IPL escola superior de tecnologia e gestão instituto politécnico de leiria

Applications for the Internet

Computer Engineering	2nd Year	2nd Semester	2017-18	Periodic Evaluation
Project	Deadline for Results Disclosure: 22 June 2018			
Date: 19 April 2018	**Delivery Date: 9 June 2018**			

Project – Personal Finances Assistant

OBJECTIVE

The objective of this project is to implement a Web Application, using Laravel Framework, to help users to manage their personal finances.

SCENARY

The "Personal Finances Assistant" Web Application (or Platform) main purpose is to help users to manage their personal finances. Anyone can create a user account on the application, and each user with an account will have a public profile (with its name and photo) and a private area where he can register all its personal financial movements, organize personal accounts (financial accounts), view a summary of their financial status and obtain statistical information about its revenues and expenses.

Also, each user may specify a group of associate members - other registered users on the platform that will have access to all its personal financial information. This will be a read-only access, where the associate members can view all the information, but cannot change, remove or add any financial data (account or movement). Note that if user "A" defines user "B" as an associate member, does not automatically means that user "A" is also an associate member of user "B".

In addition to the platform users (registered users), the application supports 2 additional types of users: anonymous users and administrators. Anonymous users (users that have not authenticated) may access generic information about the application or platform (static content that describes the application/platform, "About us", etc.) and create (register) a new user.

Administrators are responsible for user administration, where they may block or reactivate user accounts and change its type ("administrator" or "normal"). The administrator cannot create or remove any account, nor block, reactivate or change the type of its own account.

Administrators can also use the platform as any other registered user, which means that he also has a public profile and can use all the financial features available to a normal user. For all non-administrator features, the administrator has no special privileges. For instance, he cannot view any financial information of other users, except if they include them as an associate member.

PERSONAL FINANCE

The personal finance of each user is managed essentially by a set of financial accounts and movements. There can be several types of accounts (*defined by the application database seed*), such as a "bank account", "pocket money", "PayPal account", "credit card"; "meal card", etc.

All accounts have an owner (titular), an account code (a string that is unique for each user), the date when the account was created, a description, the account starting balance (saldo inicial) and current balance (saldo actual). Account balances (starting or current) can be positive or negative (it means that the user owes money). An account can be deleted (removed) if it has no movements or closed otherwise. Closed accounts will be implemented as "*soft deletes*" and may be reopened ("*restored*"), but as long as they remain closed, they should be invisible to the user (except to reopen them) – they don't appear on account lists and their financial data is ignored from any summary or statistic information.

Each user might have several accounts and each account may have several financial movements. There are 2 types of financial movements: expenses (despesas) and revenues (receitas). An expense movement will decrease the current account balance and a revenue movement will increase it. In addition to the type (expense/revenue), movements are also classified according to categories (*defined by the application database seed*). Examples of expenses movements: food; clothes; services; electricity; phone; fuel; mortgage payment; etc. Examples of revenue movements: salary; bonus; royalties; interests; gifts; dividends; product sales; etc.

All movements will belong to an account and have a type (expense/revenue), a category, a description, the value (always positive – increases or decreases account balance according to the movement type), the date (only date – no time information) of the movement, the start account balance and end account balance. The start and end account balance will be calculated according to the end account balance of the previous movement. All movement balances (start and end) and current account balances (of the account) must be consistent and calculated automatically.

Table 1: Example of movements

Account . . .			**Current Balance:**		**674,50 €**
Account Movements:					
Type	**Category**	**Date**	**Value**	**Start Balance**	**End Balance**
Revenue	Gift	14-2-2018	**50,00 €**	624,00 €	674,50 €
Expense	Clothes	13-2-2018	**230,50 €**	855,00 €	624,50 €
Expense	Food	13-2-2018	**145,00 €**	1.000,00 €	855,00 €
Revenue	Salary	12-2-2018	**1.000,00 €**	0,00 €	1.000,00 €

When a new movement is added as the last movement of the account (newer date), only the new movement balances and the current account balance will be calculated and updated. If the new movement is added with a date prior to the last movement of the account, all balances (start and end) of all movements from that date on will have to be recalculated, as well as the current account balance. The same applies when the user updates or deletes a movement, so that all movement and account balances are consistent and calculated automatically.

If two movements have the same date, the order of insertion (created_*at* timestamp) is used to define the order of the movement - the last movement inserted will be considered the newer movement.

Each movement can also be associated to a document, such as an invoice, a receipt, or other, related to itself. The platform must allow users to upload a digital copy of the document (pdf, png or jpeg only) and associate it with a specific movement. Later on, users might download or view a copy of those documents or delete them from the application. Only the owner, or an associate member can view (download) a document.

The financial information collected from user's accounts (ignoring closed accounts) and related movements, will be used by the application to create a summary of user's financial status, as well as statistical information about its revenues and expenses. Summary must include the total balance of all accounts (user's grand total) and the summary information about each account including the relative weight (percentage) of each account over the total balance. The statistical information must include the total expenses or revenues by category for a given time frame, and the evolution through time (monthly) of expenses and revenues (by category), account balances and total balance.

Each user has its own summary and statistical information, which can only be accessed by him and by its associate members. Summary and statistical information should be presented both as textual and graphical data.

FUNCTIONAL REQUIREMENTS

Project's functional requirements will be presented as user stories, organized by types of users.

ALL USERS

US 1. As a user (anonymous or authenticated user) I want to access the site's initial page with information about the total of registered users, total number of accounts and movements registered on the platform;

ANONYMOUS USERS

US 2. As an anonymous user I want to register as a new user of the application. Registration data should include user's name (only spaces and letters), e-mail (must be unique), password (3 or more characters), phone number (optional) and a photo (optional - by uploading an image);

US 3. As an anonymous user I want to authenticate the application with valid credentials (email and password);

US 4. As an anonymous user I want to reset a forgotten password, by passing an e-mail address to the application, and if the e-mail is associated to a valid user account, receiving an e-mail with a link to reset the account password.

ADMINISTRATION USERS

US 5. As an administrator I want to view a list of registered users (including blocked users and administrators) of the application. For each user show at least the name, email, type and status (blocked or not) fields;

US 6. As an administrator I want to be able to filter users by name (partial match), type (normal or administrator) and blocked status (whether it is blocked or not);

US 7. As an administrator I want to be able to block or reactivate users or to change its type to normal or administrator. These changes can be applied to any user, except myself.

NORMAL USERS

The following user stories are relative to all registered users, including administrators, as they have all the privileges of a normal registered user.

US 8. As a user I want to be able to logout from the application. After logging out, I should be redirect to the site's initial page;

US 9. As a user I should be able to change my password;

US 10. As a user I should be able to update my profile, specifically: my name, e-mail, phone number and photo (by uploading an image). When updating the e-mail, the application must guarantee that the new e-mail is unique among all users (including blocked users);

US 11. As a user I want to view and filter (by name) the public profile of all registered users, including administrators and blocked users. Each public profile includes only the name and photo (if any) of the user. Users that belong to my group of associate members should be identifiable as such, as well as users whose groups of associate members I belong to;

US 12. As a user I want to view the list of users that belong to my group of associate members. The list should show at least the name and email of the member;

US 13. As a user I want to view the list of users whose groups of associate members I belong to. The list should show at least the name and email of the member and a link to the accounts page of the member (see US 14);

US 14. As a user I want to view the list of my opened accounts and the list of my closed accounts;

US 15. As a user I want to delete (if it has no movements) or close an opened account;

US 16. As a user I want to reopen a closed account;

US 17. As a user I want to create accounts. Account data should include the account type, the date when the account was created (not necessarily the current date), the account code and the start balance of the account. It might also include a description;

US 18. As a user I want to edit the data of accounts, namely its type, code (always unique for each user), description and start balance.

US 19. As a user, when I change the start balance of an account, I want the application to recalculate all start and end balances of all movements of the account, as well as the current balance of that account;

US 20. As a user I want to view all movements of an account. The listing must show at least the following fields: category, date, value, type and end_balance;

US 21. As a user I want to add, edit or delete movements of an account. Movement data should include the type (revenue or expense), category, date (no time information is required), value and a description (optional). It might also include a link to the associated document (US24). Note that start and end balance are to be calculated automatically – user should not edit these values;

US 22. As a user, when I add, edit (the type, value or date) or delete movements, I want the application to recalculate the account current balance, as well as the start and end balance of all affected movements (according to the date of the added, edited or deleted movement). For performance optimization purposes, students should guarantee that recalculation is only applied to affected movements;

US 23. As a user I want to add (by uploading) documents (pdf, png or jpeg) and associate each one with a specific movement;

US 24. As a user I want to remove documents and dissociate them from the related movement;

US 25. As a user I want to view/download documents from the associated movements. Only I and users that belong to my group of associate members can view/download these documents;

US 26. As a user I want to view a summary of my financial situation, namely the total balance of all accounts (user's grand total) and the summary information about each account including the relative weight (percentage) of each account over the total balance;

US 27. As a user I want to view (with textual and graphical data) the statistical information about the total revenues and expenses by categories, on a given time frame;

US 28. As a user I want to view (with textual and graphical data) the statistical information about the evolution through time (monthly) of the revenues and expenses by categories, on a given time frame;

US 29. As a user I want to add any user to my group of associate members, which will give him permission to view all my financial data;

US 30. As a user I want to remove any user from my group of associate members, which will inhibit him from accessing my financial data;

US 31. As a user I want to access all financial information and view/download documents of any user whose group of associate members I belong to – the fact that a user belongs to my group of associate members does not grant me authorization to access its financial data;

US 32. As a user, when accessing someone else's financial data, and documents, I shall not be allowed to change any data - I cannot add, edit or delete anything that belongs to the other user;

US 33. As a user, I would like to have some private customization - to be defined by the students - such as preferred accounts or categories of movements, changing the order of accounts, summary information or statistics, etc.

TESTING

Tests are provided to validate some of the user stories. You should replace Laravel's "tests" folder with the contents of the "tests.zip" found on Moodle. The provided tests cover most of the user's stories and will be used to assess each user story. To be able to run the tests you should follow the guidelines regarding the routes and form fields of the table below. Besides the required routes you are free to add additional routes to fulfill your layout requirements.

User Story	**Route**	**Verb**	**Additional notes**
US.1	/	GET	The page should render at least the information mentioned on the user story
US.2	/register /register	GET POST	Default Laravel that renders the register form Default Laravel route for registering a new user. The form MUST have at least the following fields **email**, **password**, **password_confirmation, name**, **phone** and **profile_photo**.
US.3	/login	POST	Default Laravel route to authenticate a user. The form MUST have at least the following fields: **email** and **password**.
US.5	/users	GET	
US.6	/users	GET	Filters are specified on the query string and should have the following names: **name** (pattern to search)**, type** (admin | normal) and **status** (blocked | unblocked). The filter is only applied if a valid value is provided. Example: `/users?name=joh&type=admin`

US.7	/users/{user}/block /users/{user}/unblock /users/{user}/promote /users/{user}/demote	PATCH PATCH PATCH PATCH	The specified user is blocked The specified user is unblocked The specified user is promoted to type admin The specified user is demoted to type normal
US.9	/me/password	PATCH	The form MUST have at least the following fields: **old_password**, **password** and **password_confirmation**.
US.10	/me/profile	PUT	The form MUST have at least the following fields **email**, **name**, **phone** and **profile_photo**.
US.11	/profiles	GET	Users that belong to my group of associate members are tagged with the word "associate". Users whose groups of associate members I belong to are tagged with the word "associate-of". The filter is specified on the query string and should have the following name: **name** (pattern to search). The filter is only applied if a valid value is provided. Example for filtering all users whose name contains the word "joh": `/profiles?name=joh`
US.12	/me/associates	GET	
US.13	/me/associate-of	GET	
US.14	/accounts/{user} /accounts/{user}/opened /accounts/{user}/closed	GET GET GET	Show all accounts for the specified user Show only opened accounts for the specified user Show only closed accounts for the specified user
US.15	/account/{account} /account/{account}/close	DELETE PATCH	Deletes the specified account Closes the specified account
US.16	/account/{account}/open	PATCH	Reopens the specified closed account

US.17	/account	POST	The form MUST have at least the following fields **type**, **code**, **date**, **start_balance** and **description**.
US.18	/account/{account}	PUT	The form MUST have at least the following fields **type**, **code**, **start_balance** and **description**.
US.20	/movements/{account}	GET	
US.21	/movements/{account}/create	GET	Renders the form for adding a new movement
	/movements/{account}/create	POST	Creates a new movement
	/movements/{account}/{movement}	GET	Renders the form to edit the specified movement
	/movements/{account}/{movement}	PUT	Updates the specified movement
	/movements/{account}/{movement}	DELETE	Deletes the specified movement For the POST/PUT routes, the form MUST have at least the following fields **type**, **category**, **date**, **value**, **description**, **document_file** and **document_description**.
US.23	/documents/{movement}	POST	The form MUST have at least the following fields: **document_file** and **document_description**.
US.24	/documents/{movement}/{document}	DELETE	
US.25	/documents/{movement}/{document}	GET	
US.26	/me/dashboard	GET	You can have additional routes to show more statistics. This route should present the data specified on the User Story.
US.29	/me/associates	POST	The form MUST have at least the field **associated_user**.
US.30	/me/associates/{user}	DELETE	

Steps for running tests:

1. Replace the tests folder with the folder provided by the teachers;
2. Move the phpunit.xml to the root folder of the project;
3. Go the project root folder (from homestead ssh shell) and type:
   `composer require doctrine/dbal`
4. Open .env file and check if APP_URL is set to your project's domain

5. To run all the tests, go the project root folder (from homestead ssh shell) and type:
   `vendor/bin/phpunit`
6. To run a specific test, e.g., US.3, type:
   `vendor/bin/phpunit --filter UserStory3Test`
7. To run tests in Dox format, add the --testdox flag. Examples:
   `vendor/bin/phpunit --testdox`
   `vendor/bin/phpunit --textdox  --filter UserStory3Test`

NON-FUNCTIONAL REQUIREMENTS

Project's non-functional requirements are the following:

NFR 1. Application must use the Laravel Framework 5.6;

NFR 2. Application must use a MySQL database whose structure is provided and cannot be changed;

NFR 3. Application structure, architecture and code must follow Laravel recommended conventions and practices, as well as PSR-2 guidelines;

NFR 4. When displaying data as a list or table, the total number of potential records should be considered. This implies that for all but very small datasets, lists and tables should implement pagination and sortable columns. Also, when possible filtering the dataset should also be considered;

NFR 5. Form's data should always be validated on the server, according to rules extrapolated from the scenery, functional requirements and database structure;

NFR 6. Visual appearance and layout should be consistent through the entire application, and adapted to the applications' objective and usage context;

NFR 7. Application usability should be a major concern – application should be as usable and as simple as possible;

NFR 8. Application should be as reliable as possible;

NFR 9. Application should have the best performance possible;

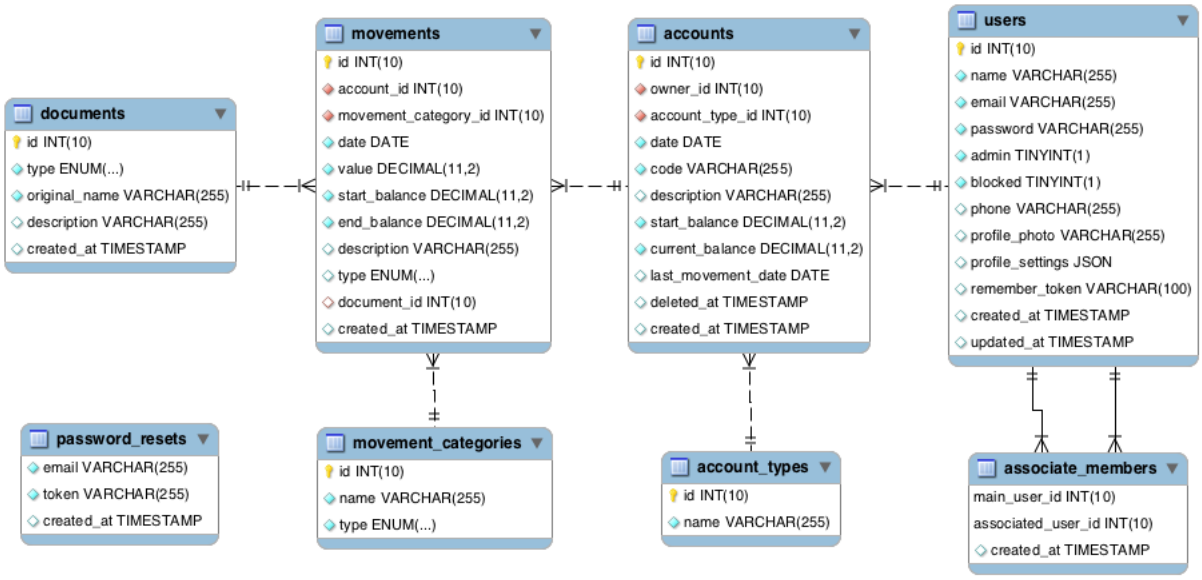
NFR 10. Application should be as secure as possible;

NFR 11. Application should guarantee the protection of users' data and privacy, ensuring that no unauthorized user can view or change any data or document it should not have access to;

NFR 12. The application should follow the guidelines of the "Testing".

DATABASE STRUCTURE

The provided database structure (implemented with migrations) is as following:



The database structure includes the tables:

- **password_resets** – table used by Laravel to implement the reset password mechanism;
- **users** – application's users' information. Includes all the columns required by Laravel authentication system;
- **associate_member** – the relation that represent all users that belong to a group of associate members of the user represented by the "main_user_id";
- **account_types** – the types of accounts (seed will be provided). Application will only use the table's data – it does not need to be editable;
- **movement_categories** – the categories of the movements (seed will be provided). Application will only use the table's data – it does not need to be editable;
- **accounts** – table with all financial accounts (including closed accounts);
- **movements** – table with all financial movements;
- **documents** – table with all documents associated to movements;

To create the database structure and populate it, replace Laravel's "database" folder with the contents of the "database.zip" found on Moodle. Then execute the following commands on the web server shell:

```
>  cd <root/folder/of/Laravel/project>
>  php artisan migrate
>  composer dump-autoload
>  php artisan db:seed
```

DELIVERY

The delivery of the project includes 2 files (both are mandatory):

- Excel file with the **report** – the report will include group elements identification and information about the project implementation. The model for the report will be provided by the teacher;
- Zip file with all project's **code** – this file should include a copy of all the project's folders and files, except the folder "vendor".

EVALUATION

The evaluation (avaliação) of the project will use the following criteria:

Number	Weight	Criteria
1	**80%**	Functional Requirements
2	**20%**	Non-Functional Requirements

- The functional requirements will be evaluated by considering the implementation of all user stories specified.
    - Each of these user stories: **11, 19, 21, 22, 23, 25, 28** will weight 1/20 of the grade for this criterion, or **4%** of the project total grade.
    - Each of the remaining 26 user stories will weight 1/40 of the of the grade for this criterion, or **2%** of the project total grade.

  Each user story grade will depend on the quality, usability, reliability and performance of the implementation. A mix of automatic and manual tests, informal usability tests and code analysis will be used to assess each user story.
- The non-functional requirements will be evaluated globally for the entire project. Some of them will be analyzed by viewing the structure and code of the project; others by analyzing the application operation and behavior and others by random tests (automatic or manual) executed on the application (for example, to check if privacy is maintained or if the application is secure).