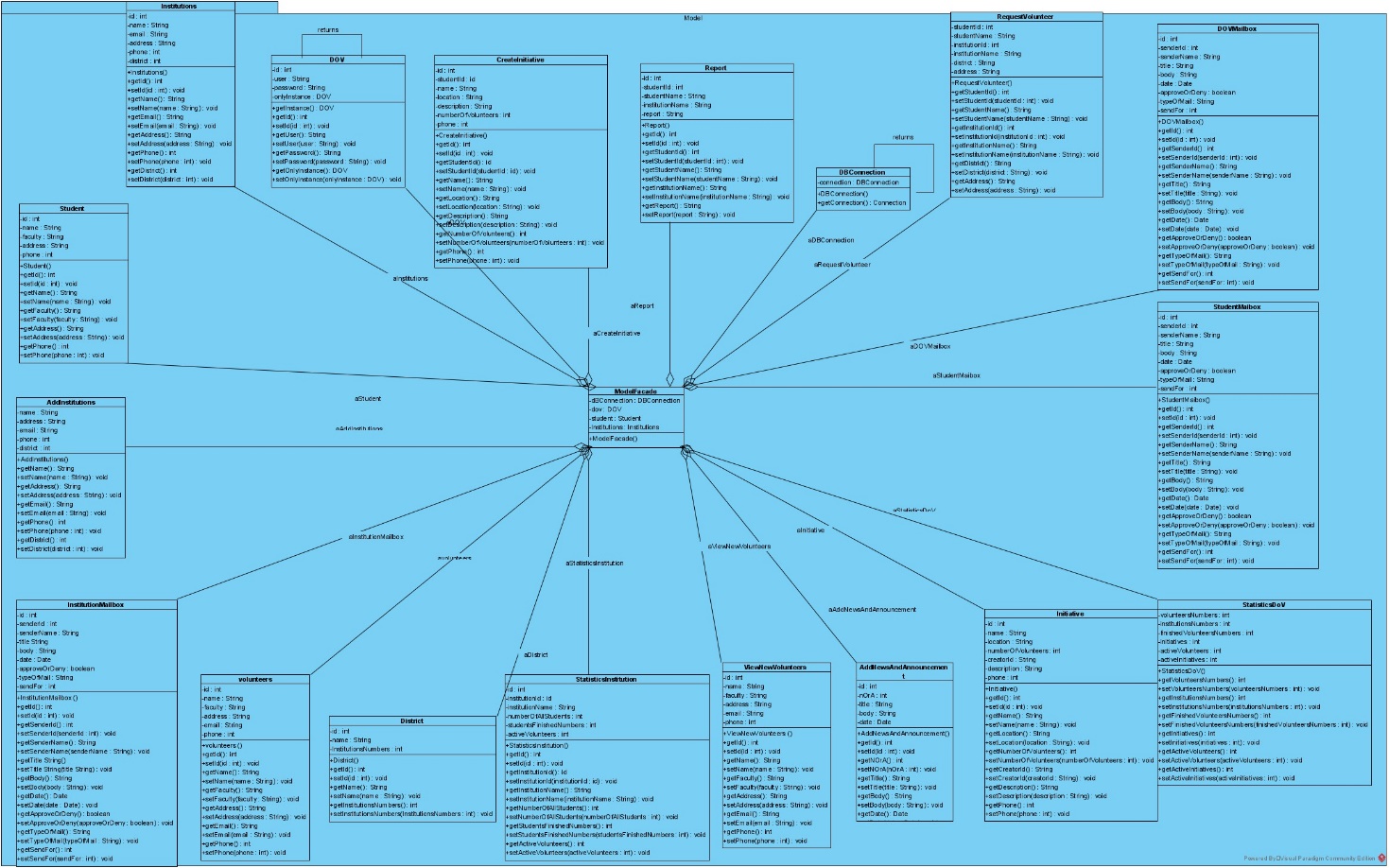
**2.2 Inter- Package Dependencies**



**2.3 Data Decomposition**



**2.4 Packages Detailed Design**

**2.4.1 Model Package**

**2.4.1.1 DOV Class**

[**Instance**: dov.

**Attributes**: id is an int, user is a string of 4 characters, password is a string of 8 characters, onlyInstance is a string of 12 characters.

**Methods**: getId, setId, getName, setName, getPassword, setPassword.

]

**2.4.1.2 Institutions Class**

**[Instance**: institution.

**Attributes**: id is a string of 2 characters, name is a string of 4 characters, address is a string of 7, district is a string of 8 characters, email is a string of 5, phone is a string of 5 characters.

**Methods**: getId, setId, getName, setName, getEmail, setEmail, getAddress, setAddress, getPhone, setPhone, getDistrict, setDistrict.

**]**

**2.4.1.3 Student Class**

**[Instance**: student.

**Attributes**: id is a string of 2 characters, name is a string of 4 characters, faculty is a string of 7, address is a string of 7 characters, email is a string of 4 char, phone is a string of 5 characters.

**Methods**: getId, setId, getName, setName, get Faculty, set Faculty, getAddress, setAddress, getEmail, setEmail, getPhone, setPhone.

**]**

**2.4.1.4 DBConnection Class**

**[Instance**: dbConnection.

**Attributes**: connection is a string of 10 characters.

**Methods**: DBConnection, getConnection, showConnection**.**

**]**

**2.4.1.5 District Class**

**[Instance**: **district**.

**Attributes**: **id**, name, InstitutionsNumbers

**Methods**: **getId, setId, getName, setName, getInstitutionsNumbers, setInstitutionsNumbers.**

**]**

**2.4.1.6 CreateInitiative Class**

**[Instance**: **createInitiative**.

**Attributes**: id, studentId, name, location, description, numbersOfNumbers, phone.

**Methods**: getId, setId, getStudentId, setStudentId, getName, setName, getLocation, setLocation, getPhone, setPhone, getDescription, setDescription, getNumberOfVolunteers, setNumberOfVolunteers.

]

**2.4.1.7 Report Class**

**[Instance**: **report**.

**Attributes**: id, studentId, studentName, institutionName, report.

**Methods**: getId, setId, getStudentId, setStudentId, getStudentName, setStudentName, getInstitutionName, setInstitutionName, getReport, setReport.

**]**

**2.4.1.8 AddInstitutions Class**

**[Instance**: **addInstitutions**.

**Attributes**: name, address, email, phone, district.

**Methods**: getId, setId, getName, setName, getAddress, setAddress, getEmail, setEmail, getPhone, setPhone, getDistrict, setDistrict.

**]**

**2.4.1.9 AddNewsAndAnnouncment Class**

**[Instance**: **addNewsAndAnnouncment**.

**Attributes**: id, nOrA, title, body, date.

**Methods**: getId, setId, getnOrA, setnOrA, getTitle, setTitle, getBody, setBody, getDate, setDate.

**2.4.1.10 RequestVolunteer Class**

**[Instance**: **requestVolunteer**.

**Attributes**: studentId, studentName, institutionId, institutionName, district, address.

**Methods**: getId, setId, getStudentId, setStudentId, getStudentName, setStudentName, getInstitutionId, setInstitutionId, getInstitutionName, setInstitutionName,getDistrict,setDistrict,getAddress, setAddress.

**]**

**2.4.1.11 Initiative Class**

**[Instance**: **initiative**.

**Attributes**: id, name, location, numberOfVolunteers, creatorId, description, phone.

**Methods**: getId, setId, getName, setName, getLocation, setLocation, getNumbersOfVolunteers, setNumbersOfVolunteers, getCreator, setCreator, getDescription, setDescription, getPhone, setPhone.

**]**

**2.4.1.12 StatisticsInstitution Class**

**[Instance**: **statisticsInstitution**.

**Attributes**: id, institutionName, institutionName, numberOfAllStudents, studentsFinishedNumbers, activeVolunteers.

**Methods**: getId, setId, getInstitutionId, setInstitutionId, getInstitutionName, setInstitutionName, getNumberOfAllStudents, setNumberOfAllStudents, geStudentsFinishedNumbers, setStudentsFinishedNumbers, getActiveVolunteers, setActiveVolunteers.

**]**

**2.4.1.13 StatisticsDOV Class**

**[Instance**: **statisticsDOV**.

**Attributes**: id ,volunteersNumbers, institutionsNumbers, finishedVolunteersNumbers, initiatives, activeVolunteers, activeintiatives.

**Methods**: getId, setId, getVolunteersNumbers, setVolunteersNumber, getInstitutionsNumbers, setInstitutionsNumbers, getFinishedVolunteersNumbers, setFinishedVolunteersNumbers, getInitiatives, setInitiatives, getActiveVolunteers, setActiveVolunteers, geActiveVolunteers, setActiveVolunteers.

**]**

**2.4.1.14 ModelFacade Class**

**[Instance**: modelFacade.

**Attributes**: dbConnection is a string of 12 characters, dov is a string of 3 characters, student is a string of 7, institution is a string of 11 characters.

**Methods**: ModelFacade**.**

**]**

**2.4.1.15 DOVMailbox Class**

**[Instance**: **dovMailbox**.

**Attributes**: id is an int of 2 characters, senderId is a int of 8 characters, senderName is a string of 11 title is a string of 5 characters, body is a string of 4 characters, date is a Date of 4 characters, approveOrDeny is a boolean of 14 characters, typeOfMail is a string of 10 characters .

**Methods**: getId, setId, getSenderId, setSenderId, getSenderName, setSenderName, getTitle, setTitle, getBody, setBody, getDate, setDate, isApproveOrDeny, setApproveOrDeny, getTypeOfMail, setTypeOfMail.

**]**

**2.4.1.16** **InstitutionMailbox Class**

**[Instance**: **institutionMailbox**.

**Attributes**: id is an int of 2 characters, senderId is a int of 8 characters, senderName is a string of 11 title is a string of 5 characters, body is a string of 4 characters, date is a Date of 4 characters, approveOrDeny is a boolean of 14 characters, typeOfMail is a string of 10 characters .

**Methods**: getId, setId, getSenderId, setSenderId, getSenderName, setSenderName, getTitle, setTitle, getBody, setBody, getDate, setDate, isApproveOrDeny, setApproveOrDeny, getTypeOfMail, setTypeOfMail.

**]**

**2.4.1.17 StudentMailbox Class**

**[Instance**: **studentMailbox**.

**Attributes**: : id is an int of 2 characters, senderId is a int of 8 characters, senderName is a string of 11 title is a string of 5 characters, body is a string of 4 characters, date is a Date of 4 characters, approveOrDeny is a boolean of 14 characters, typeOfMail is a string of 10 characters .

**Methods**: getId, setId, getSenderId, setSenderId, getSenderName, setSenderName, getTitle, setTitle, getBody, setBody, getDate, setDate, isApproveOrDeny, setApproveOrDeny, getTypeOfMail, setTypeOfMail.

**]**

**2.4.1.17 StudentMailbox Class**

**[Instance**: **studentMailbox**.

**Attributes**: : id is an int of 2 characters, senderId is a int of 8 characters, senderName is a string of 11 title is a string of 5 characters, body is a string of 4 characters, date is a Date of 4 characters, approveOrDeny is a boolean of 14 characters, typeOfMail is a string of 10 characters .

**Methods**: getId, setId, getSenderId, setSenderId, getSenderName, setSenderName, getTitle, setTitle, getBody, setBody, getDate, setDate, isApproveOrDeny, setApproveOrDeny, getTypeOfMail, setTypeOfMail.

**]**

**2.4.1.18 ViewNewVolunteers Class**

**[Instance**: **viewNewVolunteers**.

**Attributes**: id, institutionId, name, faculty, address, email, phone.

**Methods**: getId, setId, getInstitutionId, setInstitutionId, getName, setName, getFaculty, setFaculty, getAddress, setAddress, getEmail, setEmail, isPhone, setPhone.

**]**

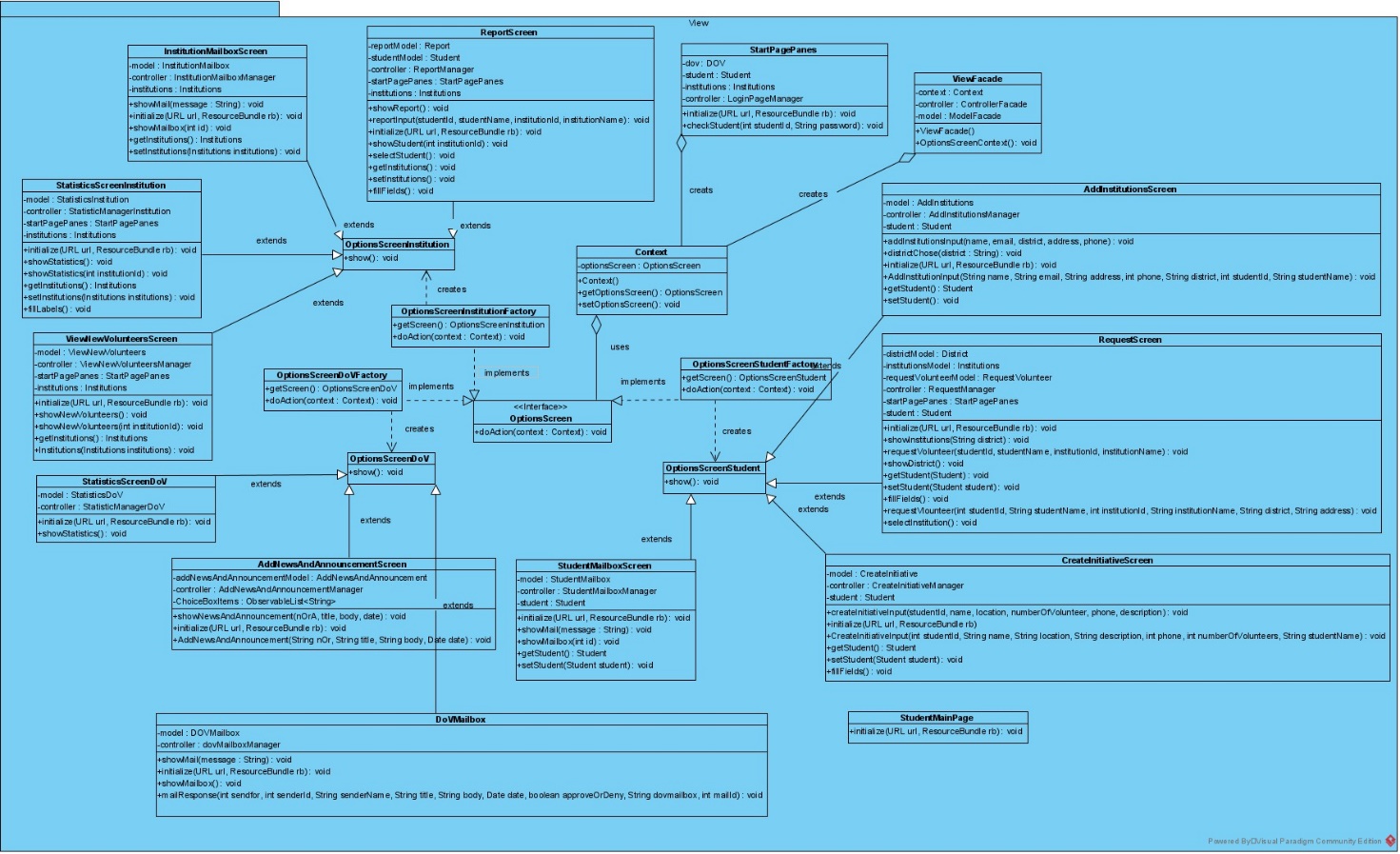
**2.4.1.19 Volunteers Class**

**[Instance**: **volunteers**.

**Attributes**: : id, institutionId, name, faculty, address, email, phone.

**Methods**: getId, setId, getName, setName, getFaculty, setFaculty, getAddress, setAddress, getEmail, setEmail, isPhone, setPhone.

**]**

**2.4.2 View Package**

**2.4.2.1 ViewFaçade Class**

[**Instance**: viewFacade.

**Attributes**: context is an object of class Context, controller is an object of class ControllerFacade, model is an object of class ModelFacade.

**Methods**: ViewFaçade, OptionsScreenContext.

]

**2.4.2.2 Context Class**

[**Instance**: context.

**Attributes**: optionsScreen is an object of class OptionsScreen.

**Methods**: Context, getOptionsScreen, setOptionsScreen.

]

**2.4.2.3 OptionsScreen Interface Class**

[**Instance**: optionsScreen.

**Methods**: doAction.

]

**2.4.2.4 OptionsScreenStudentFactory Class**

[**Instance**: optionsScreenStudentFactory.

**Methods**: getScreen, doAction.

]

**2.4.2.5 StudentMailbox Class**

[**Instance**: studentMailbox.

**Attributes**: model is an object of class StudentMailbox, controller is an object of class StudentMailboxManager, student is an object of class Student.

**Methods**: initialize, showMailbox, getStudent, setStudent.

]

**2.4.2.6 CreateInitiativeScreen Class**

[**Instance**: createInitiativeScreen.

**Attributes**: model is an object of class CreateInitiative, controller is an object of class CreateInitiativeManager, student is an object of class Student.

**Methods**: initialize, CreateInitiativeInput, getStudent, setStudent, fillFields.

]

**2.4.2.7 DistrictScreen Class**

[**Instance**: districtScreen.

**Attributes**: model is an object of class AddInstitutions, controller is an object of class AddInstitutionsManager, student is an object of class Student.

**Methods**: showDistrict, districtChose.

]

**2.4.2.8 AddInstitutionsScreen Class**

[**Instance**: addInstitutionsScreen.

**Attributes**: model is an object of class AddInstitutions, controller is an object of class AddInstitutionsManager, student is an object of class Student.

model is an object of class AddInstitutions,

**Methods**: initialize, AddInstitutionInput, getStudent, setStudent.

]

**2.4.2.9 RequestScreen Class**

[**Instance**: **requestScreen**.

**Attributes**: districtModel is an object of class District, institutionsModel is an object of class Institutions, requestVolunteerModel is an object of class RequestVolunteer, controller is an object of class RequestManager, startPagePanes is an object of class StartPagePanes, student is an object of class Student,

**Methods**: Student, showInstitutions, getStudent, setStudent, fillFields, requestVlounteer, selectInstitution.

]

**2.4.2.10 OptionsScreenDoVFactory Class**

[**Instance**: optionsScreenDoVFactory.

**Methods**: getScreen, doAction.

]

**2.4.2.11 DoVMailbox Class**

[**Instance**: doVMailbox.

**Attributes**: model is an object of class DOVMailbox, controller is an object of class dovMailboxManager.

**Methods**: initialize, showMailbox, mailResponse.

]

**2.4.2.12 AddNewsAndAnnouncementScreen Class**

[**Instance**: addNewsAndAnnouncementScreen.

**Attributes**: addNewsAndAnnouncementModelis an object of class AddNewsAndAnnouncement, controller is an object of class AddNewsAndAnnouncementManager..

**Methods**: initialize, AddNewsAndAnnouncement.

**2.4.2.13 StatisticsScreenDoV Class**

[**Instance**: statisticsScreenDoV.

**Attributes**: model is an object of class StatisticsDoV, controller is an object of class StatisticManagerDoV.

**Methods**: initialize, showStatistics.

]

**2.4.2.14 OptionsScreenInstitutionFactory Class**

[**Instance**: optionsScreenInstitutionFactory.

**Methods**: getScreen, doAction.

]

**2.4.2.15 OptionsScreenInstitution Class**

[**Instance**: optionsScreenInstitution.

**Methods**: show.

]

**2.4.2.16 InstitutionMailbox Class**

[**Instance**: institutionMailbox.

**Attributes**: model is an object of class InstitutionMailbox, controller is an object of class InstitutionMailboxManager, institutions is an object of class Institutions.

**Methods**: initialize, showMailbox, getInstitutions, setInstitutions.

]

**2.4.2.17 StatisticsScreenInstitution Class**

[**Instance**: statisticsScreenInstitution.

**Attributes**: model is an object of class StatisticsInstitution, controller is an object of class StatisticManagerInstitution, startPagePanes is an object of class StartPagePanes, institutions is an object of class Institutions.

**Methods**: initialize, showStatistics, getInstitutions, setInstitutions, fillLabels.

]

**2.4.2.18 ReportScreen Class**

[**Instance**: reportScreen.

**Attributes**: reportModel is an object of class Report, studentModel is an object of class Student, controller is an object of class ReportManager, startPagePanes is an object of class StartPagePanes, institutions is an object of class Institutions.

**Methods**: initialize, reportInput, showStudent, selectStudent, getInstitutions, setInstitutions

fillFields.

]

**2.4.2.19 ViewNewVolunteersScreen Class**

[**Instance**: viewNewVolunteersScreen.

Attributes: model is an object of class ViewNewVolunteers, controller is an object of class ViewNewVolunteersManager, startPagePanes is an object of class StartPagePanes, institutions is an object of class Institutions.

**Methods**: initialize, showNewVolunteers, getInstitutions, getInstitutions.

]

**2.4.2.20 OptionsScreenStudent Class**

[**Instance**: optionsScreenStudent.

**Methods**: show.

]

**2.4.2.21 OptionsScreenDoV Class**

[**Instance**: optionsScreenDoV.

**Methods**: show.

]

**2.4.2.22 StartPagePanes Class**

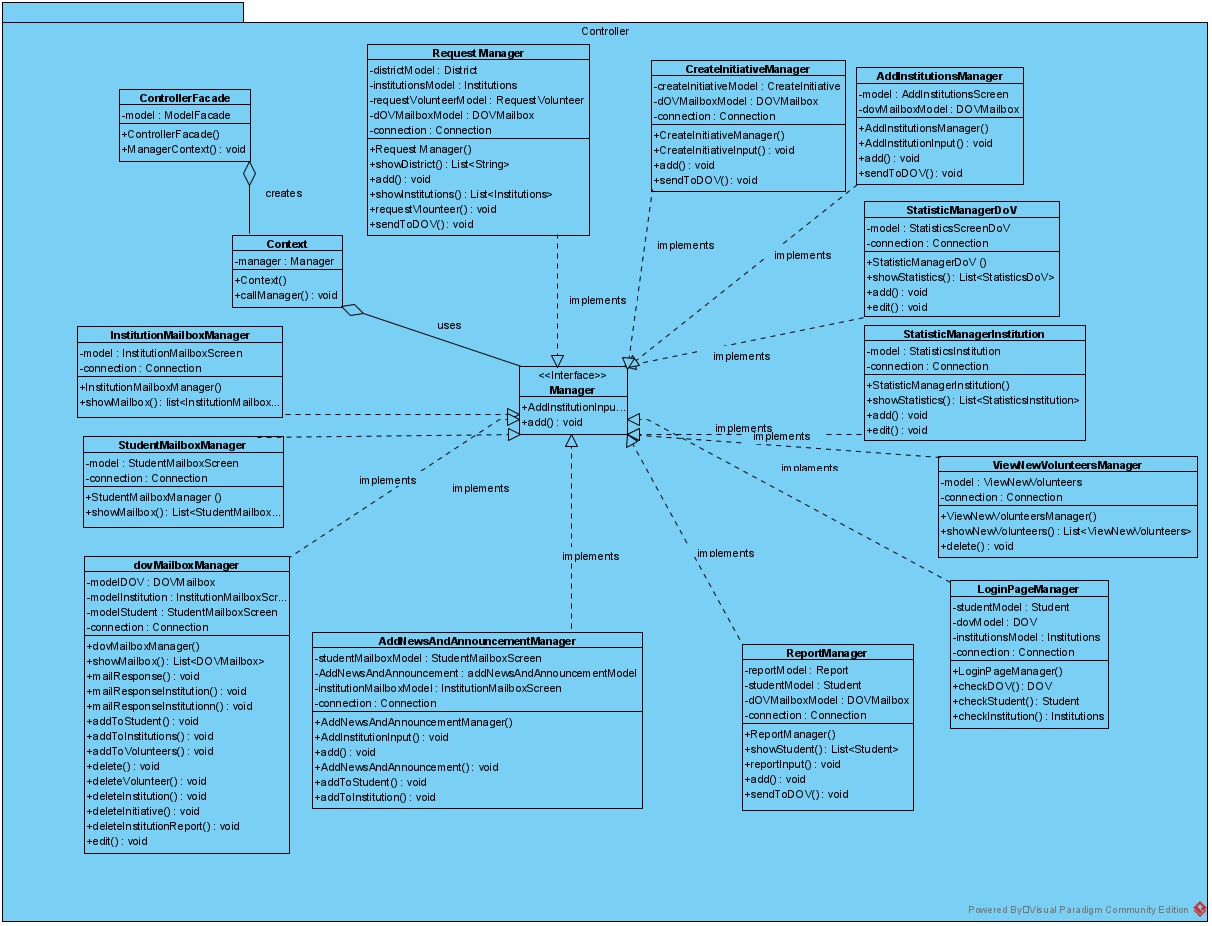
[**Instance**: StartPagePanes.

**Attributes**: dov is an Object of DOV, student is an Object of Student, institutions is an Object of Institutions, controller is an Object of LoginPageManager.

**Methods**: initialize, DOVButton, InstitutionButton, StudentButton, buttonLoginHandle,

checkStudent.

]

**2.4.3 Controller Package**

**2.4.3.1 RequestManager Class**

**[Instance**: requestManager.

**Attributes**: districtModel is an Object of District, institutionsModel is an Object of Institutions, requestVolunteerModel is an Object of RequestVolunteer, dOVMailboxModel is an Object of DOVMailbox, connection is an Object of Connection,

**Methods**: RequestManager, showDistrict, showInstitutions, requestVlounteer, add,

sendToDOV.

**]**

**2.4.3.2 CreateInitiativeManager Class**

**[Instance**: createInitiativeManager**.**

**Attributes**: createInitiativeModel is an Object of CreateInitiative, dOVMailboxModel is an Object of DOVMailbox, connection is an Object of Connection.

**Methods**: CreateInitiativeManager, CreateInitiativeInput, add, sendToDOV.

checkStudent.

**]**

**2.4.3.3 AddInstitutionsManager Class**

**[Instance**: addInstitutionsManager

**Attributes**: addInstitutionsModel is an Object of AddInstitutions, dOVMailboxModel is an Object of DOVMailbox, connection is an Object of Connection.

**Methods**: AddInstitutionsManager, AddInstitutionInput, add, sendToDOV.

**]**

**2.4.3.4 StatisticManagerInstitution Class**

**[Instance**: statisticManager

**Attributes**: model is an Object of StatisticsInstitution, connection is an Object of Connection.

**Methods**: StatisticManagerInstitution, showStatistics, add, edit.

**]**

**2.4.3.5 ViewNewVolunteersManager Class**

**[Instance**: viewNewVolunteersManager

**Attributes**: model is an Object of ViewNewVolunteers, connection is an Object of Connection.

**Methods**: ViewNewVolunteersManager, showNewVolunteers, delete.

**]**

**2.4.3.6 ReportManager Class**

**[Instance**: reportManager

**Attributes**: reportModel is an Object of Report, studentModel is an Object of Student, dOVMailboxModel is an Object of DOVMailbox, connection is an Object of Connection.

**Methods**: ReportManager, showStudent, reportInput, add, sendToDOV.

**]**

**2.4.3.7 AddNewsAndAnnouncmentManager Class**

**[Instance**: addNewsAndAnnouncmentManager

**Attributes**: addNewsAndAnnouncementModel is an Object of AddNewsAndAnnouncement, studentMailboxModel is an Object of StudentMailbox, institutionMailboxModel is an Object of InstitutionMailbox, connection is an Object of Connection.

**Methods**: AddNewsAndAnnouncementManager, AddNewsAndAnnouncement, add, addToStudent, addToInstitution.

**]**

**2.4.3.8 InstitutionMailboxManager Class**

**[Instance**: institutionMailboxManager

**Attributes**: model is an Object of InstitutionMailbox, connection is an Object of Connection.

**Methods**: InstitutionMailboxManager, showMailbox.

**]**

**2.4.3.9 StudentMailboxManager Class**

**[Instance**: studentmailboxManager

**Attributes**: model is an Object of StudentMailbox, connection is an Object of Connection.

**Methods**: StudentMailboxManager, showMailbox.

**]**

**2.4.3.10 MailboxManager Class**

**[Instance**: mailboxManager

**Attributes**: modelInstitution is an Object of InstitutionMailbox, modelStudent is an Object of StudentMailbox, connection is an Object of Connection.

**Methods**: dovMailboxManager, showMailbox, mailResponse, mailResponse, mailResponseInstitution, mailResponseStudent, addToStudent, addToInstitutions, delete, edit

**]**

**2.4.3.11 Context Class**

**[Instance**: context

**Attributes**: dov is an Object of DOV, student is an Object of Student, institutions is an Object of Institutions, controller is an Object of LoginPageManager.

**Methods**: initialize, DOVButton, InstitutionButton, StudentButton, buttonLoginHandle,

checkStudent.

**]**

**2.4.3.12 ControllerFacade Class**

**[Instance**: controllerFacade

**Attributes**: dov is an Object of DOV, student is an Object of Student, institutions is an Object of Institutions, controller is an Object of LoginPageManager.

**Methods**: initialize, DOVButton, InstitutionButton, StudentButton, buttonLoginHandle,

checkStudent.

**]**

**2.4.3.13 LoginPageManager Class**

**[Instance**: LoginPageManager

**Attributes**: studentModel is an Object of Student, dovModel is an Object of DOV, institutionsModel is an Object of Institutions, connection is an Object of Connection.

**Methods**: LoginPageManager, checkDOV, checkStudent, checkInstitution.

**]**

**2.4.3.14 StatisticManagerDoV Class**

**[Instance**: **statisticManagerDoV**

**Attributes**: model is an Object of StatisticsDoV, connection is an Object of Connection.

**Methods**: StatisticManagerDoV, showStatistics, add, edit.

**]**

**2.4.3.15 Manager interface Class**

**[Instance**: manager**]**

**Chapter 3. System Implementation and Testing for Volunteer Work System**

**3.1 Implementation and Coding Standards**









We used the Waterfall software approach of implementation to complete our project, we specified the requirements first then we designed the whole program before we started our implementation using Java programming language with JavaFX GUI library, we worked in design the GUI and programming simultaneously, for the implementation, we use MVC to separate the application’s concerns and we used other design patterns too, we declared comments in our program depended on the complexity and we checked the sections that required to add comments to make clear for the viewers, the comments can show the purpose for methods and how its work and its style.

For design patterns, we used more than one design patterns like: MVC, Model to represents an object carrying data and it also have logic to update controller if its data changes, View represents the visualization of the data that model contains, Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes and it keeps view and model separate. Singleton, involves a single class which is responsible to create an object while making sure that only single object gets created, this class provides a way to access its only object which can be accessed directly without instantiating the object of the class. Factory, provides one of the best ways to create an object. Façade, hides the complexities of the system and provides an interface to the client using which the client can access the system. Strategy, a class behavior or its algorithm can be changed at run time.

**3.2 System Testing**

I started testing when the GUI screens were ready so I can give it inputs and see what the result will be and if there are any exceptions, especially with the database. First, I tested every GUI screen alone and every database operation that is connected with the GUI, then I started to test all the screens that are related to each other, After that, I tested the whole system starting with the start page and switching accounts. Finally, I let someone that I know use the system and let him try different things and it was well accepted.

**3.2.1 Test Cases**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | Function | Test Steps | Test Data | Expected Results | Actual Results | Status |
| 1 | Request to Volunteer | We checked if we received the student application or not in our system | Student data if we received it correctly or not | To successfully send the application | Problems happened at first receiving the data but we solve it | 1 |
| 2 | Create Initiative | We checked if the student can request to add a new initiative | New Initiative data to received correctly | To successfully request to add a new initiative | No problem, student can request any initiative | 1 |
| 3 | Add Institutions | We checked if the student can request to add a new Institution | New Institution data to received correctly | To successfully request to add a new Institution | Problems happened at first receiving the data but we solve it | 1 |
| 4 | Add News and Announcements | We checked if the DOV can add a new News or Announcement | News data to received correctly | To successfully show the News | No problem, DOV can add news | 1 |
| 5 | Approve or Deny Requests and Reports | We checked if the DOV can approve or deny volunteer requests | Massage reply sent to student by database | Student should get a feedback for her request | No problem | 1 |
| 6 | See Statistics | DOV and Institution can check statistics | Information about volunteers | To show every information successfully | We got a database retrieve problem first but we solve it | 1 |
| 7 | View New Volunteers | Institution should have ability to view every new volunteer | New volunteers' information | Institution can see all new volunteers | No problems | 1 |
| 8 | Send Reports | Institution should have ability to send Reports to DOV about every finished volunteer | Report's information for every volunteer | DOV receive reports successfully | No problems | 1 |

**4. References**

[1] Ian Somerville, “Software Engineering”, 10th edition, Pearson, 2016

[2] Dr. Abdelkareem Alashqar, “Software Engineering II” course videos and lectures

[3] tutorialspoint, “Java design patterns”, www.tutorialspoint.com

[4] JavaTpoint, https://www.javatpoint.com/design-patterns-in-java.

[5] Visual Paradigm, https://www.visual-paradigm.com

[6] GitHub, https://github.com/MahmoudAbusaqer/SE2-Project-VolunteerWorkSystem.

[7] JavaFX Guide, https://www.oracle.com/java/technologies/javase/javafx-overview.html.