INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

INSTITUTO POLITÉCNICO DE LISBOA
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Event Organizer

Marco Batista n.° 42125, e-mail: `a42125@alunos.isel.pt`

Diogo Martins n.° 42393, e-mail: `a42393@alunos.isel.pt`

Supervisor: Nuno Leite

Projeto e Seminário

Report

September 2019

Abstract

There is a large library of platforms and apps that allow one to invite people to an event. These platforms, however, have some downsides, for example, splitting expenses of an event is hard to do on those platforms, they require new users to make an account and activating it, are not available on multiple platforms, among other issues. The goal of this project is to develop a new platform that tackles these disadvantages. This way the group implemented a new solution using modern platforms (such as ASP.NET CORE and Xamarin) that can be used by a large number of people on modern devices and can later be expanded to new environments that might be created. The group was presented with some challenges that had to be overcome, like the authentication without account creation using multiple providers, the notifications mechanism, or the code sharing between platforms. The result is a new application supported by a Web API without the need to create a new account on another platform, using the already existing accounts on other services like Facebook and Microsoft's Outlook, for example.

Keywords: Events, Item, Task, User, Invitation.

List of Acronyms

CRUD	Create, Read, Update and Delete
IDE	Integrated development environment
ORM	Object-Relational Mapping

Contents

List of Figures

Chapter 1

Introduction

While there are some applications that allow for the creation of events, invitations and publishing, none of them focus on what really happens during and in the planning of an event.

In a real case scenario like the organization of a *party* type of event, there are some important factors to take into account. For this type of events it's possible to list some basic needs like: budget management (expenses like food and drinks), location, or entertainment (such as music).

The purpose of this project consists of simplifying the planning and management of events having the focus on people collaboration through a mobile application, while keeping it possible to expand to other platforms.

The proposed solution is comprised of a Web API and a client application. The Web API is supported by .NET CORE MVC and a PostgreSQL SQL database engine. The client application is built using Xamarin Forms, with aim especially for Android devices and Windows devices. All operations (such as creating events, items, tasks, etc...) happen in the API and the client application communicates with it using HTTP.

The remainder of the report is organized in seven chapters. Chapter 2 presents the State of the Art. In Chapter 3, the project functionality is presented. Chapter 4 describes the development support used in the project. In Chapter 5 the project roadmap is analyzed. Chapter 6 draws some conclusions and gives pointers to future work.

Chapter 2

State of the art

It’s possible to identify some applications that include an event planning function: *Facebook (2019)*, *Doodle*, *Planner (2019)*, and *Asana (2019)*. The first two listed, *Facebook* and *Doodle*, both support the creation of events, but they focus on scheduling while also allowing for some basic functionalities like invitation of people to the event.

Super Planner allows for more complex event management, providing calculators for venue capacity, food/catering and staffing, as well as others for managing a budget limit.

Asana can be used to assign tasks to people and manage the work that needs to be done. However, none of these have a focus on collaboration between people focused on an event. While *Asana* has the capability of assigning tasks between people, it serves more as a work management tool to organize teams in business projects.

Super Planner has good tools to manage an event, but lacks collaboration between people and invitation of people to the event is not possible.

Facebook and Doodle are great tools for basic event planning, but do not offer much more than setting a location, date and giving a description of the event. It is important to mention that *Facebook* does offer some tools that could prove useful, such as posting a message in the event page, but that are not specifically made for this purpose and end up being too cumbersome.

The current state of the art now counts with a really similar application to what we are proposing, an application called *Eventbrite (2019)*. This application has a really similar concept but in a different way. First of all it implements a ticket system. Defining an event requires you to make a ticket price, which is not ideal since the group wants the event’s expenses to be modifiable throughout the event, implementing a voting system for it if wanted. There are others that are more focused on team management, for projects, like *Cvent (2019)* and *Events (2019)* and then there are ones like *EventBrite* that implement a ticket system like *Ticket Taylor*.

Also one has to create an account to use this platform, which is not ideal because people are normally reluctant to create accounts on platforms as there are too many in the current days.

As far as the group knows there are none that implement a division of tasks, expenses and playlist creation that have a focus on collaboration of people, as well as multimedia functionality. These features make sense together in an event planning app as a big volume of events contain all of these aspects in common.

Chapter 3

Requirements

Contents

3.1 Functional Requirements

3.1.1 Mandatory Requirements

The mandatory functional requirements are the following:

- Creation of an Event;
- Definition of the location of the event using a maps platform (for example, *Google Maps*);
- Guests invitations;
- Guest task managing;
- Event expense managing (including the voting system);
- *Playlist* creation.

3.1.2 Optional Requirements

- Ease of payment through services like *PayPal* or *MbWay*;
- Ease of expense managing by adding products through their barcode;
- Deployment of the APIs on the cloud (for example: *Azure* or *Google Cloud*)

3.2 Functionallity

The platform allows the user to signup. Figure 3.1 illustrates the *User Signup* operation.

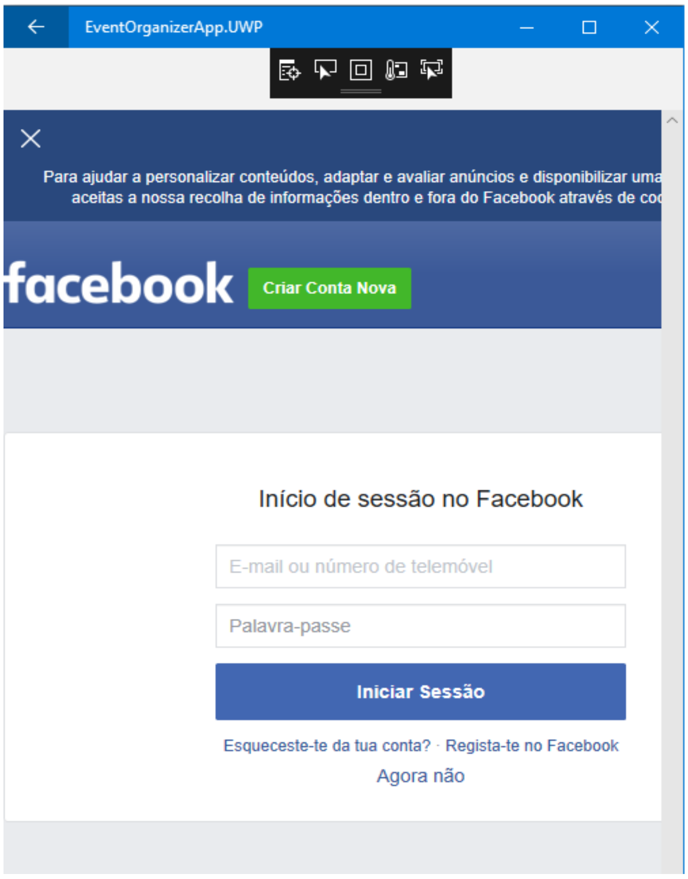

EventOrganizerApp.UWP
Para ajudar a personalizar conteúdos, adaptar e avaliar anúncios e disponibilizar uma
aceitas a nossa recolha de informações dentro e fora do Facebook através de coo
facebook
Criar Conta Nova
Início de sessão no Facebook
E-mail ou número de telemóvel
Palavra-passe
Iniciar Sessão
Esqueceste-te da tua conta? · Regista-te no Facebook
Agora não

Figure 3.1: User Signup Screen

Users are redirected to the facebook login page to signup. It's possible to use *Facebook* as an identity provider. Once a user is signed in, it has the ability to create an event, giving its Title and Description, also the start and end dates. After the event is created, it can add its initial expenses/items so the invited users know if they need to pay for anything in advance, and how much.

Figure 3.2 illustrates the *Event creation* operation.

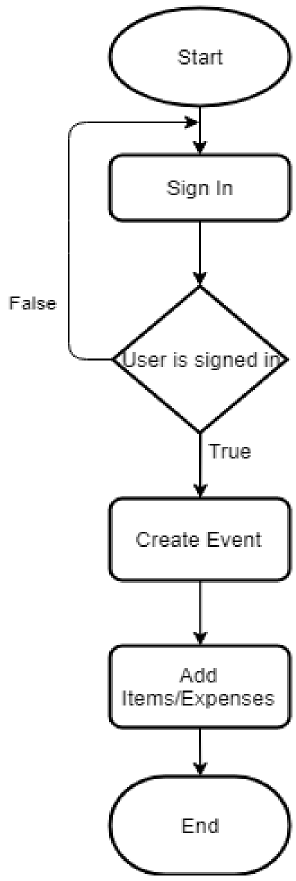
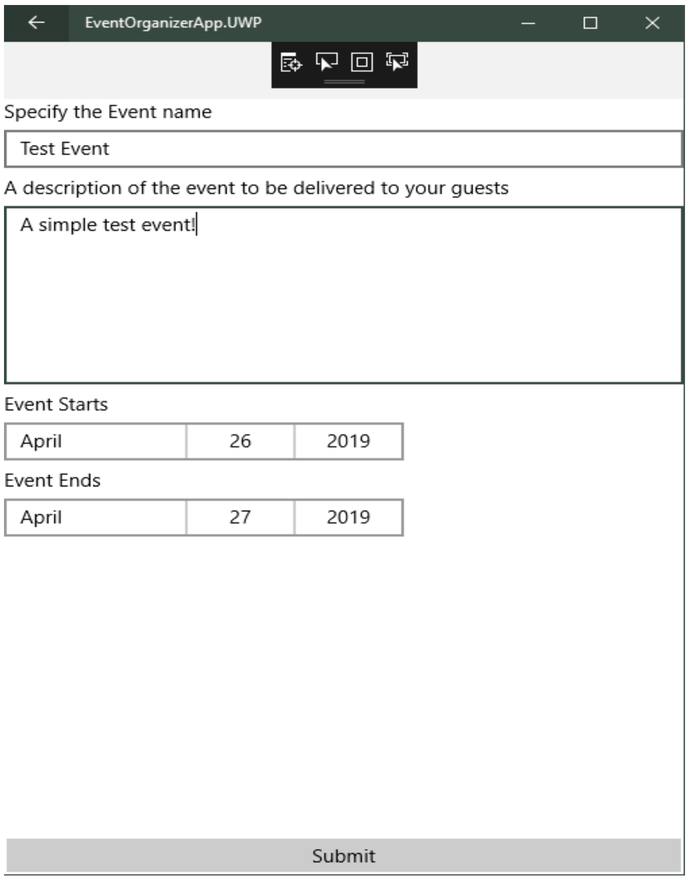


Figure 3.2: Create Event Flowchart

Figure 3.3 illustrates the *Event creation* screen.


EventOrganizerApp.UWP

Specify the Event name

Test Event

A description of the event to be delivered to your guests

A simple test event!

Event Starts

April 26 2019

Event Ends

April 27 2019

Submit

Figure 3.3: Event Creation Screen

To add an item the user has to choose an event and then specify a title, description, price (if the item has a price) and it’s type (types can be: *Food*, *Drink*, *Decoration*, *Taxes* and *Others*).

Figure 3.4 illustrates Item Creation.

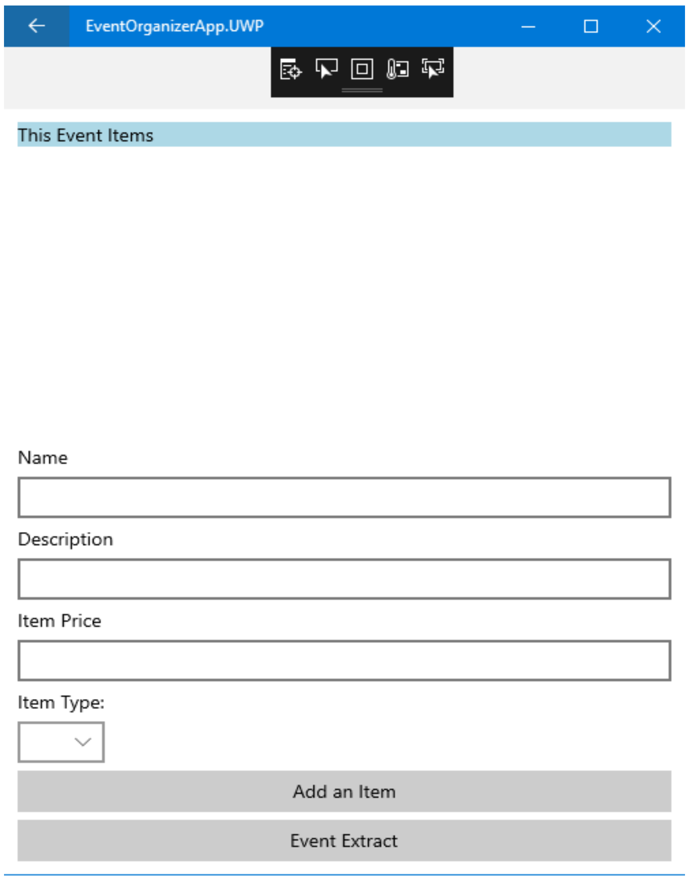


Figure 3.4: Item Creation Screen

When items are created in the event a guest can see the event extract. This screen will present to the guest who is paying for what items and how much.

By selecting an item, inputting how much one has payed for it and clicking "Change Payed Value" an entry will be added to the event extract so other guests can see how much the others payed and how much it has to pay also.

Figure 3.5 illustrates Event Extract.

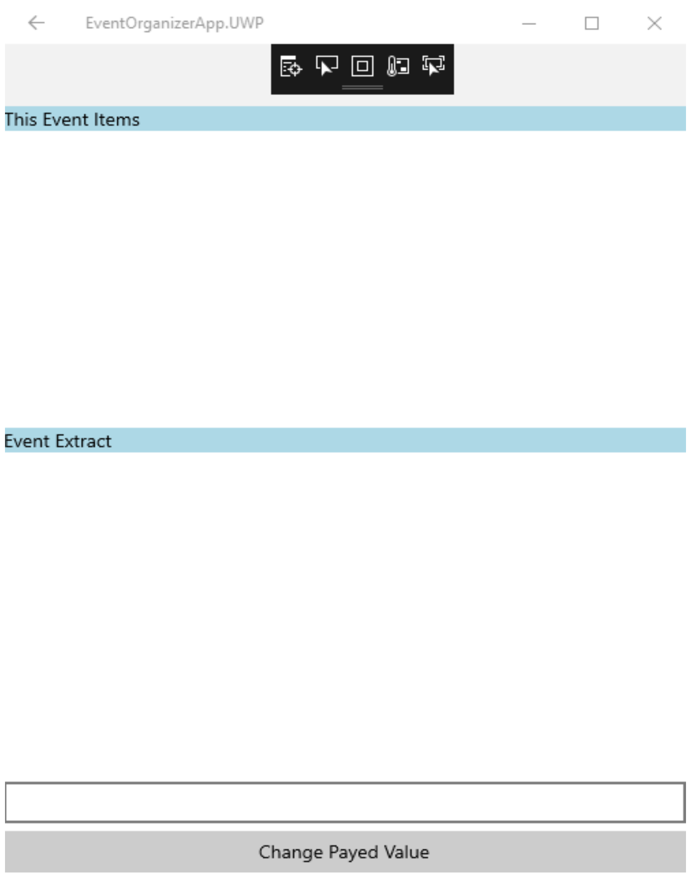


Figure 3.5: Item Creation Screen

After the user has at least one event created it can see all the events that it created until now and when the user chooses an event, it can list and edit items aswell as invite users:

Figure 3.6 illustrates the Event edit.

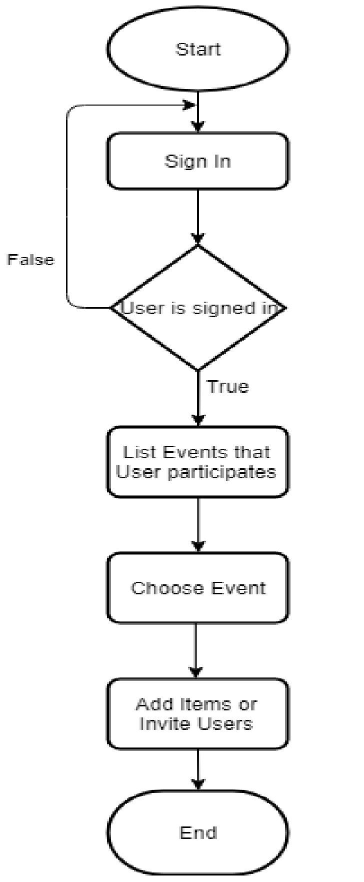
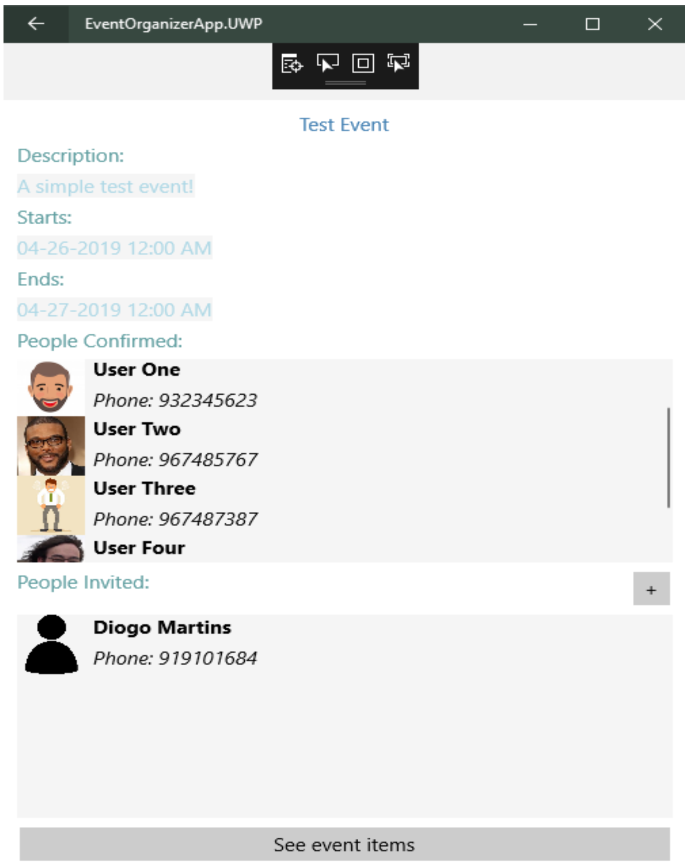


Figure 3.6: Edit Event Flowchart

Figure 3.7 illustrates the Event edit.



EventOrganizerApp.UWP

Test Event

Description:

A simple test event!

Starts:

04-26-2019 12:00 AM

Ends:

04-27-2019 12:00 AM

People Confirmed:

User One
Phone: 932345623

User Two
Phone: 967485767

User Three
Phone: 967487387

User Four

People Invited:

Diogo Martins
Phone: 919101684

Figure 3.7: Edit Event Screen

It’s also possible to add Tasks to users. Tasks are something that have to be completed until a certain expiration date. Tasks are comprised of a name, description and expiration date. They are also associated with a user so when a task is assigned to one they receive a notification.

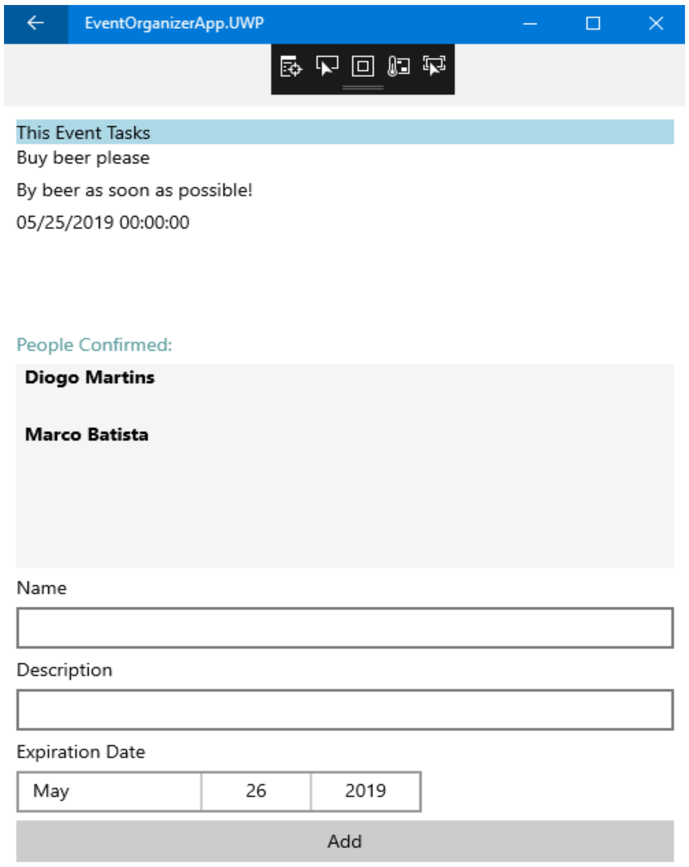
Figure 3.8 illustrates adding a Task to an Event.

3.3 Events Web API

The design of the API follows a REST approach over HTTP. This was decided so that the client/server side of the project can be *scalable* and *loosely coupled*, meaning that in the future, more clients for this API can be developed without the need to change anything to accommodate them.

In the current version, the Events API has the endpoints presented in Figure 3.9.

The API is using Entity Framework, ADO.NET as the technologies to access data of the groups PostgreSQL database.



This Event Tasks

Buy beer please

By beer as soon as possible!

05/25/2019 00:00:00

People Confirmed:

Diogo Martins

Marco Batista

Name

Description

Expiration Date

May 26 2019

Add

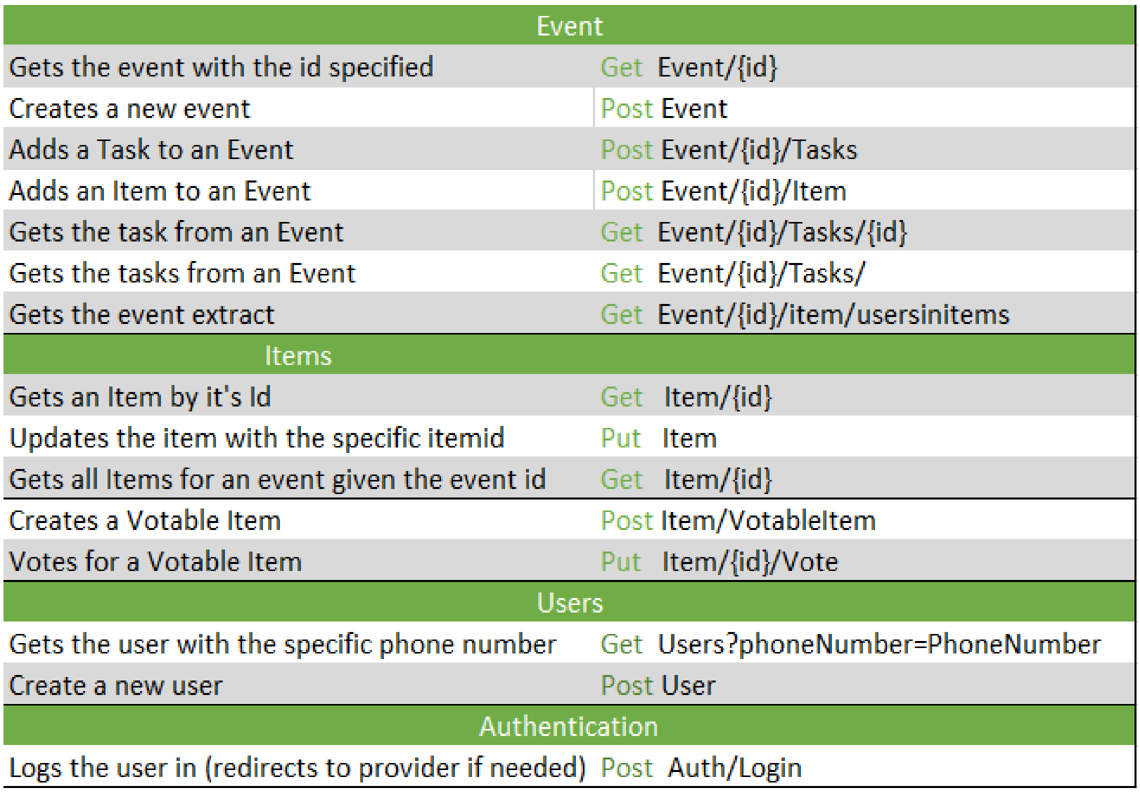
Figure 3.8: Edit Event Screen

Implementation of the API

Basic API structure The API, as mentioned earlier, is based on .NET Core. The controllers are called by the Http requests and the services that they use are initialized by the Dependency Injection engine that the framework provides. The dependencies are all configured on the Startup.cs file of the project. The services contain all the business logic of the API and when they need to reach data to operate upon these call the methods exposed by the repository wrapper implemented in the API.

The repository wrapper allows for easy access to data repositories, being similar to a chest of items where one can get all of the data they need centralized in one class.

All of the repositories extend RepositoryBase<T> which exposes all the *Create, Read, Update and Delete* (CRUD) operations while allowing for deferred execution when querying the database. Deferred execution allows for creating the query and only executing it against the database when a terminal method is called (for example .ToList()). This allows for a reduction of the queries ran on the database meaning it's possible to get all the data from a custom query only on one travel to the database server.



Event		
Gets the event with the id specified	Get	Event/{id}
Creates a new event	Post	Event
Adds a Task to an Event	Post	Event/{id}/Tasks
Adds an Item to an Event	Post	Event/{id}/Item
Gets the task from an Event	Get	Event/{id}/Tasks/{id}
Gets the tasks from an Event	Get	Event/{id}/Tasks/
Gets the event extract	Get	Event/{id}/item/usersinitems
Items		
Gets an Item by it's Id	Get	Item/{id}
Updates the item with the specific itemid	Put	Item
Gets all Items for an event given the event id	Get	Item/{id}
Creates a Votable Item	Post	Item/VotableItem
Votes for a Votable Item	Put	Item/{id}/Vote
Users		
Gets the user with the specific phone number	Get	Users?phoneNumber=PhoneNumber
Create a new user	Post	User
Authentication		
Logs the user in (redirects to provider if needed)	Post	Auth/Login

Figure 3.9: Events API endpoints in use by client application

Logging The API allows for logging by using a group implemented Logging Provider and adding it to the Startup on configuration using *ILoggerFactory*. The logging level is controlled by the appsettings file of the project where there are 7 levels:

- Trace
- Debug
- Information
- Warning
- Error
- Critical
- None

Logging is called (or not) automatically by the platform according to the level defined in the settings.

In case of an occurrence of an unhandled exception there is an ErrorController than has an Error method that is a generic exception handler. This exception handler returns a 500 Http Status Code and before it returns to the user it sends an email to the groups application email box with a Message and StackTrace of the Exception. If the logging level is defined for every one except None it also gets logged to the logging database.

The logging database DAL is implemented using ADO.NET because of the fast execution of this technology while accessing a database. The tradeoff here is that this performance, which allows for the logging to be fast and less impact-full of the API performance overall, comes at a cost of the flexibility and code maintenance that Entity Framework provides.

Email Sending The email sending is used as the main notification system. It is used in the following situations:

- When a user is registered an email is sent to its mailbox
- Upon the creation of an item or a task (sending the task to the person whom that task was assigned too)
- Sent to the group's mailbox when an Exception happens.

for when a new user is registered (sending a welcome email to the user mailbox) and on an unhandled exception. The methods defined on this service are asynchronous methods so they run in the background as SMTP communication usually isn't very fast so this way the API overall performance is not impacted a lot in a negative way. The class the group is using to send emails is *SmtpClient* and it's configuration is given by the appsettings file. The email provider the group is using is *Outlook*.

ModelState Validation This API takes advantage of the technology of model validation present in the framework. This validation technology goes by the name of ModelState Validation. This technique consists on decorating the models used by the client application with Data Annotations. These Data Annotations are placed in the properties of the model as Attributes. These can define if a property is required, what is the maximum length of a property (for example, in case of a string), the value range, etc... The client application sends the model and when the request reaches the web API controller, one can check if the model sent by the client app is valid, by calling ModelState.IsValid, before sending the request to the other layers of the API (service, data, etc...). If ModelState.IsValid returns true, then the model obeys to all validations described in the data annotations. However, if false, the request isn't sent to the other layers, and on the response a body containing the validation errors will be sent (accompanied by a HTTP 400 – Bad Request – status code). These validation errors will help the client understand what is wrong with it's request so they can correct it.

3.4 Music Web Api

The *Music Web API* is a set of definitions that aim to unify music providers by having them expose common functionalities through a single *Web API definition*.

3.4.1 Structure

The *Music Web API* repository contains three main projects that are:

- MusicWebApi – This project contains the interfaces that need to be implemented in order to follow the Music Web Api definition.
- MusicWebApi.Client – .NET Standard project that eases the use of Music Web Api Services by abstracting HTTP requests in simple method calls.
- MusicWebApi.Models – Another .NET Standard project shared by the first two containing the model objects for the API requests and responses.

In addition to these projects there are also projects that contain the implementations of the *Music Web API*, such as *SpotifyMusicWebApi*, an implementation for the popular Spotify music service.

3.4.2 Definition

As previously mentioned, the *Music Web API* is an API definition for services that wish to expose the functionality of music providers, existing or native, in a uniform and standard interface.

Music applications can make use of this if there is a need to have multiple music providers without having to create multiple user interfaces or API access layer implementations. This can prove useful in both the maintainability of the code by keeping it uncluttered and centralized (only one API access service is needed) and in keeping the application easily expandable simply by adding to the music services available, without the need of big code changes.

The following diagram exemplifies the functionality of the *Music Web API* ecosystem described earlier:

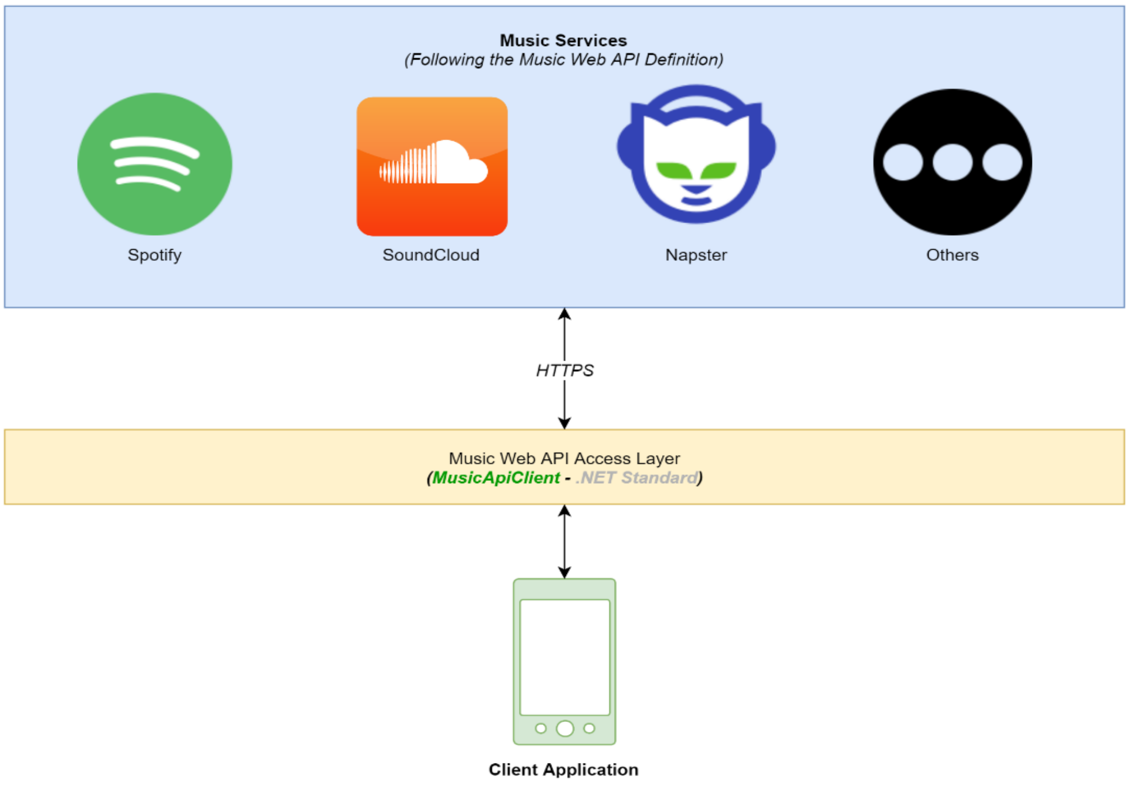
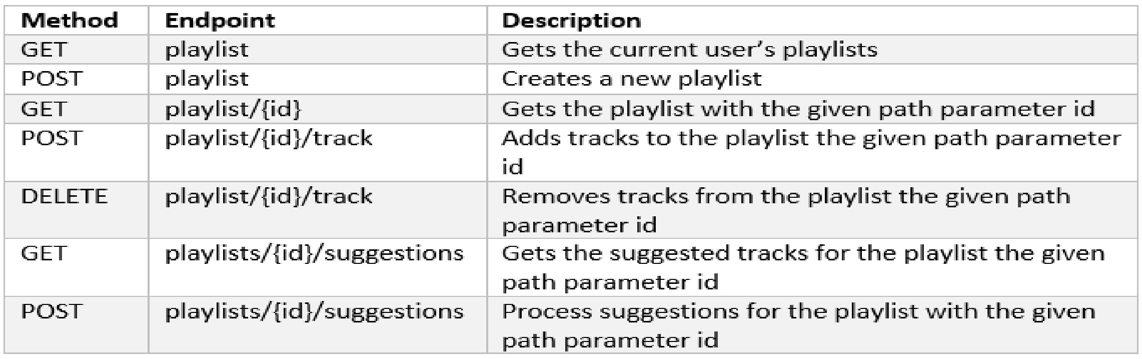
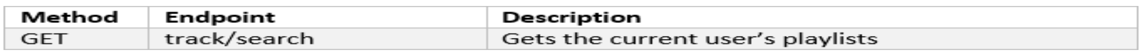

Music Services
(Following the Music Web API Definition)
Spotify
SoundCloud
Napster
Others
HTTPS
Music Web API Access Layer
(MusicApiClient - .NET Standard)
Client Application

Figure 3.10: Diagram of the Music Web API functionality



Method	Endpoint	Description
GET	playlist	Gets the current user's playlists
POST	playlist	Creates a new playlist
GET	playlist/{id}	Gets the playlist with the given path parameter id
POST	playlist/{id}/track	Adds tracks to the playlist the given path parameter id
DELETE	playlist/{id}/track	Removes tracks from the playlist the given path parameter id
GET	playlists/{id}/suggestions	Gets the suggested tracks for the playlist the given path parameter id
POST	playlists/{id}/suggestions	Process suggestions for the playlist with the given path parameter id

Figure 3.11: Playlist Controller



Method	Endpoint	Description
GET	track/search	Gets the current user's playlists

Figure 3.12: Track Controller

The decision to keep the *create playlist* and *add tracks to playlist* endpoints working with the POST method instead of PUT was made because the *Music Web API* definition cannot guarantee that, in the case where implementations use third party providers such as Spotify, the creation and addition of tracks is idempotent.

Although it makes the API less fault tolerant overall, it is a necessary trade off to make it broader by supporting implementations that use third party services.

3.4.3 Authentication and authorization

All music services that follow the definition *Music Web API* must use the industry-standard protocol, OAuth 2.0, for authorization.

As such among the three controller interfaces in the MusicWebApi project an AuthController can be found. This controller has only two methods, one that begins the authentication flow for the user and another that serves as a redirect URI that must resolve the code and return a *TokenResponse* object, which represents the standard OAuth 2.0 response, in the HTTP body.

3.4.4 Client

The client project allows for safer calls to a Music Web Api service by having methods that receive and return the exact model objects and build the appropriate HTTP requests. This makes applications that use these services cleaner and faster to develop, only needing to import the nugget package for this client and call the methods wanted.

3.5 Client Application

The initial plan for the client application was *"a mobile application, using the Android platform. The development environment will be .NET with the Xamarin SDK."*. The group decided to use the Xamarin Forms variation of the environment since it has two advantages when compared to Xamarin Native:

- The ability to share similar UI between two different platforms (in addition to application code);
- Easier maintenance since even more code is shared.

The group decided it was best to sacrifice application size (as this is the biggest con to Xamarin Forms) for better code organization and overall looks.

This decision also made it so there are now two client applications instead of just one. Right now it's possible to run the event organizer client application on both Android and UWP/Windows.

The platform supports the current operations:

- Events - Create events;
- Items/Expenses - Addition of Expenses to an event;
- Tasks - Add tasks;
- Events - Cancel an event;
- Register Users - Registering users using authentication providers (*Facebook*)
- Invites - Invite users to participate in an event;
- Add Payment - Associate payment with a certain item and see the current extract of the event

Implementation of the client app

Layout and basic application structure This application as stated earlier is implemented using Xamarin Forms. This toolkit serves as our view engine for the application and suggests ways of organizing the client application so code can be reused between platforms.

The application has Content Pages that are the Views of the app. These are written using *Extensible Application Markup Language* (XAML) which is a declarative markup language that allows to create the user interfaces for the application. These are written as files with the ".xaml" extension.
The XAML content pages have a .cs file associated with them that allow C# code to be written to them by subscribing to events that are exposed by the Xamarin Forms platform. This is simillar to ASP.NET's *Web Forms*.

For example, when navigating between views there is a concept of "Push" and "Pop". Push meaning going forward and Pop going backwards in the page navigation.

When one is "Pushing" to a new view that view needs to be instantiated with the *new* operator, executing the views ctor. The constructor shouldn't have code that takes time to complete in its body, only layout changes or event declaration should happen here, for example.

If one has to, for example, call the API when navigating to a new view one should subscribe to the *OnAppearing* method which is called when the view is rendered on the target device.

Local Notifications Local notifications are handled with the help of a Xamarin Plugin called *Xam.Plugins.Notifier*. This plugin allows for cross-platform local notification handling by exposing three methods to the programmer. The group decided to use this plugin because of it's simplicity and the easiness of allowing the same notifications working for all platforms, in this case, UWP and Android. However there are some limitations with this plugin:

- There is no way to get a list of the current notifications of the App;
- The latest (stable) version of the Nuget package present on the GitHub repo, as of 2019-05-18 is not working with scheduled notifications on UWP (meaning the group had to search the github repo of this plugin and on pull request #43 https://github.com/edsnider/localnotificationsplugin/pull/43 this was fixed).

The way the group handled the cancelling of scheduled notifications was to save the notifications according to an Id that could be easily retrieved when canceling, by giving the object that caused the notification to happen and an id:

These are the method signatures that handle this problem:

- public void AddNotificationCountByObjectAndId<T>(T obj, int id, int count) where T : class - for adding notifications

- public void CancelNotificationByObjectAndId<T>(T obj, int id) where T : class - for cancelling notifications

For example if one wanted to add two notifications for an Event one would pass the EventModel object and its Id. This would save a configuration on the device after scheduling the notifications and if you wanted to cancel all of the notifications you would call CancelNotificationByObjectAndId with the same parameters that were supplied to an earlier call to AddNotificationCountByObjectAndId.

Note that with this implementation it's not possible to cancel a specific notification. In case one wanted to cancel one it can be done by cancelling all notifications for a specific object and then creating new ones.

The handling of saving the configuration is done with the help of Xam.Plugins.Settings as used in other parts of this projects client application.

API Request Handling The application requests the groups API using a Request Service that is designed using *HttpClient* and a class that the group implemented called NetworkService. This NetworkService class in conjunction with *Polly* exposes a "Retry" method that allows for an arbitrary number of attempts of requesting the API in case a network communication fails, while allowing the group to execute a function on each attempt (for example, this can be used on a case where the user could be warned about the operation taking longer than usual). However the Request Service only uses the Network Service on GET methods because of the retry being a problem on non idempotent operations, which could cause inconsistencies to the application state.

Permission Handlers The maps and contact providers are implemented with permission handling in mind. Users might be prompted to grant permission to a functionality in the application, and the application should behave accordingly to the granting or denying of the permission. On Android permission handling is different from other platforms, as on the user decision it calls a method on the MainActivity class of the application, meaning that the programmer will lose context from the page that requested the permission for a specific functionality. To solve this the providers that require permissions have interfaces that expose callback setters, so immediately after the user grants or denies permission, there is behavior attached to its choice (for example, going back in the navigation stack when a user denies access to something). Callback setters are useful and flexible because they separate the concerns to the provider. For example, after the user grants permissions to the contact list, it should be provided on the screen. For this functionality you set a callback to be executed after the granting of the permission by the user. On UWP this isn't needed, as the permission checking is not based on calling a method on another class. The UWP API normally exposes methods that return the permission status on the fly and then ask the user if the application does not have permission. As a side note there is a Xamarin plugin for handling permissions called Plugin.Permissions. However, this plugin, by analyzing its source code, is somewhat heavy on the performance side, as to handle the permission architecture of android there is a need to use locking mechanisms to synchronize the permission request and its subsequent status.

Location/Maps The definition of the location of the event is supported by a plugin by the name of Xamarin.Forms.Maps. This plugin allows for the cross-platform integration of maps in a mobile application, providing a simple to use interface that is common to every platform. The plugin is cross-platform, however, for each platform there is a different map provider:

- Android – Google Maps map provider
- UWP – Bing map provider
- iOS – Apple maps map provider

However, when using a map provider, the application needs to ask the user for permission (mainly for location). This was handled by creating a wrapper around each provider and supplying instances to the cross-platform project of the application via dependency injection. These wrappers handle permission handling as well as every map operation possible that the application needs to support (moving the map to a specific location, showing a dialog when the user doesn't give permission for map usage, adding pins on the map, etc…). The group implemented the map feature for the Android and UWP platforms. As seen in the contacts provider, handling permissions on Android is particularly hard compared to other platforms. The Android runtime calls a specific method in the main activity to handle permissions, hence the need to pass page components to the maps provider (to call methods and implement functionality after the user grants permission.)

If the request for permission is denied, then the user is returned to the previous page. However, on the UWP application the settings page for permissions of the app is opened automatically, so the user can grant permission for map usage.

The search component of the map functionality of the application is provided by the OpenStreetMap API. The group decided to use this API as it is free, easy to use (there's not even a need for requesting an API key). The only restrictions there are for this API is that you shouldn't abuse requesting it and it is obligatory to identify your application by providing a custom user-agent in the HTTP headers. The user searches an address, the results are presented, and after the user chooses one from the list, the coordinates are passed into the map provider (giving the opportunity for the user to save the event location).

Contacts When one is inviting a user to an event a list of contacts present in the invitee's device is shown. The list of contacts is brought with the help of contact providers and a Xamarin plugin called Xamarin.Forms.Contacts. The Xamarin plugin is straightforward to use. This plugin provides a service and there is only need to call a method on that service. However this plugin does not handle permissions and it does not work on UWP. So the group implemented a "hybrid" solution with contact providers. Contact providers offer an interface to implement contact functionality on multiple platforms. This contact provider allows one to supply the list view that is going to hold the contacts to an instance of itself. This is especially useful for handling Android permissions.

3.5.1 Database

The application and logging databases are being supported by PostgreSQL. The group choose this engine over others because this one is free and open-source, it also supports a wide range of operating systems and is usually faster to install and setup.

Figure 3.13 illustrates the data model.

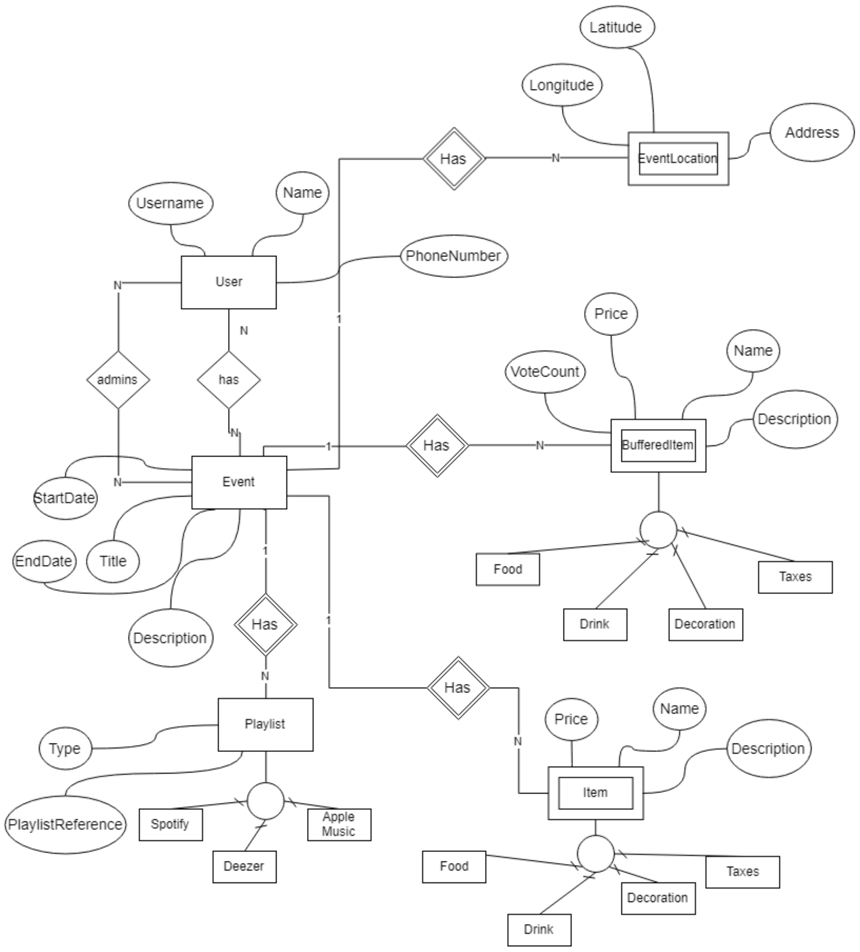


Figure 3.13: Data Model

- Event - Represents the Events, has a Title, Description and Start/End dates of the event;
- Item - Represents the Items (or expenses) of the Event. Has a Name, Description, Price, State and it can have various types;
- Task - Represents the Tasks of the Event assigned to participants. Has a Name, Description, Expiration Date, State and references an event and a participant;

- BufferedItem - It is the same as the Items except it represents only the items that have been successfully approved by all members of the event (note the VoteCount field, it is incremented by 1 every time a user votes). When the VoteCount field value is the same as the number of users in an Event, the BufferedItem gets moved to the Item table;
- Playlist - Represents a Playlist, has a reference (URL to the playlist itself) and a type (this type describes the service that provides the playlist. Example: *Spotify*);
- User - Describes a User, has a Name, Username and a Phone Number;
- EventLocation - Represent the Event location. Has an address, longitude and latitude.

The logging database is apart of the applicational one. This is because not only might the applicational database be in a different location from the logging one but the logs can grow exponentionally, affecting storage and performance of the applicational one if the log table was in the same database.

The logging database only hosts one table on the default schema. The table is called EventLog and it represents logs. It is comprised of an EventId (type of log), LogLevel, Date, Message and StackTrace (these last two are only present in case of an exception logging):

Figure 3.14 illustrates the logging database model.

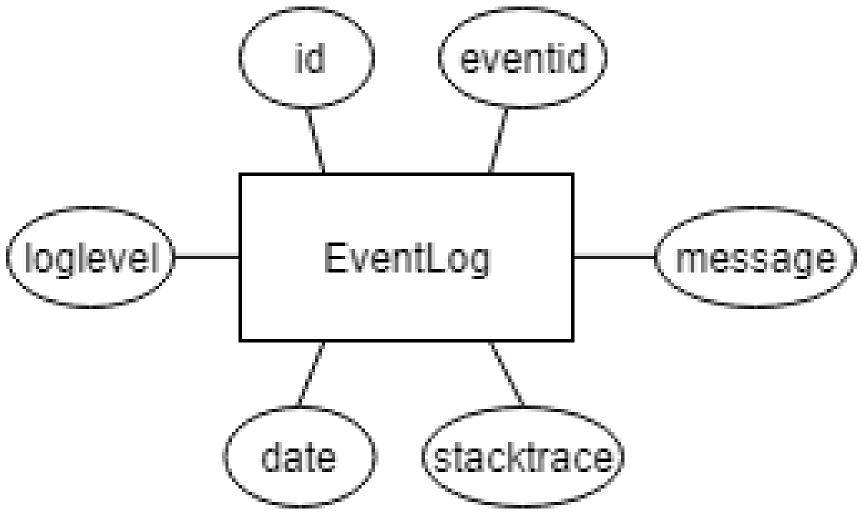


Figure 3.14: Data Model

Note: All of these tables have a unique id that is an autoincremented/serial upon insertion integer.

Chapter 4

Development support

Contents

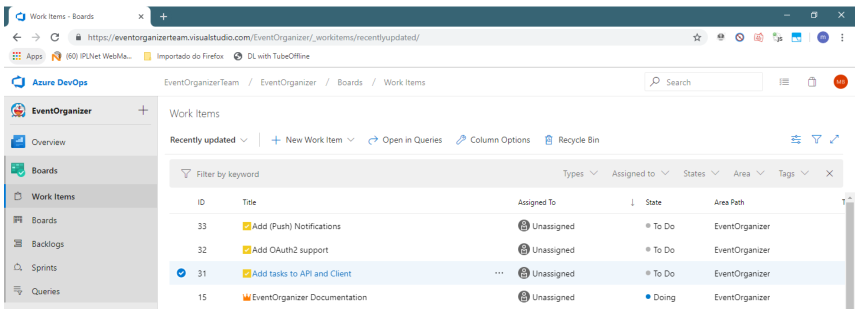
4.1 Azure DevOps Services

To ease development the group decided to use *Microsoft (2019)*. This platform provides development collaboration tools along with mechanisms that allow for a better work environment that follow DevOps practices.

Figure 4.1 illustrates DevOps Work Items (Azure Boards).

From the many tools provided by the service, the following are the ones used in this project:

- Azure Boards
- Azure Repos
- Azure Pipelines
- Azure Artifacts



ID	Title	Assigned To	State	Area Path
33	Add (Push) Notifications	Unassigned	To Do	EventOrganizer
32	Add OAuth2 support	Unassigned	To Do	EventOrganizer
31	Add tasks to API and Client	Unassigned	To Do	EventOrganizer
15	EventOrganizer Documentation	Unassigned	Doing	EventOrganizer

Figure 4.1: DevOps Boards Work Items

4.1.1 Azure Repos

Azure Repos is a set of version control tools, allowing for multiple repositories, each one having its own version control system. For this project four repositories where created:

- EventOrganizer.WebApi – Where the source code for the EventAPI and its HTTP Client is maintained.
- EventOrganizer.Client – Where the Xamarin client application source code is maintained.
- EventOrganizer.Database – Repository containing the scripts and backups for database management.

- EventOrganizer.Documentation – Dedicated to gather project documentation used in development.

- EventOrganizer.MusicApi – Contains the source code for the Music Web Api and its HTTP Client.

Figure 4.2 illustrates the feature branching strategy used.

Wanting to keep the development repositories, EventOrganizer.WebApi, EventOrganizer.MusicApi and EventOrganizer.Client organized and easy to maintain, the group decided to use the following branching strategy:

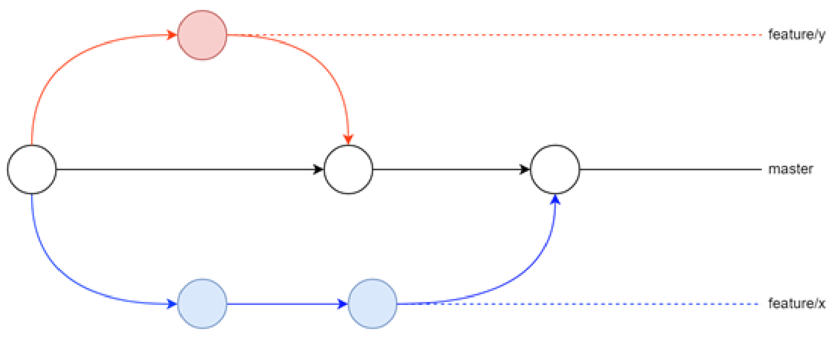


Figure 4.2: Feature Branching Strategy

This branching strategy consists of creating a feature branch for each new functionality that needs to be implemented. This, combined with frequent pull requests, helped the team maintaining the source code, making sure the master branch always had a stable version and making bugs less frequent and smaller problems to resolve.

4.1.2 Azure Pipelines

Azure Pipelines is a service that can be used to build, test and deploy projects by having remote machines run a preconfigured pipeline.

For this project the group configured three pipelines:

- Event Web API Pipeline – Builds the EventAPI project and deploys the Web API to Azure App Services.

- Generate Client Nugets – Builds and packages the EventAPI HTTP client libraries into nuget packages that contain versioning.

- Event Web API Pipeline – Builds the EventAPI project and deploys the Web API to Azure App Services.

The first two pipelines are queued for execution every time there is a new stable version, meaning every time there is a push to the master branch.

4.1.3 Azure Artifacts

Azure Artifacts gave the team the possibility of having a private nuget feed where the nuget packages generated in the Azure Pipelines are automatically pushed. This service comes with a page in Azure DevOps, where the nugets can also be viewed and managed.

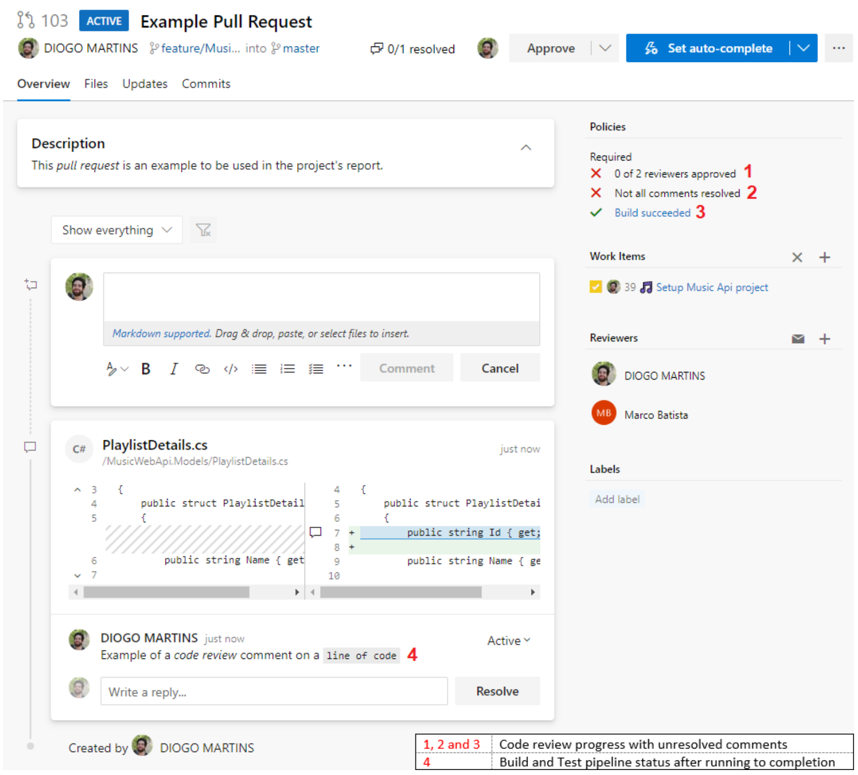
4.1.4 Pull Requests

To keep code versioning efficient and organized, the decision to block direct pushes to the master branch was made.

When a feature is complete, instead of merging the feature branch to the master branch, a pull request must be open and reviewed before being completed, which results in an automatic merge to the master branch.

In addition to the mandatory review of the code, pull requests must also pass a build pipeline configured for each repository. This was done by first creating a new Azure Pipeline for each repository that builds and tests the respective solutions and secondly adding a build policy for the master branch, which specifies that the previously created pipeline should run to completion successfully.

The page for a pull request where the required policies include the previously mentioned code review, build and test run can be seen in the screen capture below.



Example Pull Request

DIOGO MARTINS feature/Musi... into master

Description

This *pull request* is an example to be used in the project's report.

Markdown supported. Drag & drop, paste, or select files to insert.

PlaylistDetails.cs

/MusicWebApi.Models/PlaylistDetails.cs

just now

```
{
    public struct PlaylistDetail
    {

        public string Name { get
```

```
{
    public struct PlaylistDetai
    {
+       public string Id { get;
+
        public string Name { ge
```

DIOGO MARTINS just now

Example of a *code review* comment on a `line of code` 4

Write a reply...

Created by DIOGO MARTINS

Policies

Required

✗ 0 of 2 reviewers approved 1

✗ Not all comments resolved 2

✓ Build succeeded 3

Work Items

39 Setup Music Api project

Reviewers

DIOGO MARTINS

MB Marco Batista

Labels

1, 2 and 3	Code review progress with unresolved comments
4	Build and Test pipeline status after running to completion

Figure 4.3: An example of a pull request in the development repositories

4.2 Other tools

The group is also using Microsoft Visual Studio as it's main programming IDE as it's really flexible and it's integration with Nuget, Xamarin and UWP is seamless to the group and offers (almost) no problems and Jetbrains DataGrip as the main database IDE (although sometimes the group also uses pgAdmin for more low level operations on the PostgreSQL database).

4.3 Architecture

Figure 4.4 illustrates the system architecture.

4.3.1 Client

The client component of this project is implemented as a mobile application, using the *Android* and *Windows* platform.

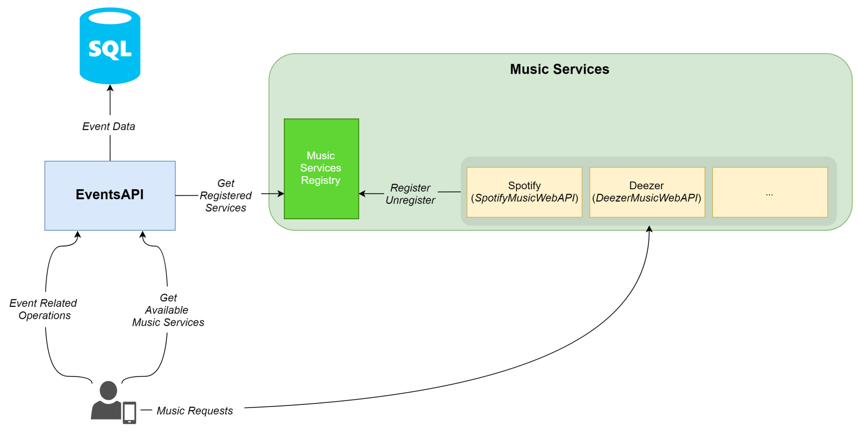


Figure 4.4: System architecture.

4.3.2 Server

The server component is implemented as a REST API, using .NET Core MVC.

- Event API - Supports the event creation and all of its operations (adding, editing and removing guests, expenses, tasks, among others);
- Music API - Supports the *playlist* creation;
- Payments API (optional) - Supports the payment of the event's expenses.

The database operations are supported by the PostgreSQL database engine.

Observations

The motivation behind choosing this server-client architecture is the possibility of eventually creating other clients, without having to change the servers that support it.

The reason why the Payments and Media API aren't unified into the Event API is for the sake of having the benefit of the client only communicating with these APIs, yet having the possibility of comunicating with a wide range of services without having to implement them specifically.

The motivation behind the usage of SQL instead of NoSQL database types is because the entities used in this application are of relational types. Also, the atomicty and transactional behaviour that SQL databases support is of importance for consistency across event information.

Chapter 5

Roadmap

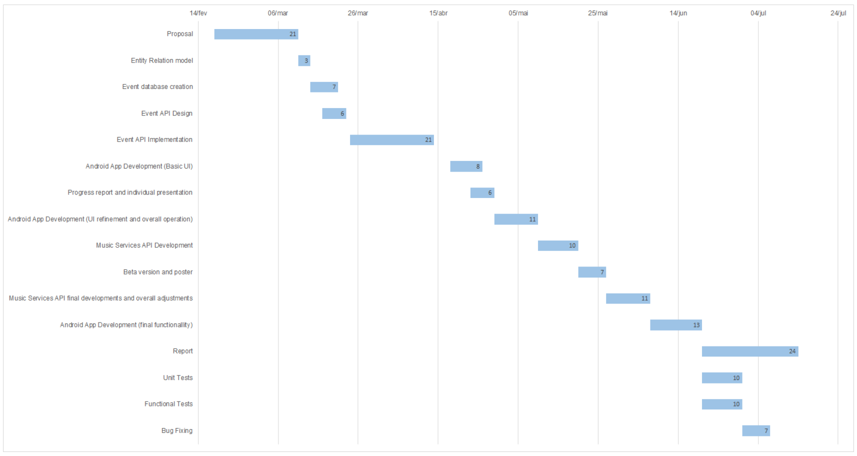


Figure 5.1: Gantt diagram depicting the project plan.

Chapter 6

Conclusions and Future Work

Contents

6.1 About Xamarin

While Xamarin is a great toolkit to develop cross platform applications that reuse a lot of the code and view definitions (especially Xamarin Forms) between devices the group realized that there is some problems with the implementation of the technology and the documentation available for it.
For example, upon using WebView there is a bug on the Navigation mechanism of the page that sometimes occurs in an unhandled null reference exception while trying to "Pop" to an earlier calling Content Page, this has been experienced by many users on the UWP and Android implementations of this type of Page.
Also the group had some troubles trying to test the authentication mechanism on the client application because as far as the group's research went there is no way to avoid having a valid certificate when handling OAuth2 requests on Xamarin (not even local trusted certificates). So the solution was deploying the app on Azure since this hosting provides a valid certificate.

6.2 About .NET Core

.NET Core is a great free and open-source framework that allows cross-platform software development. The documentation is solid and the group felt at home coding the API using this technology coming from a .NET Framework coding background.

6.3 About PostgreSQL

PostgreSQL is a great SQL database engine that is really similar in usage to Microsoft's SQL Server but it is completely free and open source and it has a huge community supporting it. However the group had to host their own PostgreSQL server on a personal computer until it found out that ElephantSQL (2019) provides a free service of a PostgreSQL instance with 20Mb of space (which is enough for developments on this project).

6.4 Future Developments

After making the current API and Client Application more stable the group needs to get a better, more solid hosting mechanism (as of now it is a mix between Azure and the groups personal computers). The next development so the project is complete is the voting system and the Music API(the platform to unify music services to played at events). Development is currently approximately 70% done.

Appendices

Appendix A

Other Definitions

A.1 Technologies

- Xamarin Forms - Cross-platform UI development toolkit for .NET;
- .NET Core - Free and open-source framework;
- ADO.NET - Data access technology included in .NET Framework/CORE;
- Entity Framework - ORM framework for ADO.NET;
- Azure - Cloud computing service by Microsoft;

References

Asana (2019). Use a asana para gerenciar trabalhos, projetos e tarefas da sua equipe on-line. URL: https://asana.com/pt.

Cvent (2019). Event management software & hospitality solutions. URL: https://www.cvent.com/.

ElephantSQL (2019). Elephantsql - postgresql as a service. URL: https://www.elephantsql.com/.

Eventbrite (2019). Eventbrite. URL: https://www.eventbrite.com/.

Events, X. (2019). Platform for attendee generation and management | xing events. URL: https://www.xing-events.com/en/.

Facebook (2019). Facebook. URL: https://www.facebook.com/.

Microsoft (2019). Azure devops services | microsoft azure. URL: https://azure.microsoft.com/en-in/services/devops/.

Planner, S. (2019). Super planner - event planning on the app store. URL: https://itunes.apple.com/us/app/super-planner-event-planning/id383727111?mt=8.