Party planner

Group 3

Michal Martinček 279954

Filip Krupa 281653

Roxana Maria Spiridon 280055

Luís Miguel Ferreira Freire 280038

Supervisors:

Jakob Knop Rasmussen

Joseph Chukwudi Okika

VIA University College

30012 **characters**

VIA Software engineering

Third semester

20.12.2019

Table of content

Abstract

The purpose of the following document is to record and detail the product which resulted from the development process of GROUP 3 for the third semester project. The paper contains valuable information about the technical aspects of the product, along with details regarding the technologies used and implemented.

The system is aimed to be heterogeneous, following that, different languages such as C# and Java Enterprise were used in development. In addition markup languages such as HTML, Javascript and the PostgreSQL sql dialect were used for the different components of the system. The product, in essence a party planner platform, displays a 3 tier architecture and makes use of different technologies used in computer communication such as web services and socket TCP connection protocols.

The two main topics discussed in the document bring in discussion the business value of the project (Analysis chapter), as well as the quality of the physical product, naming its technical side along with its actual implementation. The data analysis was based upon the user stories identified.

The main technical choices, other then the once imposed by the school body, name the use of the Razor framework for the first tier development and the choice of a JBOSS web service provider for exporting the second tier artifacts. Additionally the project will attempt to answer to the problem of having multiple users through a synchronized management of the Treads.

The results of the paper summarizes a feasible solution to the project requirements as well as a good product that facilitates an answer to the need of a party planner platform.

1. Introduction

The purpose of this project is to provide a heterogeneous software application, using tools which the group learned throughout the semester, meant to deliver a solution to the problem of organizing a party or a festive event. The product, named PartyPlanner addresses the question of how difficult it is to organize, administer and advertise an event of such a nature, taking in consideration the amount of popularity the event organisation industry has received in recent decades.

Focusing on the target audience, as it is mentioned in the background description of the project, the source of inspiration was studying the behaviour of college students. One of the most important aspects of an event organisation is the time constraint. That does not translate well with the impatience most students have. A good party does suppose a fairly organisation, but it is so often that teenagers and students wish for something spontaneous, on the go, fast, which most of the time ends in disaster. Considering this fact, the group has seen a potential gap in the market and opportunity to create their own solution.

Currently students are using social networks as a tool for social events and gatherings. The product is meant to follow the trend and aims to provide a system reachable from a network point. Instead of already established social networks, the prevention of data tracking and informal atmosphere is provided.

As a delimitations the project will not include any catering or housing services as well as to not collect any fees for registration.

The goal is mainly to develop a web page which will be loved from the view of users and secured from the point of view of admins and possible attackers. This means it has an easy to use interface and proper way of using web services.

The security measurements will only revolve around the HTTPS concept and other threats will be described and discussed in analysis and the design part of this document.

2. Analysis

The following segment describes the system's requirements for it to fit the idea and theme of a party planner platform. The approach towards the research is under the scope of the business idea and its results are meant toward the possible stakeholders of the product.

Having the aforementioned, the analysis of the project description shows that the core of the system revolves around the Party object and the ways should it be accessible, editable and shared across the platform. It is implied that for the modification of the Party object different rights are supposed to be given or existing to the current user of the application, resulting in the division of multiple actors. The identified actors are represented by the usual user of the system that can participate in different parties and the host, which has full authority over the object information state.

The core of the system is the Party object and two types of users - normal user, participant of the party and the host, creator/manager of the party. Connected to this, the party has own specific properties.

2.1. Requirements

2.1.1. Functional Requirements

The following requirements describe the system available features. The requirements are presented using a user story format. With that being said, PartyPlanner has the following features:

1. As a user, I want to be able to create a party
2. As a host, I want to be able to invite/add users to the party
3. As a user, I want to be able to login/register
4. As a host, I want to be able to create a shopping list for the party
5. As a user, I want to see the parties I participate in
6. As a host, I want to be able to make the party private
7. As a host, I want to be able to edit a party
8. As a user, I want to be able to accept/decline an invite to a party
9. As a host, I want to be able to add a Spotify playlist to the party
10. As a user, I want to be able to add songs to the party's Spotify playlist

2.1.2. Non-Functional Requirements

1. The system should be implemented using Java, C# and PostgreSQL.
2. The system should run on every major web browser independently of OS.

2.2. Use Case Model

The system’s functional requirements are represented through the following use case diagram. In that way we identify 8 distinct use cases.

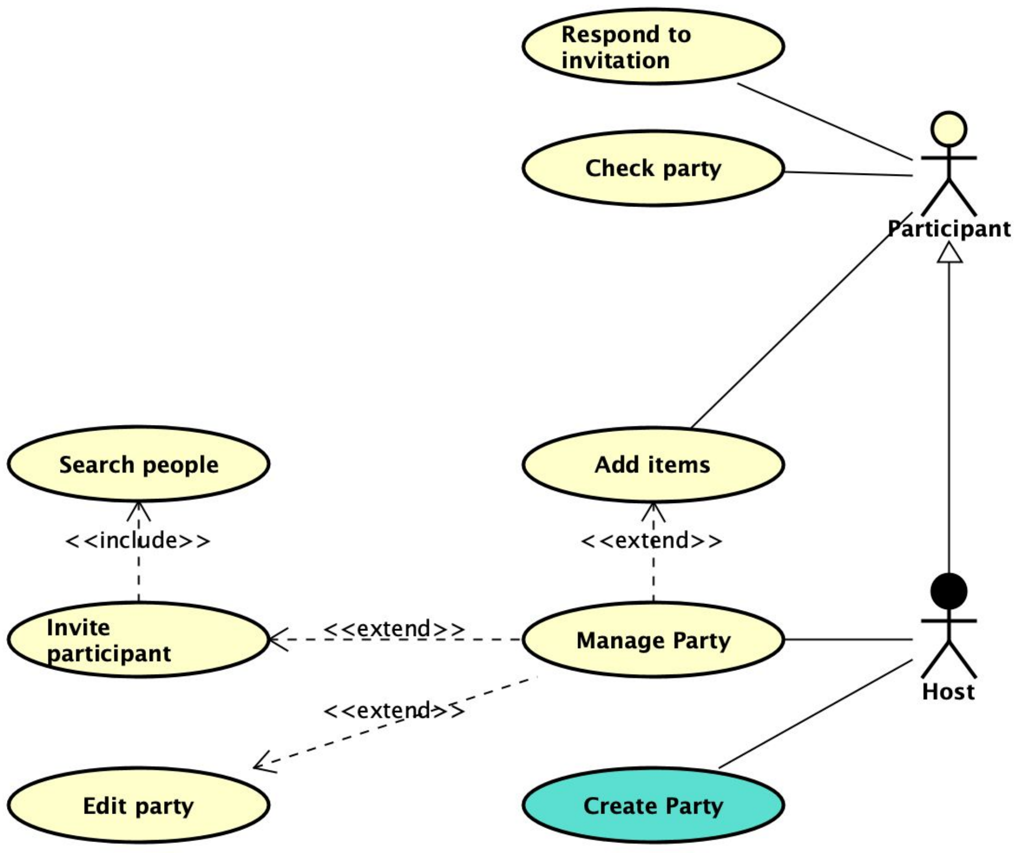
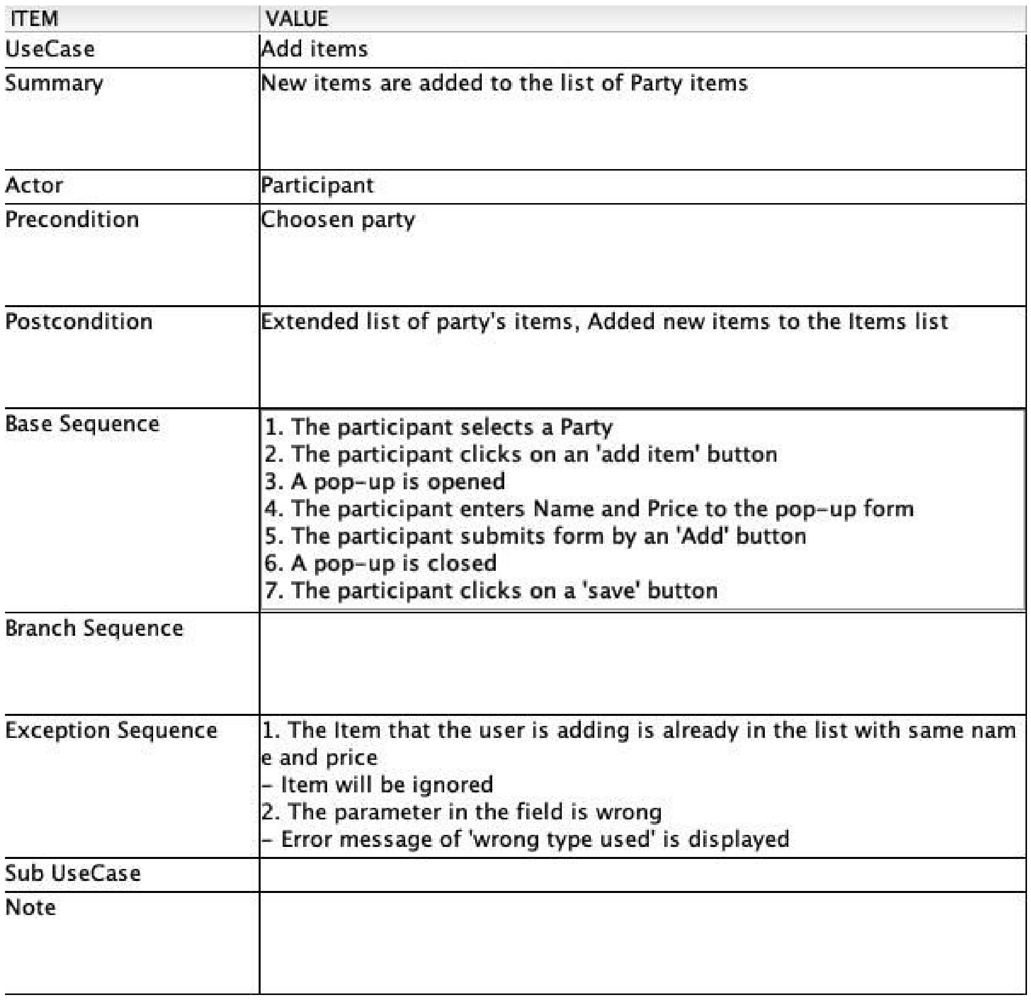


Figure 1: Use Case Model Diagram

The diagram describes the features of the PartyPlanner as well as its user interaction. As a result the Participant’s interaction with the system is limited to only Checking a party details, to add items to a Party in which he/she participates in and respond to an invitation in the case another user requests his/her participation in the Party in question. On the other hand, an actor having the Host role has given more possibilities, those having the ability to search for participants, add them as well as editing the Party details.

2.3. Use case Description and Activity Diagram

The description table below represents the procedure a user performs in interaction with the system if the client's intention is to add an item to a Party in which he/she participates. This UseCase has been chosen as the one to follow throughout this report, because of its importance in the system's core functionality and being a key feature.



ITEM	VALUE
UseCase	Add items
Summary	New items are added to the list of Party items
Actor	Participant
Precondition	Choosen party
Postcondition	Extended list of party's items, Added new items to the Items list
Base Sequence	1. The participant selects a Party 2. The participant clicks on an 'add item' button 3. A pop-up is opened 4. The participant enters Name and Price to the pop-up form 5. The participant submits form by an 'Add' button 6. A pop-up is closed 7. The participant clicks on a 'save' button
Branch Sequence	
Exception Sequence	1. The Item that the user is adding is already in the list with same nam e and price - Item will be ignored 2. The parameter in the field is wrong - Error message of 'wrong type used' is displayed
Sub UseCase	
Note	

Figure 2: Use Case Description

The Use case description describes the steps the user has to complete to be able to execute the intended action. In addition, from a UI point of view, the activity diagram shows details about the user interface, given the final product. For an even greater understanding of the use case, please refer to the Activity diagram below:

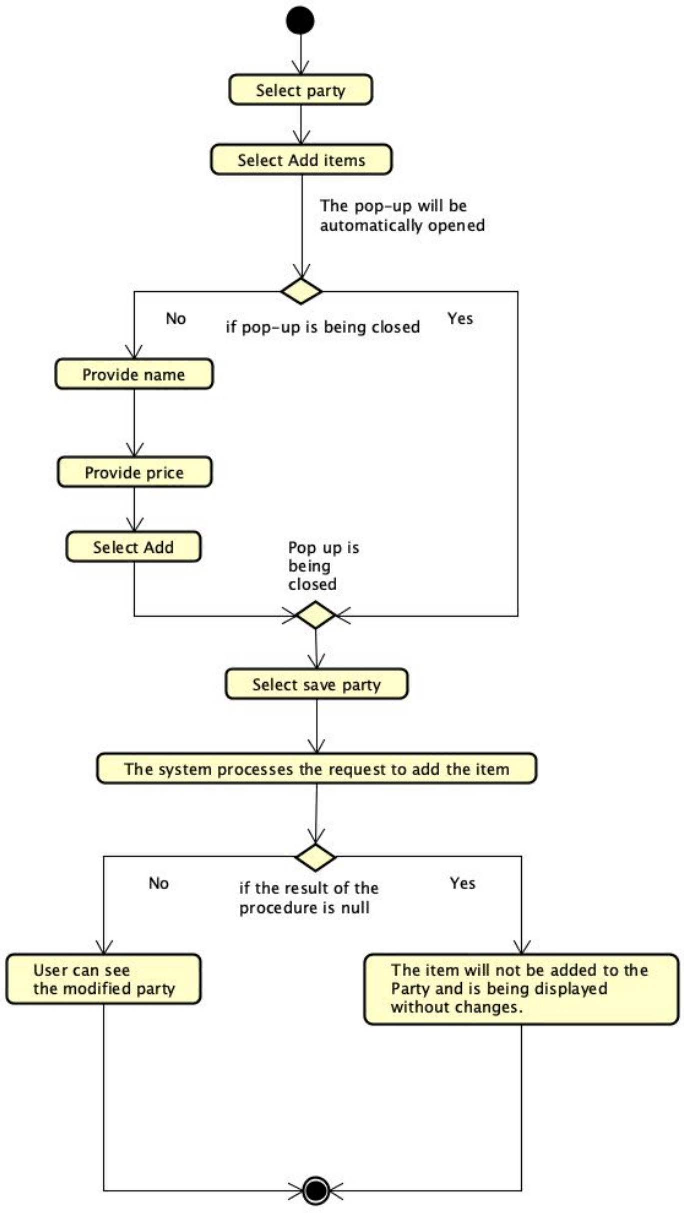

Select party
Select Add items
The pop-up will be automatically opened
No
if pop-up is being closed
Yes
Provide name
Provide price
Select Add
Pop up is being closed
Select save party
The system processes the request to add the item
No
if the result of the procedure is null
Yes
User can see the modified party
The item will not be added to the Party and is being displayed without changes.

Figure 3: Activity diagram

2.4. Domain Model

The result of Analysis can be seen in the following diagram that represents the basic domain needed to fulfill the aforementioned requirements. In essence the domain model is meant to streamline the data transfers and prevent the need for adapters in each component of the system.

Naming the entities, we discover that for the development it is required:

- a Person Object that will define and incapacitate the characteristics of the system’s user. Those include a name, an email address, a username and password
- an Item Object that will define and incapacitate the characteristics of an item meant for a specific Party. Those include a name and the price of the item.
- a Party object specifically meant to represent how a real-life party will be characterized in the system. The Party will include a list of Items and Persons as well as general details of the event. Those include a partyTitle, a description of the event, a location, a date, a time, a property meant to determine if the event is open to the large public, and a particular Person instance representing the host of the event
- an Invitation object; This object is used when a user of the system requires the participation of another user to a party that he/she is hosting. The characteristics of an Invitation include the Party object to which the participation is required, a status and an identifier of the user/ Person to which the invitation is made.

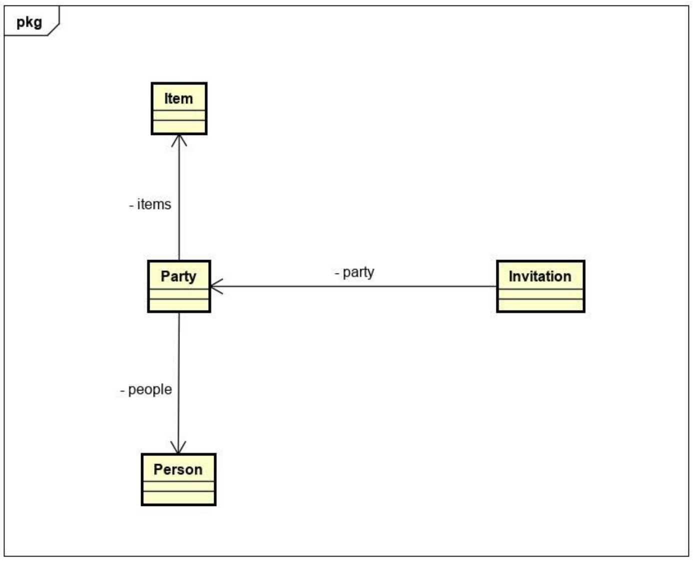


Figure 4: Domain Model Diagram

3. Design

Taking in consideration the requirements set by the school body, the architecture it identifies itself as a 3 tier architecture with a heterogenous characteristics. These requirements very clearly specify the inclusion of a three tier architecture. Considering this, design decisions have been made to allow each tier to operate in a way that could be considered as mimicking a MVC based system.

The only difference would be that every tier should be mostly language agnostic and therefore could theoretically be replaced with some other 'component' which could even be written in a different programming language therefore successfully fulfilling this requirement.

More specific examples and descriptions will be provided in the next sections. Besides this, it is important to highlight the whole structure of sending and sharing data between tiers. It has been decided to use a package design pattern between C# Razor pages and the business logic in Java and a box design pattern between Java and PostgreSQL access tier, both which shall be described in the 'Technologies' chapter.

3.1. Architecture

The system is composed of three tiers, specifically C# on the first, Java Enterprise on the second and Java Enterprise plus PostgreSQL based database on the third one. Institutional requirements are represented throughout the system, such as consuming and exposing of web services by specialized Java API, Razor web based GUI providing easy access for multiple users and socket communication between tiers.

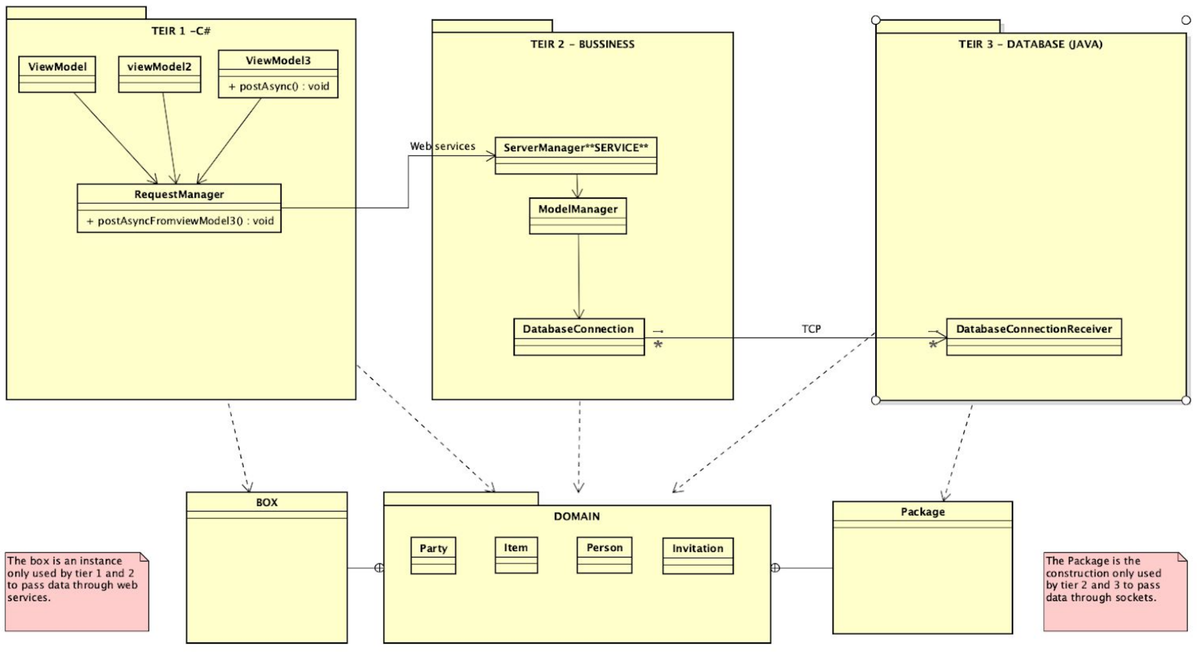

TEIR 1 -C#
ViewModel
viewModel2
ViewModel3
+ postAsync() : void
RequestManager
+ postAsyncFromviewModel3() : void
Web services
TEIR 2 - BUSSINESS
ServerManager**SERVICE**
ModelManager
DatabaseConnection
TCP
TEIR 3 - DATABASE (JAVA)
DatabaseConnectionReceiver
BOX
DOMAIN
Party
Item
Person
Invitation
Package
The box is an instance only used by tier 1 and 2 to pass data through web services.
The Package is the construction only used by tier 2 and 3 to pass data through sockets.

Figure 5: Architecture Diagram

The following section develops more upon the inner workings between the tiers and the technologies used to insure the great communication between the components

3.2. Technologies

3.2.1. Web services

The system is using a REST API web services hosted on the second tier. Based on this, the whole business logic is running on a JBoss service provider. In that sense it is the JBoss service that intercepts a client request through the set url that corresponds to the actual component of the system. The class PartyPlannerService defines the url links meant for the system and mapps the link to the method that it needs to be executed in the system. Previous research of similar competitive technologies proved this choice to be manly stabile and the documentation from external resources appeared as a big advantage.

3.2.2. Sockets Connection (TCP protocol)

The product makes use of a Socket computer connection using a TCP protocol between the second and third tier of the system. The connection is meant to transmit from the second tier commands that are relevant and have a direct effect on the persistence (database) component of the system.

Following the establishment of the connection between tiers in the event when a procedure is triggered by the user, the system creates a new Threat out of the new connection, giving the possibility of multiple clients to proceed in performing their desired actions onto the system in parallel from one another.

The management of the Thread is done in a synchronized manner meaning that the tier 2 of the system will wait for the results to be provided by the Thread created on the third tier.

Given the notion of Thread, the third three will include a run method for the object that is runned upon creating the Socket connection. Due to the excessive number of commands and procedures possible in the system there is a need of precisely identifying the method that it needs to be called along with the write data retrieved from the socket InputStream. The solution to this question will be detailed further under the section 3.33 Design Patterns.

3.2.3. JDBC

The database, as the main storage of the user data, is implemented on a separate third tier. In addition the persistence component is written in PostgreSQL.

In correlation to the analysis part, the database domain has a similar Domain Model, identifying the Party entity, Item, Person and Invitation entity. Given the delimitation of the database and the constraints that in a table a tuble is not allowed to have multiple values, it was identified the need of two relationships that will store and log the data meant to determine which people are participating in which party, respectfully which items are needed for which party.

Due to the introduction of the primary keys, design as serial numbers (it increases which each insertion in the table), the generic domain will also inherit those properties. In this case the system provided a way of uniquely identifying all its data by their specific id. In the other tiers the objects inherit their ids as int type properties.

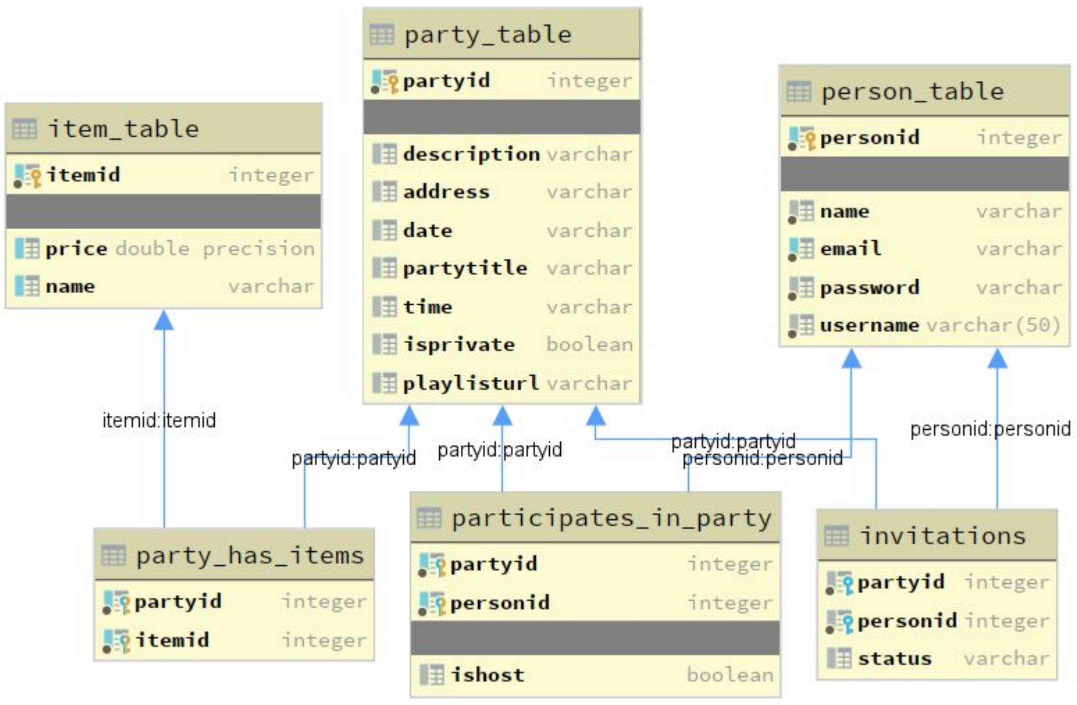


Figure 6: Database domain diagram

3.2.4. UI / UX Design Choices

While all the logic is handled by the second tier, the Java API, the first tier is using C# combined with traditional web technologies so it is easier to distribute the final software to a wider range of users, as well as introducing the client to the system in an easy to follow, user-friendly manner.

Discussion preceded the final design in which was decided to combine C# Razor pages, Javascript and Css. Every one of these tools has a specific value. Beside Javascript, which is also implemented on the webpage as a tool for adding animations to the webpage, C# .NET and its libraries have been used for the management of the inner functionality. All the pages are following standard design rules for integrity and easier navigation by implementing the norms of authentication and authorization.

3.3. Design Patterns

3.3.1. Singleton

The singleton design pattern it is used at the first tier level of the system and acts as a cache storing the necessary information needed on the different pages of the web page component. The information included is the following:

- the Person object containing the user credentials
- the list of Party objects representing the list of parties in which the user is registered as a participant or either a host
- a Party object named as “activeParty” used for the purpose of populating the page with the information of a desired party. The party object is retrieved from the list of parties aforementioned
- a list of Invitation objects that represent the list of Notifications received by the user

The instances represent static data of the user that don't change unless a major change has been executed, that will redirect to this information to be updated by retrieving it from the server. In addition they are all retrieved from the system through the means described under the Technologies section. Still there are changes that can be made frecvently to the activeParty properties in the case of editing, adding or removing from its characteristics. The following instances are meant to temporarily store this changes before they are sent to the other part of the system for execution:

- a list of Items representing the items that are to be added to the Party
- a list of Items representing the items that are to be removed from the Party
- a list of Person objects representing the people that are to be added to the Party
- a list of Person objects representing the people that are to be removed from the Party

They are to be cleared upon making the request to the system to execute the changes through the updateParty procedure.

3.3.2. Box/Package

The purpose of the box and package classes is to provide a uniform way to package, send and receive all the necessary objects for successful execution of each task as it handles both sending of the objects and resolving of how to route the request and which method to call to achieve a successful query. As seen on the diagram below, the 'box' handles the traffic that goes through the web-service between tier one and two. The 'package' handles directing all the queries aimed at the database through the socket connection connecting the business and database tiers.

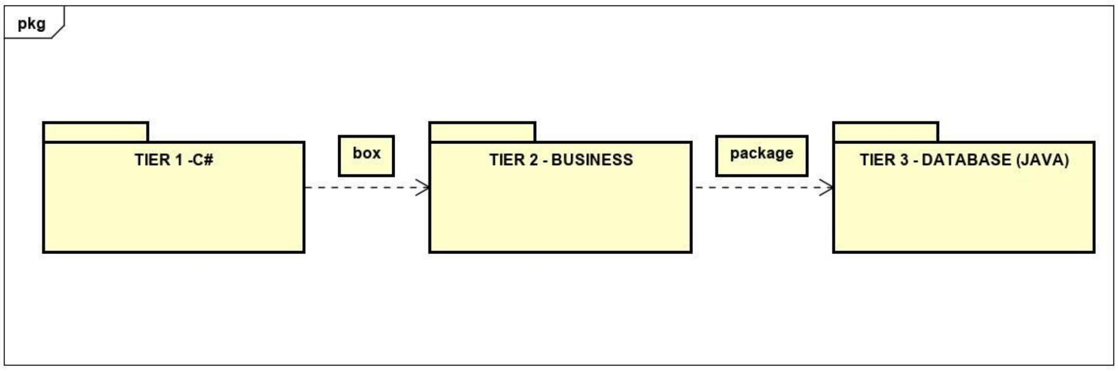


Figure 7: Box/Package diagram

To exemplify the use of the technologies and design patterns mentioned the following section provides an illustrative representation of the “add an item ” Use case through a Sequence diagram. For a better visualisation the sequence is divided into two parts as follows:

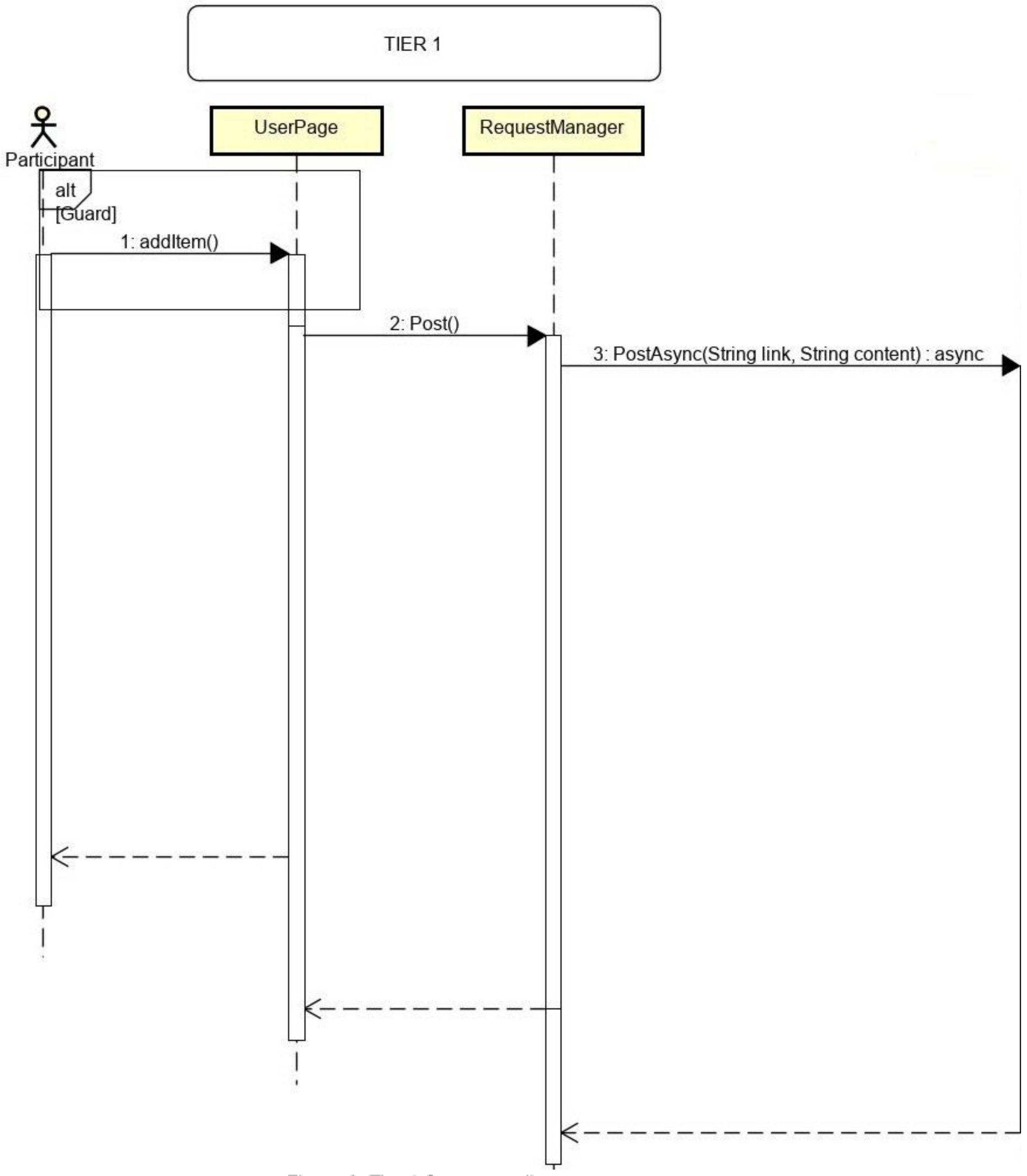


Figure 8: Tier 1 Sequence diagram

The given diagram explains the procedure on the first tier. The system records each time the participant will add an item. When ready to save the changes, the system will "request" the command to execute, resulting in a POST over to the business side of the system. The post is required the url link mapping the method that will be called using the web service and passing as argument a StringContent, containing a JSON representation of a Box object containing the Party where the items provided by the user should be added. Following is the procedure course on tier 2 and 3 as described in the following sequence diagram:

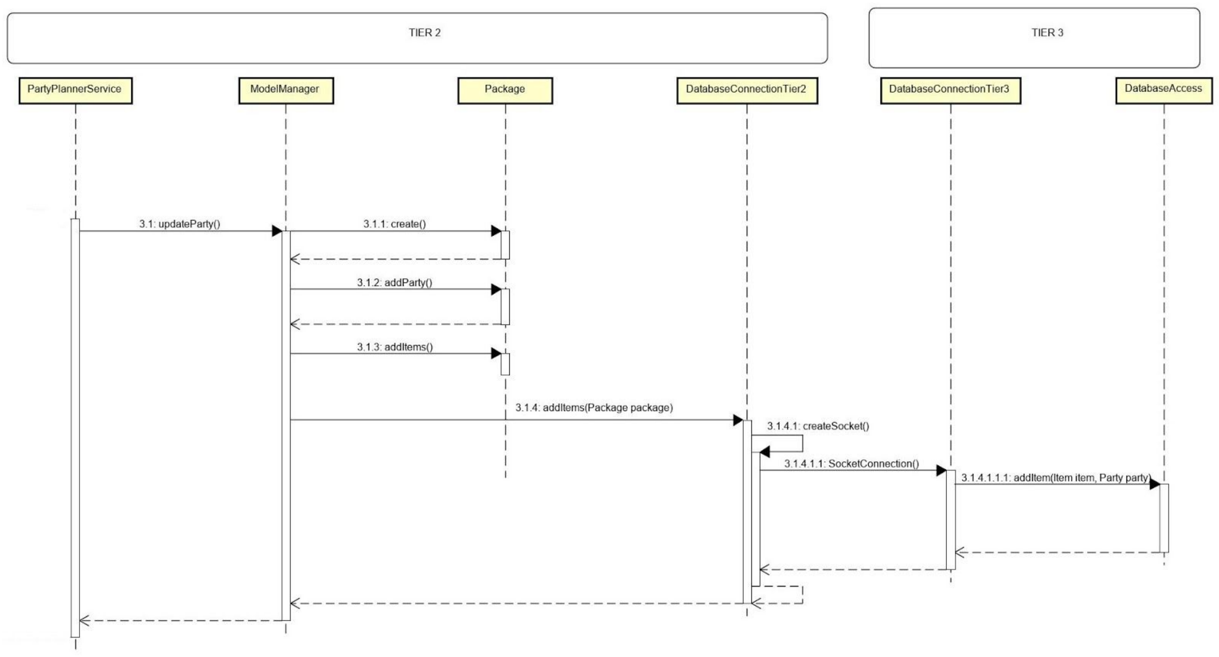


Figure 9: Tier2/3 Sequence diagram

Upon reaching the second tier the Box parameter is Deserialized and the object that it would contain processed and put inside of a Package object command. The package is given an appropriate command and sent out through a socket connection to the third tier. When received the Package is read and rendered, following that the information it contains, the list of items to be added to the Party in the database using a JDBC connection.

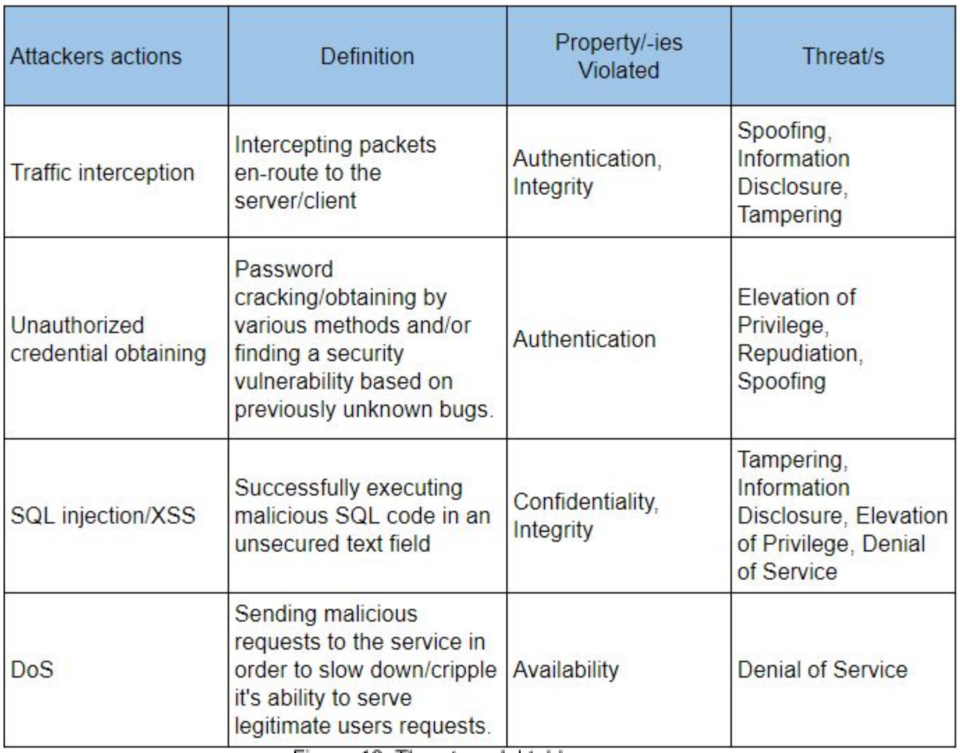
4. Security

In order to be able to asses the possible security vulnerabilities of the system a threat model and a risk assessment model were created to help the future endeavours of securing/patching any and all security holes that might arise from designing and implementing a system using technologies that are specified in more detail in the Technologies chapter of this report.

4.1. Analysis

4.1.1. Threat Model

The table below is a threat model based on the STRIDE model. It describes some of the most likely actions which the attacker could perform in order to exploit the system, their definition and properties, which each action would violate, and the threat from the STRIDE model which such action would pose.



Attackers actions	Definition	Property/-ies Violated	Threat/s
Traffic interception	Intercepting packets en-route to the server/client	Authentication, Integrity	Spoofing, Information Disclosure, Tampering
Unauthorized credential obtaining	Password cracking/obtaining by various methods and/or finding a security vulnerability based on previously unknown bugs.	Authentication	Elevation of Privilege, Repudiation, Spoofing
SQL injection/XSS	Successfully executing malicious SQL code in an unsecured text field	Confidentiality, Integrity	Tampering, Information Disclosure, Elevation of Privilege, Denial of Service
DoS	Sending malicious requests to the service in order to slow down/cripple it's ability to serve legitimate users requests.	Availability	Denial of Service

Figure 10: Threat model table

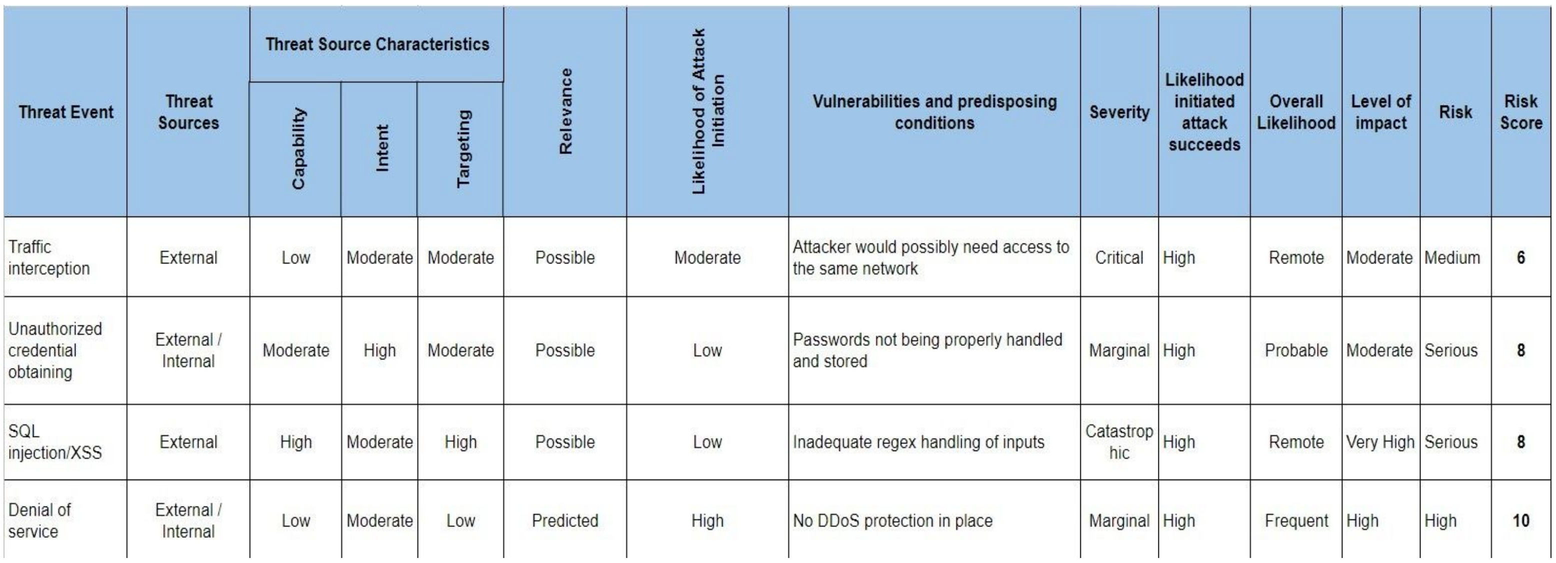
4.1.2. Objectives

Based on the threat model there are four types of actions which the attacker could exploit which are based on the STRIDE methodology. Most of the threats defined by STRIDE can be applied to these actions, therefore a creation/implementation of policies/measures would be highly suggested in order to achieve the objective of successfully achieve the desired goal of developing a secure application/system.

The objective of a potential attacker would be to circumvent any and all defensive/security measures that might be put in place to prevent such threats from happening in the first place.

4.1.3. Risk assessment model

Risk assessment model has been created as a part of the analysis for this project. This model reveals some major security vulnerabilities that would have to be (patched/handled) in order for this project to be considered as a possible real-world capable product. Focus was put on the security from both the user and "system" perspectives.

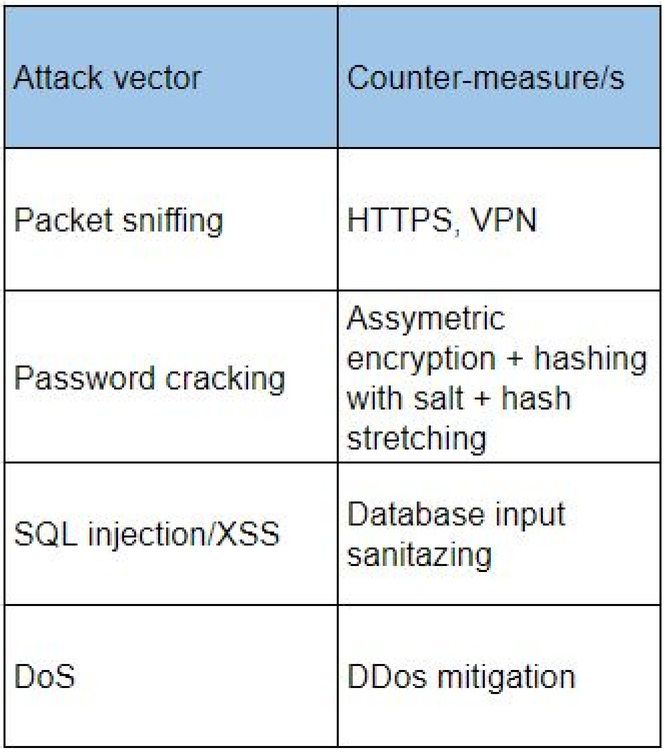


Threat Event	Threat Sources	Threat Source Characteristics			Relevance	Likelihood of Attack Initiation	Vulnerabilities and predisposing conditions	Severity	Likelihood initiated attack succeeds	Overall Likelihood	Level of impact	Risk	Risk Score
		Capability	Intent	Targeting									
Traffic interception	External	Low	Moderate	Moderate	Possible	Moderate	Attacker would possibly need access to the same network	Critical	High	Remote	Moderate	Medium	**6**
Unauthorized credential obtaining	External / Internal	Moderate	High	Moderate	Possible	Low	Passwords not being properly handled and stored	Marginal	High	Probable	Moderate	Serious	**8**
SQL injection/XSS	External	High	Moderate	High	Possible	Low	Inadequate regex handling of inputs	Catastrop hic	High	Remote	Very High	Serious	**8**
Denial of service	External / Internal	Low	Moderate	Low	Predicted	High	No DDoS protection in place	Marginal	High	Frequent	High	High	**10**

Figure 11: Risk assessment model table

4.2. Design

According to the conducted analysis of the possible threats that might arise there are four main attack vectors for an attacker to exploit. All of them are specified in the table below as well as the mechanisms that might be effective in the attempt to counteract the malicious effects of such vectors being exploited.
However for this project only HTTPS has been implemented as per the institutional requirements.



Attack vector	Counter-measure/s
Packet sniffing	HTTPS, VPN
Password cracking	Assymetric encryption + hashing with salt + hash stretching
SQL injection/XSS	Database input sanitazing
DoS	DDos mitigation

Figure 12: Counter measures table

5. Implementation

This chapter's main focus is the implementation details of "Add item to the list" user story, as it is one of the main functionalities connected to the system's core. The concept is graphically represented by a sequence diagram and broken down into smaller pieces, describing the procedure from the first tier to the third.

5.1. TIER 1 - C# .NET

Starting from the webpage, the user interface of the product, the part of the page where the user can input details about the item, is shown in the picture below. The structure of the pop-up and its form are based on HTML and CSS technologies binded to C# .NET classes.

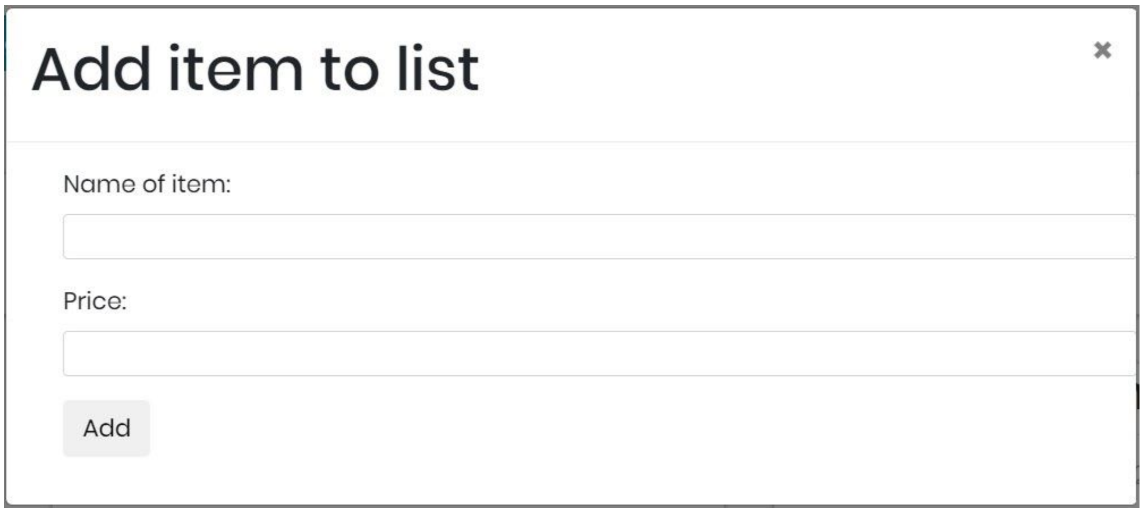
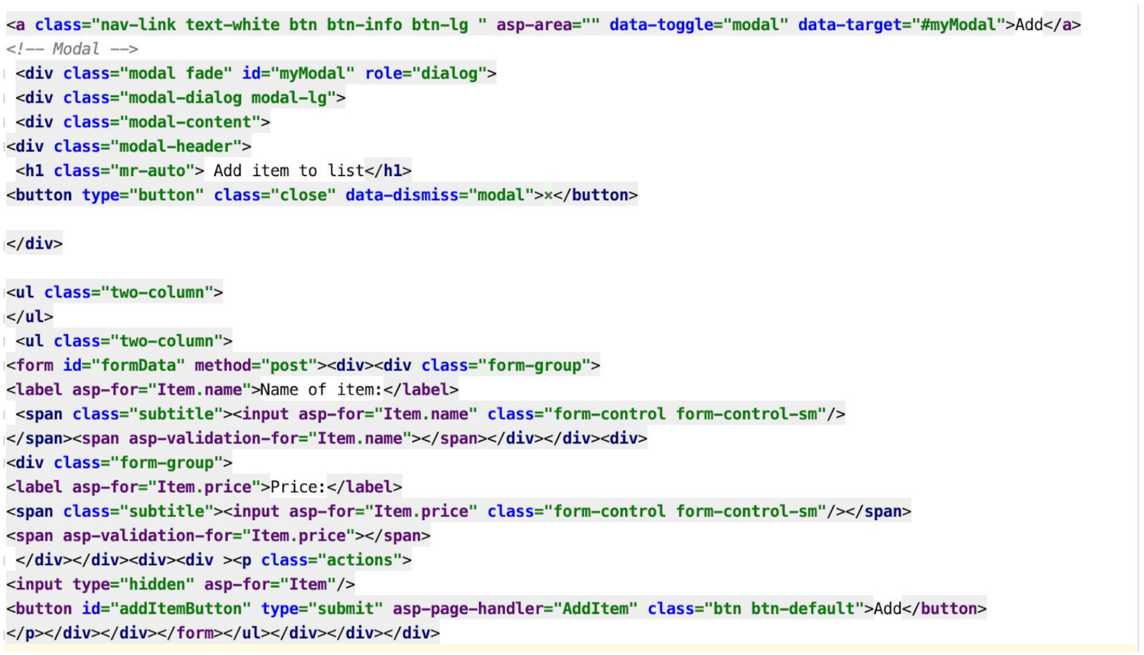


Figure 13: Add item to list

The markup code describes the integration of the graphical elements from the picture above. The input form is integrated in the popup element, so the page seems more interactive and acquiring the item's elements is more fluent.

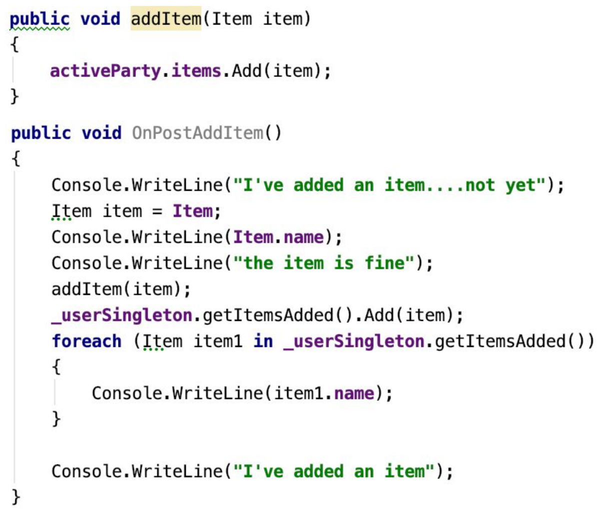


```
<a class="nav-link text-white btn btn-info btn-lg " asp-area="" data-toggle="modal" data-target="#myModal">Add</a>
<!-- Modal -->
 <div class="modal fade" id="myModal" role="dialog">
 <div class="modal-dialog modal-lg">
 <div class="modal-content">
<div class="modal-header">
 <h1 class="mr-auto"> Add item to list</h1>
<button type="button" class="close" data-dismiss="modal">×</button>

</div>

<ul class="two-column">
</ul>
 <ul class="two-column">
<form id="formData" method="post"><div><div class="form-group">
<label asp-for="Item.name">Name of item:</label>
 <span class="subtitle"><input asp-for="Item.name" class="form-control form-control-sm"/>
</span><span asp-validation-for="Item.name"></span></div></div><div>
<div class="form-group">
<label asp-for="Item.price">Price:</label>
<span class="subtitle"><input asp-for="Item.price" class="form-control form-control-sm"/></span>
<span asp-validation-for="Item.price"></span>
 </div></div><div><div ><p class="actions">
<input type="hidden" asp-for="Item"/>
<button id="addItemButton" type="submit" asp-page-handler="AddItem" class="btn btn-default">Add</button>
</p></div></div></form></ul></div></div></div>
```

Figure 14: Code snippet

Figure 15: Code snippet

The next construction is from the C# view class UserPage. The method OnPostAddItem is adding the item from the view to the singleton temporarily so it can later be saved when the save button is pressed.

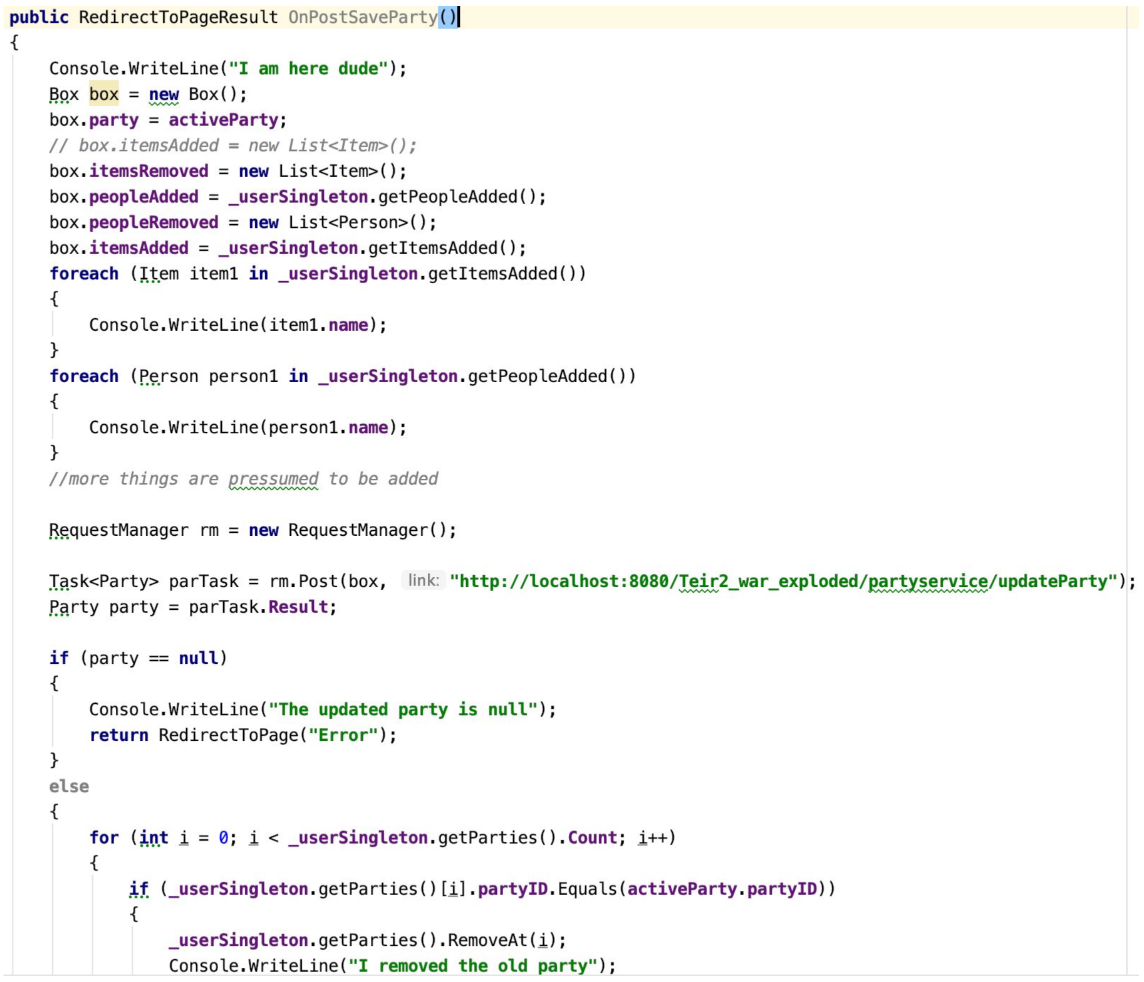


```
public void addItem(Item item)
{
    activeParty.items.Add(item);
}

public void OnPostAddItem()
{
    Console.WriteLine("I've added an item....not yet");
    Item item = Item;
    Console.WriteLine(Item.name);
    Console.WriteLine("the item is fine");
    addItem(item);
    _userSingleton.getItemsAdded().Add(item);
    foreach (Item item1 in _userSingleton.getItemsAdded())
    {
        Console.WriteLine(item1.name);
    }

    Console.WriteLine("I've added an item");
}
```

Figure 16: Code snippet

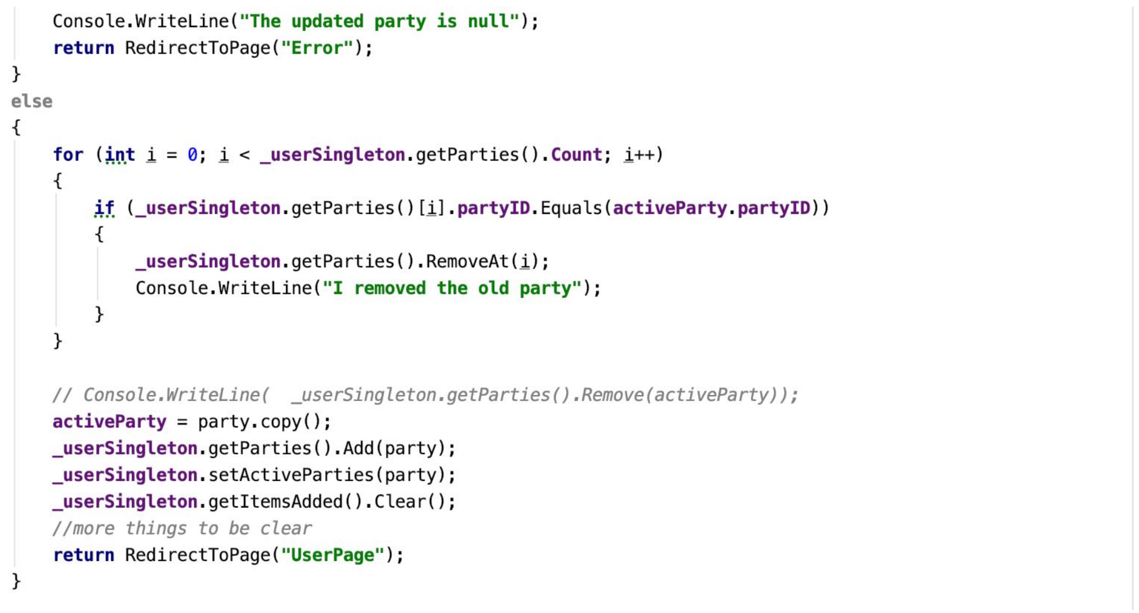
The ‘save’ functionality, is handled by OnPostSaveParty, which stays behind the floating save button in the graphical interface. The method works like a controller of the class passing all the edited data that is stored on the singleton to the second tier through de method Post() which part of the class RequestManager. In essence the method creates an instance of a Box object to which it is being added the activeParty property of the singleton and the temporarily stored Items in the itemsAdded properties of the singleton which is passed onto the system for execution.

If this is successful, the Singleton is cleared, the active party is updated and the host is redirected to the UserPage again but with the updated information of the party.



```
public RedirectToPageResult OnPostSaveParty()
{
    Console.WriteLine("I am here dude");
    Box box = new Box();
    box.party = activeParty;
    // box.itemsAdded = new List<Item>();
    box.itemsRemoved = new List<Item>();
    box.peopleAdded = _userSingleton.getPeopleAdded();
    box.peopleRemoved = new List<Person>();
    box.itemsAdded = _userSingleton.getItemsAdded();
    foreach (Item item1 in _userSingleton.getItemsAdded())
    {
        Console.WriteLine(item1.name);
    }
    foreach (Person person1 in _userSingleton.getPeopleAdded())
    {
        Console.WriteLine(person1.name);
    }
    //more things are pressumed to be added

    RequestManager rm = new RequestManager();

    Task<Party> parTask = rm.Post(box,  link: "http://localhost:8080/Teir2_war_exploded/partyservice/updateParty");
    Party party = parTask.Result;

    if (party == null)
    {
        Console.WriteLine("The updated party is null");
        return RedirectToPage("Error");
    }
    else
    {
        for (int i = 0; i < _userSingleton.getParties().Count; i++)
        {
            if (_userSingleton.getParties()[i].partyID.Equals(activeParty.partyID))
            {
                _userSingleton.getParties().RemoveAt(i);
                Console.WriteLine("I removed the old party");
```

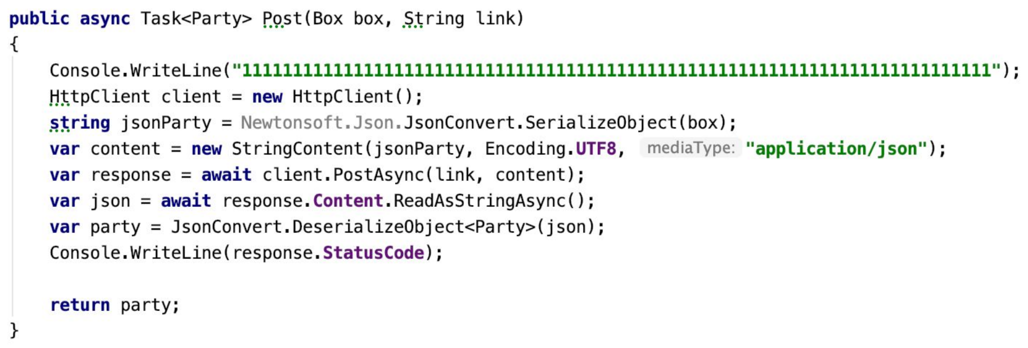
Figure 17: Code snippet



```
        Console.WriteLine("The updated party is null");
        return RedirectToPage("Error");
    }
    else
    {
        for (int i = 0; i < _userSingleton.getParties().Count; i++)
        {
            if (_userSingleton.getParties()[i].partyID.Equals(activeParty.partyID))
            {
                _userSingleton.getParties().RemoveAt(i);
                Console.WriteLine("I removed the old party");
            }
        }

        // Console.WriteLine(  _userSingleton.getParties().Remove(activeParty));
        activeParty = party.copy();
        _userSingleton.getParties().Add(party);
        _userSingleton.setActiveParties(party);
        _userSingleton.getItemsAdded().Clear();
        //more things to be clear
        return RedirectToPage("UserPage");
    }
```

Figure 18: Code snippet

The final point of the first tier   data processing is in the Request manager class, where all the I/O methods for communication with API are passed through.



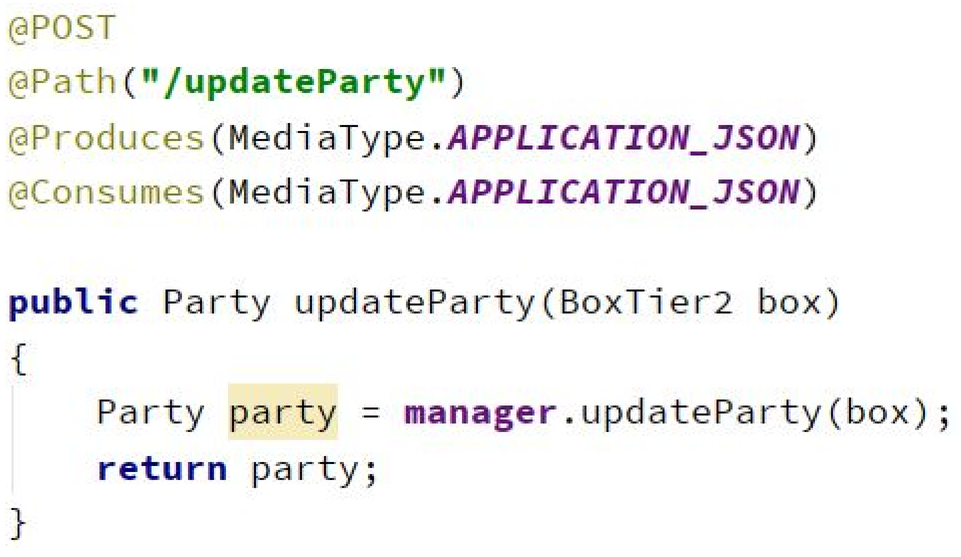
```
public async Task<Party> Post(Box box, String link)
{
    Console.WriteLine("1111111111111111111111111111111111111111111111111111111111111111111111111111");
    HttpClient client = new HttpClient();
    string jsonParty = Newtonsoft.Json.JsonConvert.SerializeObject(box);
    var content = new StringContent(jsonParty, Encoding.UTF8,  mediaType: "application/json");
    var response = await client.PostAsync(link, content);
    var json = await response.Content.ReadAsStringAsync();
    var party = JsonConvert.DeserializeObject<Party>(json);
    Console.WriteLine(response.StatusCode);

    return party;
}
```

Figure 19: Code snippet

This class acts as a controller for the first tier so all the view classes are passing objects here, in this case a box containing the changes made in the party, to be sent to the second, the Java business logic tier.

5.2. TIER 2 - Java

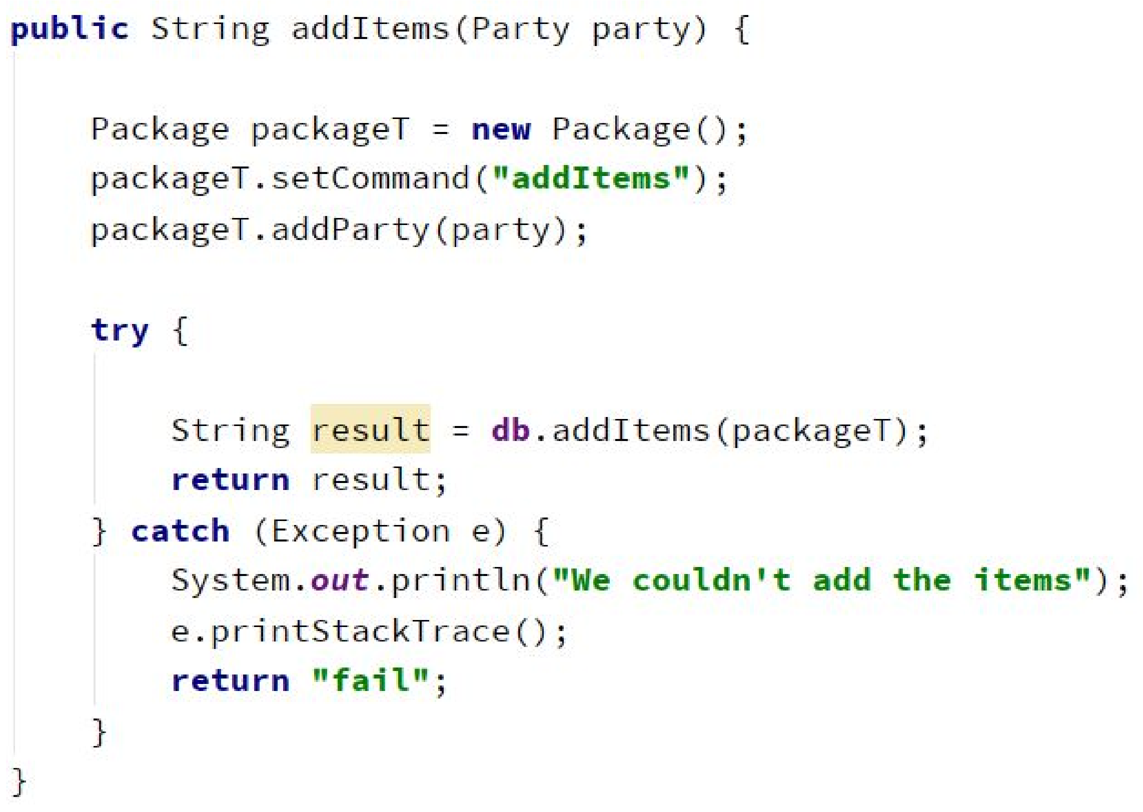
As seen in the previous chapter data is sent through the post method and received by the 'PartyPlanner' class which acts as a facade for the rest of the system. Since the use case of adding items to a party has been chosen as the one to follow throughout this report all the examples below will include the relevant implementation for this method at every step of the 'items' way towards being successfully stored in the database. The following picture shows the 'PartyPlanner' class in the second tier and its method of 'updateParty' which receives the information from the first tier, calls the tier ModelManager and forwards the information retrieved and



```java
@POST
@Path("/updateParty")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)

public Party updateParty(BoxTier2 box)
{
    Party party = manager.updateParty(box);
    return party;
}
```

Figure 20: Code snippet

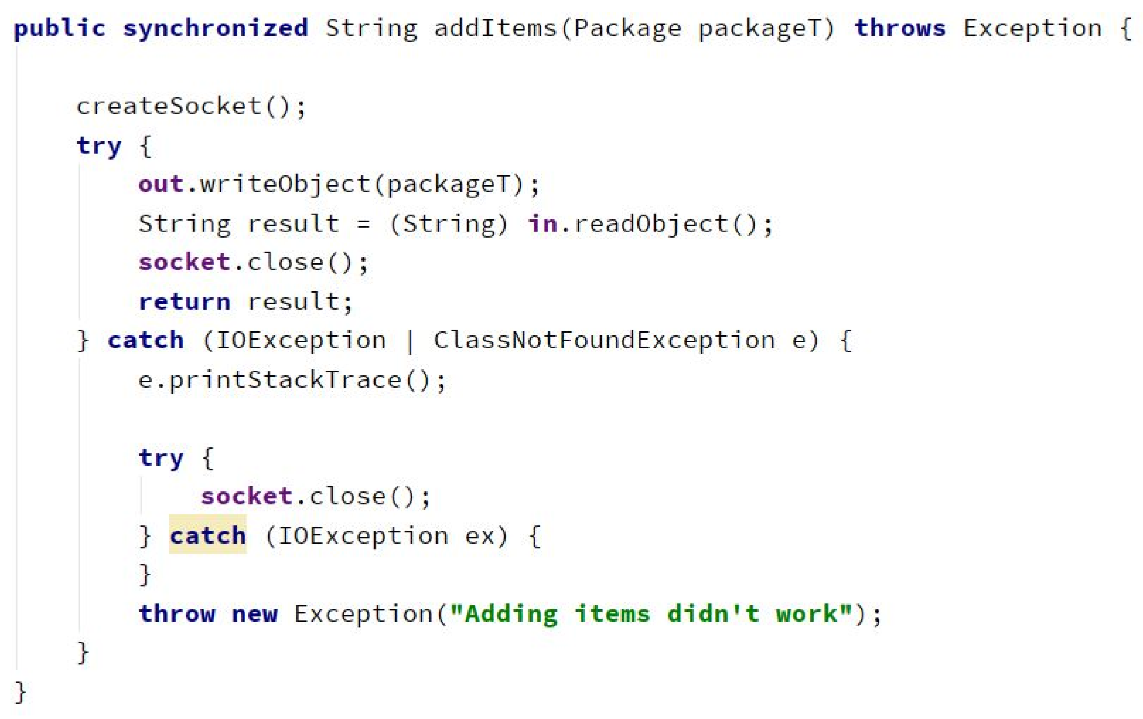
Following the data flow gets us to the 'ModelManager' class for tier two in which the procedure for packaging the 'party' and the list of "items" and coupling it with a 'command' is handled. This command is meant to be interpreted by the tier three's model manager. In this case, the package has the command "addItems" so that the socket handler knows what to do with it.



```java
public String addItems(Party party) {

    Package packageT = new Package();
    packageT.setCommand("addItems");
    packageT.addParty(party);

    try {

        String result = db.addItems(packageT);
        return result;
    } catch (Exception e) {
        System.out.println("We couldn't add the items");
        e.printStackTrace();
        return "fail";
    }
}
```

Figure 21: Code snippet

Lastly in tier two, is the ‘Database connection’ class which handles the socket creation and the following transmission of data. It calls ‘createSockets’ method in preparation for sending a ‘package’ as its argument. Afterwards it sends out the package using the ‘ObjectOutputStream’ variable. The procedure on following tiers is managed in synchronized manner and so, the method waits for the result and then closes the socket connection. For more information the source code disposes of a more detailed version javadoc.

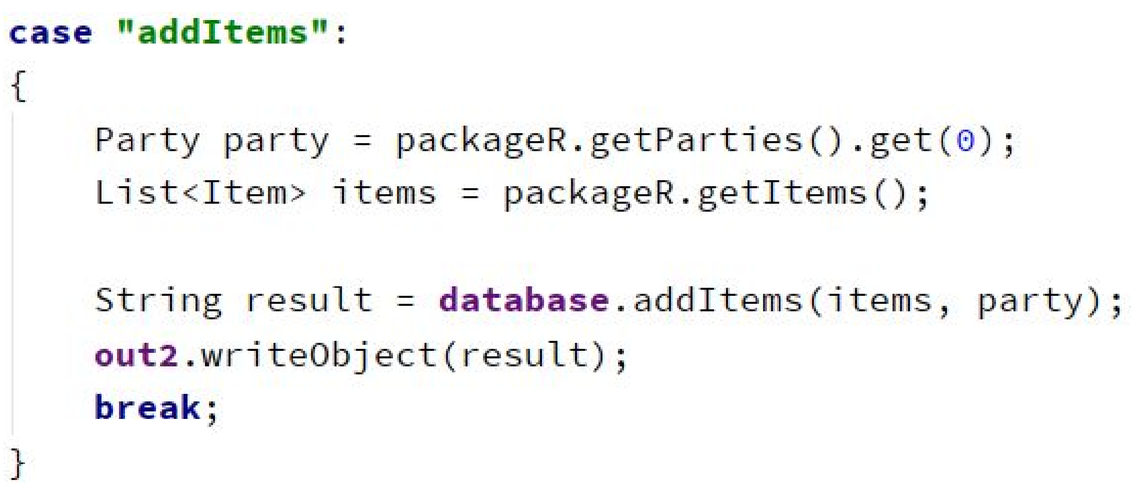


```
public synchronized String addItems(Package packageT) throws Exception {

    createSocket();
    try {
        out.writeObject(packageT);
        String result = (String) in.readObject();
        socket.close();
        return result;
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();

        try {
            socket.close();
        } catch (IOException ex) {
        }
        throw new Exception("Adding items didn't work");
    }
}
```

Figure 22: Code snippet

5.3. TIER 3 - PostgreSQL

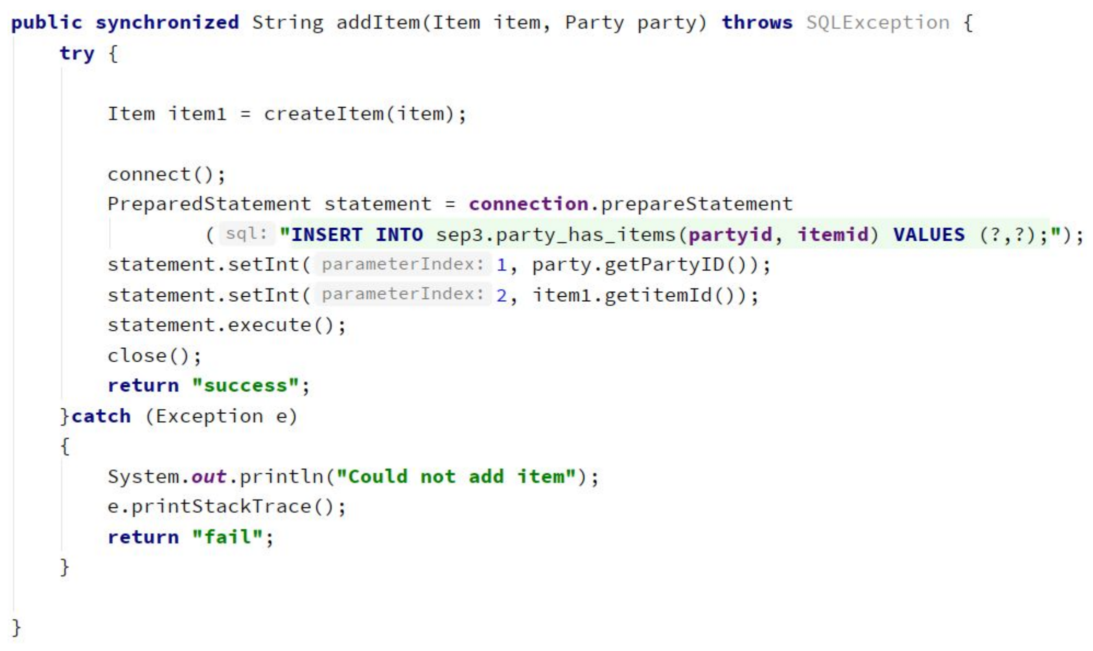
Getting into tier three we find ourselves at the point where the system has to decide how to proceed according to the previously mentioned ‘command’. In the picture below the case for ‘addItems’ can be seen. It extracts the ‘party’ object from the package alongside all the ‘items’ and stores them in a list. Both of these variables are then sent to the last point of this sequence: the database access class.



```java
case "addItems":
{
    Party party = packageR.getParties().get(0);
    List<Item> items = packageR.getItems();

    String result = database.addItems(items, party);
    out2.writeObject(result);
    break;
}
```

Figure 23. Code snippet

Here we can see the class which is used to access the actual database through the JDBC connection that is provided from the PostgreSQL driver and initialized by calling the ‘connect’ method just before attempting to prepare a SQL statement which after inputting all of the necessary variables shall be executed in the database.



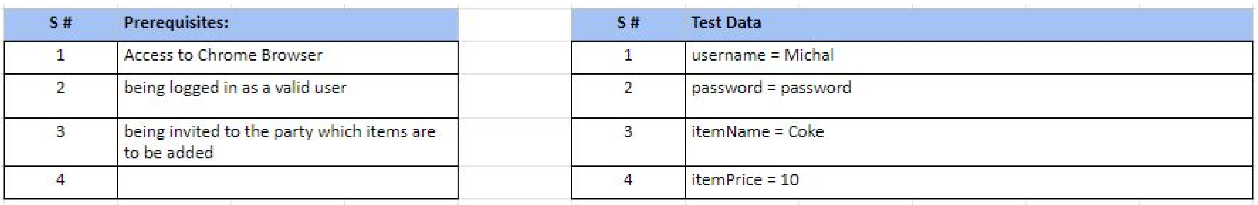
```java
public synchronized String addItem(Item item, Party party) throws SQLException {
    try {

        Item item1 = createItem(item);

        connect();
        PreparedStatement statement = connection.prepareStatement
                ( sql: "INSERT INTO sep3.party_has_items(partyid, itemid) VALUES (?,?);");
        statement.setInt( parameterIndex: 1, party.getPartyID());
        statement.setInt( parameterIndex: 2, item1.getitemId());
        statement.execute();
        close();
        return "success";
    }catch (Exception e)
    {
        System.out.println("Could not add item");
        e.printStackTrace();
        return "fail";
    }

}
```

6. Test

The purpose of testing the system in this case is to find possible flaws that might have been missed during the Implementation phase. In an ideal real world scenario, continuous testing methods and/or frameworks would be preferred to the ones described in this document. However during the development of this project this technique was not employed mainly because of the time constraints and difficulty of implementing and deploying such a solution.

6.1. Test Specifications

The test specifications are based on the need to verify the validity of inputs, which can be seen in the 'Test Data' column of the table below, and are made in a way that ensures having a unified and stable test environment. The table below also shows the steps that have been taken in order to create such testing environment in the 'Prerequisites' column.



S #	Prerequisites:
1	Access to Chrome Browser
2	being logged in as a valid user
3	being invited to the party which items are to be added
4	

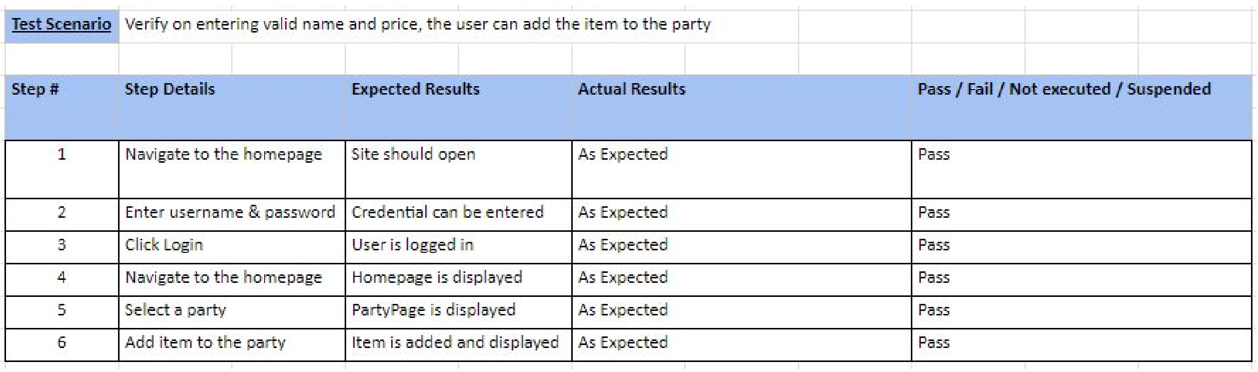
S #	Test Data
1	username = Michal
2	password = password
3	itemName = Coke
4	itemPrice = 10

Figure 24. Testing table

6.2. Black Box

The approach that has been chosen for this project is based purely on a form that is supposed to be filled in by the tester who is performing the test. Therefore it is not the most accurate representation of the state in which the system is in at the point of test.

However what it does provide is some insight into the most ideal behaviour of a potential user and how their actions affect the functionality. The table below represents a ‘best case scenario’ for the use case ‘Add Item’. It includes the steps taken by the tester in a numbered fashion, results that are expected to happen based on each action taken, actual results according to the testers best judgment and a ‘grade’ is assigned each step after its completion.



Test Scenario	Verify on entering valid name and price, the user can add the item to the party			
Step #	**Step Details**	**Expected Results**	**Actual Results**	**Pass / Fail / Not executed / Suspended**
1	Navigate to the homepage	Site should open	As Expected	Pass
2	Enter username & password	Credential can be entered	As Expected	Pass
3	Click Login	User is logged in	As Expected	Pass
4	Navigate to the homepage	Homepage is displayed	As Expected	Pass
5	Select a party	PartyPage is displayed	As Expected	Pass
6	Add item to the party	Item is added and displayed	As Expected	Pass

Figure 25. Testing table

7. Results and Discussion

The results of implementation proved the correctness of the heterogeneous system design. The end product has a working database, connected through Sockets to the business logic tier that is exposing web services and a first tier consuming these from the API, displaying the necessary information to the user. All these are according to the design chosen.

The system is successfully passing data between multiple tiers and could be easily expanded with more functionality and features thanks mainly to this design. Besides the custom Java API, Spotify service has been implemented to allow users the ability to use their own custom playlists for each party.

A combination of a custom REST-based API and an external service such as Spotify allowed even more possibilities to create an upgradeable and possibly scalable, therefore successful product.
The system is built on two main pillars that serve as a foundation for the whole system. Simple structure of the user interface and correct responses from APIs.

Overall, the system fulfills the requirements set initially and performs as intended, having a structured code that is manageable and readable as possible.

8. Conclusions

The project's product is a platform for event planning, mainly oriented towards student parties. A group of potential customers already interacted with the software and appreciated the overall atmosphere and functions of the product. Judging from this, the project has been successful.

The knowledge gained can be separated into different areas. Specifically highlighted: management of human resources, proper choice of the design patterns or coding cooperation and sharing through Github. The version control versatility improved the stability of the system and helped avoid misconceptions. Strong inner responsibility, management, and open communication was needed for successful cooperation. All of those aspects, followed by proper time management, proved the group's ability to divide tasks and understated work abilities.

All the technical aspects are followed as much as possible considering the time scale. Every new part of the technology was learned through discussion, usually based on the interest and responsibility of the member using it at the time.

Every tier has its own purpose and major technology, based on the institutional requirements.

The group sees potential improvements for this type of concept in the future, which are described in the project future.

9. Project future

The final product that has been delivered is functional and major bugs have been fixed before the final submission. However there still persist some minor bugs which do not affect the major functionality.

Nonetheless the time scale proved to be a challenge for learning all the new technologies properly and implementing them in a way that would be satisfying for the team.

Therefore the future would most definitely entail some improvements to the underlying code base and some tweaks to the UI to streamline the UX and make it more understandable to the regular user.

Another possibility for future development would be to develop a business model and figure out a way to monetize the platform without driving away potential customers.

10. Appendices