
Implementation Document

for

CampusPay

Version 1.0

Prepared by

Group 2:

Aditya Bangar	210069
Akshat Rajani	210812
Harsh Bihany	210406
Kalika	210482
Monil Lodha	210630
Pratham Sahu	210755
Pulkit Gopalani	180564
Ravija Chandel	210835
Shantanu Kolte	210958
Siddhant Jakhotiya	211030

Group Name: TheMissingSemicolon

adityavb21@iitk.ac.in
rajanias21@iitk.ac.in
harshb21@iitk.ac.in
kalika21@iitk.ac.in
monill21@iitk.ac.in
spratham21@iitk.ac.in
gpulkit@iitk.ac.in
ravijac21@iitk.ac.in
shantanu21@iitk.ac.in
siddhantj21@iitk.ac.in

Course:

CS253

Mentor TA:

Sumit Lahiri

Date:

Mar 17, 2023

Contents

CONTENTS	I
REVISIONS	II
1 IMPLEMENTATION DETAILS	1
2 CODEBASE	2
3 COMPLETENESS	3
APPENDIX A - GROUP LOG	4

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Akshat Rajani Monil Lodha Pulkit Gopalani Siddhant Jakhotiya	Primary version of the document. The document has details about all the features that have been implemented as of 20/03/2023	20/03/2023

1 Implementation Details

Provide the details of programming languages, frameworks, libraries, database systems, build systems, etc. that you have used for implementing your software.

Provide a brief justification of choosing any tool by stating its benefits over the alternatives.

JavaScript is a widely employed programming language for the front-end development of web applications. It is compatible with all the major web browsers, including Chrome, Firefox, Safari, and Internet Explorer. JavaScript enables the creation of dynamic and interactive web pages with relative ease. However, we opted for TypeScript instead of JavaScript for its added benefit of static typing. This feature helps to prevent common programming errors during the implementation process, thus rendering the code more reliable. TypeScript has a vast and active community that facilitates error resolution through online developer community platforms. Consequently, we have implemented the front-end of our software entirely using TypeScript.

React is the framework employed for building the software. Its use of JSX enabled us to combine HTML code with TypeScript seamlessly. The folder structure provided by the React app gave us a well-organized starting point to build our application. The Virtual DOM of React enabled efficient updating of the user interface. The component-based structure of React allowed us to create reusable UI components, which we could import from libraries like Joy UI and Chakra UI. These libraries provide pre-built components with flexibility for custom styling, which resulted in a uniform design across the software and decreased the need for custom styling. The components were also easy to integrate into our scripts with required functionality, thus saving time during the development process.

For the back-end of the software, we used Python, using the Django framework. Python's intuitive and readable syntax coupled with the built-in features of Django, such as the admin interface and authentication, facilitated rapid development. The extensive and active community of Django developers provided support during the development process. Since our application involves financial transactions, the built-in security features of Django were a significant advantage. We opted for Django over other popular frameworks like Flask and Node.js due to its robust security features. Django also comes with the SQLite database management system. SQLite is an ACID-compliant RDBMS that is simple and easy to use. Therefore, it was a suitable choice for our application that facilitates transactions.

The following are the links where different api calls are sent to the backend to fetch/update data:

backend / manager / api / urls.py

bihany-harsh apis for overview page detail 9dda0aa · yesterday History

Code Blame 24 lines (23 loc) · 1.85 KB

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path("users/", views.CustomUserList.as_view(), name="users"),
6     path("users/customers/", views.CustomerList.as_view(), name="customers"),
7     path("users/vendors/", views.VendorList.as_view(), name="vendors"),
8     path("users/<str:user_id>/", views.CustomUserDetail.as_view(), name="user"),
9     path("users/<str:user_id>/transactions/", views.UserTransactionList.as_view(), name="user_transactions"),
10    path("users/<str:user_id>/navbar/", views.OverviewNavbar.as_view(), name="navbar-details"),
11    path("users/<str:user_id>/overview/", views.OverviewTable.as_view(), name="overview-details"),
12    path("users/<str:user_id>/transactions/make/", views.UserMakeTransaction.as_view(), name="user_transactions_post"),
13    path("users/<str:user_id>/clear_dues/", views.ClearDues.as_view(), name="clear_dues"),
14    path("users/<str:user_id>/clear_vendor_dues/", views.ClearDuesVendor.as_view(), name="clear_dues_vendor"),
15    path("users/<str:user_id>/vendors/", views.CustomerVendorList.as_view(), name="customer_vendor"),
16    path("users/<str:user_id>/customers/", views.VendorCustomerList.as_view(), name="vendor_customer"),
17    path("users/<str:user_id>/pending_dues/", views.PendingDuesList.as_view(), name="pending_dues"),
18    path("users/<str:user_id>/pending_dues_vendor/", views.PendingDuesVendor.as_view(), name="pending_due_for_vendor"),
19    path("users/<str:user_id>/notifications/", views.UserNotificationList.as_view(), name="notification"),
20    path("users/<str:user_id>/add_balance/", views.UserAddBalance.as_view(), name="add_balance"),
21    path("transactions/", views.TransactionList.as_view(), name="transactions"),
22    path("transactions/<str:transaction_id>/", views.TransactionDetail.as_view(), name="transaction"),
23    path("notifications/", views.NotificationList.as_view(), name="notifications"),
24 ]
```


backend / manager / auth_system / urls.py

bihany-harsh update profile auth 1ab9d86 · 2 days ago History

Code Blame 12 lines (11 loc) · 599 Bytes


```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path("register/", views.UserRegister.as_view(), name="register"),
6     path("login/", views.UserLogin.as_view(), name="login"),
7     path("logout/", views.UserLogout.as_view(), name="logout"),
8     path("user/", views.UserView.as_view(), name="user"),
9     path("password-reset/", views.PasswordReset.as_view(), name="password-reset"),
10    path("password-reset/<str:encoded_pk>/<str:token>/", views.ResetPasswordAPI.as_view(), name="reset-password"),
11    path("update-profile/", views.UserUpdate.as_view(), name="update-profile"),
12 ]
```

backend / manager / manager / urls.py

 bihany-harsh

User auth first commit

a5004fc · 6 days ago


 History


Code


Blame


23 lines (21 loc) · 845 Bytes


Raw











```
1 """manager URL Configuration
2
3 The `urlpatterns` list routes URLs to views. For more information please see:
4     https://docs.djangoproject.com/en/4.1/topics/http/urls/
5 Examples:
6 Function views
7     1. Add an import: from my_app import views
8     2. Add a URL to urlpatterns: path('', views.home, name='home')
9 Class-based views
10    1. Add an import: from other_app.views import Home
11    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
12 Including another URLconf
13    1. Import the include() function: from django.urls import include, path
14    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15 """
16 from django.contrib import admin
17 from django.urls import path, include
18
19 urlpatterns = [
20     path("admin/", admin.site.urls),
21     path("api/", include("api.urls")),
22     path("auth/", include("auth_system.urls")),
23 ]
```

2 Codebase

*Provide the link to your github repository.
Mention briefly how to navigate the codebase.*

Link to the GitHub organization for our team: <https://github.com/cs253-team2>

Backend repository: <https://github.com/cs253-team2/backend>

Frontend repository: <https://github.com/cs253-team2/frontend>

This organization has **two separate repositories** for **backend and frontend** components. The repositories are private as of now. We have given access to our project mentor Sumit Lahiri.

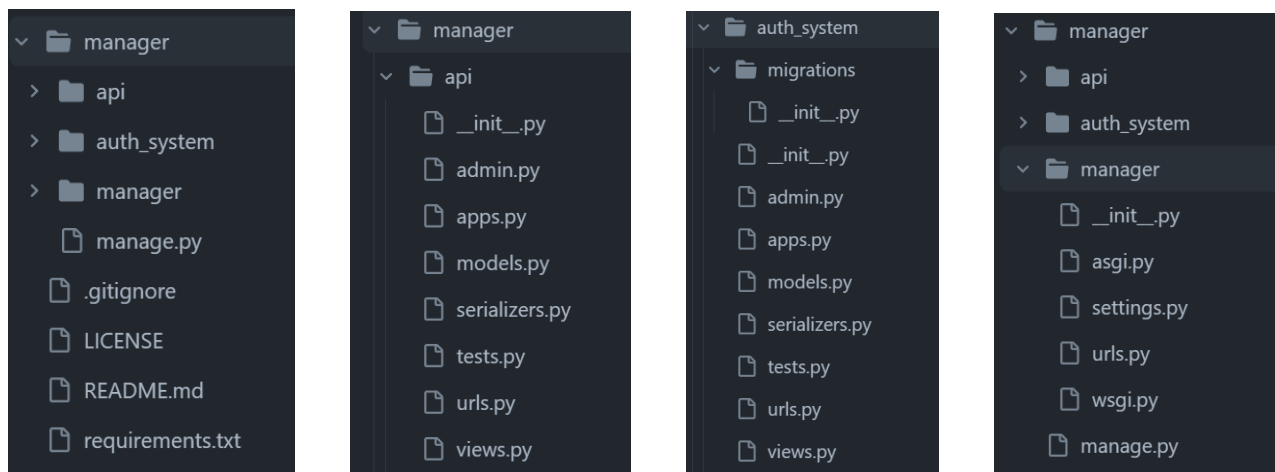


Figure: Backend repository structure

auth_system folder contains all the backend functions to register, login, logout, update profile.

api contains all the protected urls for all other functionality of the software like fetching data, notifications, transaction history and wallet status as well as creating new transactions.

manager is the default folder provided by the django app.

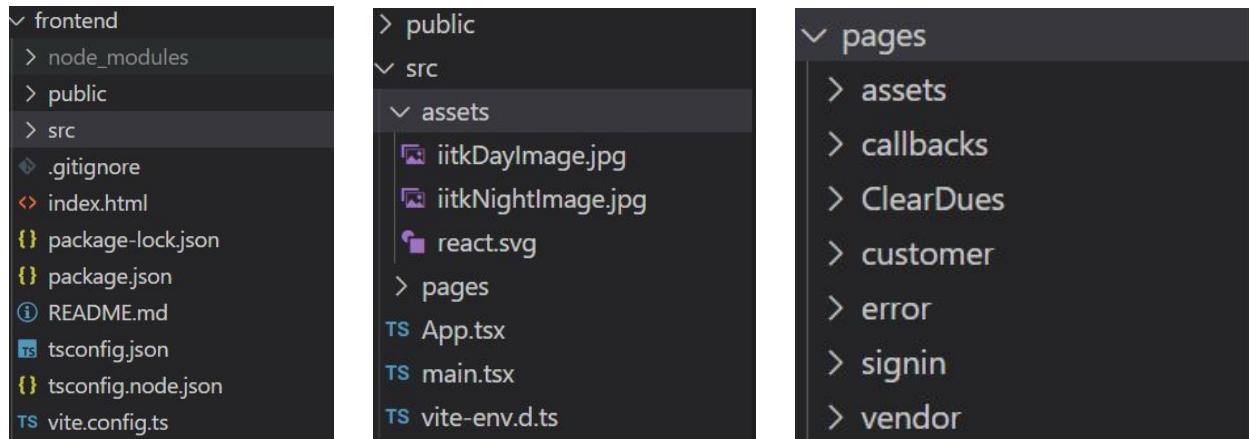


Figure: Frontend repository structure

1. The first picture showcases the structure of the root folder of the frontend repository. It is the standard template that gets created when a new react app is initialized.
2. The second picture shows the location of the main App.tsx that contains all the routes and the assets used in the project.
3. Inside our pages folder, we have the callbacks subfolder which contains all the api get/post/put calls (using axios).
4. All the pages that can be accessed after logging in as a customer/vendor are located inside the customer/vendor subfolder respectively which also has the required components as well.
5. All components associated with the process of signing in/registering are located inside the signin subfolder.
6. Whenever a user tries to access an invalid page/a page that they don't have permission to access, they are shown a page that says 'No Access', both of which are inside the error folder.

3 Completeness

Provide the details of the part of the SRS that have been completed in the implementation.

The features from the SRS that have been implemented in the current version are as follows:

- Completed features:

1. **User Registration:**

- Customers and vendors can sign-up on the platform. The user has to provide his name, phone number, email id and create a password to sign up.
- The registered user can sign in to his account using his email id and password.
[frontend/src/pages/signin/](#)

2. **Wallet facility:**

- Every user is provided a wallet by default, which can be used for transactions. The current balance can be viewed in the profile page as well as the overview page of the user.
- The user has the facility to add money to his/her wallet through the platform, (Linking the transaction with the bank, through API or any other means, has been abstracted out)
[frontend/src/pages/customer/Profile.tsx](#) [frontend/src/pages/customer/Overview.tsx](#)
[frontend/src/pages/vendor/Profile.tsx](#) [frontend/src/pages/vendor/Overview.tsx/](#)

3. **Dues Management:**

- A customer is automatically added to the vendor's ledger on their first transaction with the vendor (*see also*: Instant payment).
- A vendor can manually add a customer to his ledger through the *Add Customer* feature in the customer page.
[frontend/src/pages/vendor/Customers.tsx](#)
- A customer can remove a vendor through the *Remove* feature in the vendor page.
[frontend/src/pages/customer/Vendors.tsx](#)
- Once the customer is registered with the vendor, any expense between them can be added as dues under the *Add Dues* page: [frontend/src/pages/customer/AddDues.tsx](#)
- Additionally, the website also allows a customer to make and manage dues with another customer.

4. **Instant Payment:**

- The customer can make instant payments to the vendor anytime (provided sufficient wallet balance). The customer is automatically registered with the vendor (if not already registered) after the first transaction. [frontend/src/pages/customer/InstantPayment.tsx](#)
- Upon successful transaction, the transaction amount will be deducted from the customer's wallet and will be added to the vendor's wallet.
- Though, the customer mainly aims at customer to vendor transaction, a customer can also make direct payments to another customer as well.

- Removed features:

5. ~~Transaction verification and reporting~~

- ~~○ In case of a suspicious dues update by a vendor, users have the option to report the transaction for further review. It would also be possible to enable a real-time approval facility, in which such a transaction will be completed only when approved by the user.~~
- The above feature has been scrapped since it is redundant to have a two layer check to add a due making the platform less user-friendly.

6. ~~Pending Dues Alert, Autopay~~

- ~~○ Users will be alerted about their pending dues on the platform, at time intervals specified by the user themselves. In addition, admin like the Hall office can mandate payments at times like semester-end, or before mid-semester recess for canteens. Users can also enable an autopay feature through which dues will be cleared automatically at given time intervals, provided the wallet has sufficient balance.~~
- This additional functionality has been omitted in the initial version of the web application and could be incorporated in future versions.

7. ~~Default prevention~~

- ~~○ To prevent cases of default when the user does not pay their dues by the stipulated deadline, we plan to impose a restriction on the dues possible to be accumulated by a user. There will be a minimum balance required in every user's wallet in case they wish to add a due for some vendor. This minimum balance will be relaxable till a fixed grace amount (e.g. Rs. 100). This means that, for a current wallet balance of Rs. X, one can add dues upto Rs. X+200, after which the platform will not allow any more dues to be added. W.r.t. feature #7, if the user does not pay their dues by the admin specified deadline, the pending amount will be deducted automatically from their wallet. Remaining dues (< Rs. 100) will be added to the institute pending dues for the user.~~

- The above feature has been removed in favour of making our platform more user-friendly by allowing the customer to add dues without having any restriction on wallet balance. Sufficient wallet balance would be required at the time of actual payment,

8. ~~Vendor privileges~~

- ~~○ The vendor can control whom to add as an account user for their establishment. The vendor can decline user registration requests, and set an upper limit for pending dues, after which no transactions would be allowed. That is the user would have to clear their existing dues to resume using the account facility at that establishment.~~
- The above feature has been removed as the customer has been allowed to register with the vendor at the time of his/her first transaction(direct payment) with the vendor.

Updated features:

9. Hall specific dues management

- ~~○ Various dues corresponding to the hall of residence of a user, including Mess, Electricity, Canteen, General store will be clubbed together for convenience. The No-Dues certificate required at the end of each semester will be generated automatically and sent to the Hall Office once these dues are cleared by the user.~~
- This feature has been replaced with a more general *clear all dues* functionality which allows the user to clear existing dues with all the vendors.

10. User Interface

- ~~○ The User Interface (UI) will display Wallet balance and pending dues on all pages the user visits, tentatively on the left hand side. On the landing page, the most recent transactions will be displayed to the user. Further, on clicking one of the transactions (which includes vendor details), the user will be shown the most recent transactions corresponding to the vendor they clicked on. There will also be options to add or remove vendors, deposit or withdraw money from the wallet, instant payments and to delete accounts(after clearance of all dues).~~
- Some minor changes were done in the user interface.
- The most recent transaction and the current wallet balance are displayed in the overview page for users.
- The customer can delete the vendor through the delete *vendor* option in the vendor page.
- The *instant payment* and *deposit money* features have been described above in detail.

- The customer can view the associated vendors and their details through the vendor page.
- The vendor can view the registered customers and their details through the customer page.

- Newly Added features:

11. Notifications

- **Welcome Notification :** A notification is sent to the user at the time of his/her first registration on our application.
- **Pending Due Notification :** A notification is sent to the customer when he/she adds any expense to his ledger to convey the pending status of the transaction. A notification is sent to the vendor to convey the addition of the due by the particular customer.
- **Money transfer :** A notification is sent to the user at the time he adds money to his wallet from his bank account.
- **Transaction Success/Failed :** A notification is sent to the customer and vendor to convey the status of the transaction after payment.

12. Update Profile

- An *Update Profile* section has been made available to all users where they can update their phone number, email id and username. The new details will be updated in the database.

Provide the future development plan by listing down the features that will be added in the (may be hypothetical) future versions.

The features that might be possibly added in (hypothetical) future versions are as follows:

1. We plan on facilitating transactions using “real” money, that is, linking bank accounts and implementing actual UPI transactions on our application. This has not been incorporated in the initial version because of security concerns and lack of time. (For completeness in the demonstration, we manually add money to user accounts as of now).
2. We plan on letting vendors initiate a transaction. This transaction will be verified by the customer and can be reported which will send a request to the admin of the software.

Appendix A - Group Log

FRONTEND Implementation Log

6th March-

- Aditya did the initial code setup in react and set the basic layout for the application. Also made the options tab active in the sidebar.
- Pratham set the routing to the pages in the second sidebar and made the layout according to the specifications in the design doc. Also, tokenized the work into components with proper routing to enable group collaboration.

7th March-

- Siddhant completed the 'Profile' page
- Pratham pushed the "Notifications page"

8th March-

- Akshat pushed the 'Vendor' page.
- Kalika pushed the 'Transaction History' page
- Ravija pushed the 'All Dues' page.

9th March-

- Siddhant pushed the 'Registration' page.
- Pratham and Aditya completed routing for vendor-side pages and modified components built on the customer side to meet vendor requirements.

10th March-

- Kalika added functionality to search bar and pushed the changes
- Shantanu added page to allow users to update their profile information

11th March-

- Pratham added pages for instant payment and add dues and made links from overview page
- Akshat added the signin page background images.

14th March-

- Kalika made changes in pages copies to vendor from customer according to vendor requirements
- Kalika added 'profile' and 'update profile' in vendor pages
- Kalika added functionality to update profile button on vendor side

15th March-

- Aditya wrote the api call to fetch vendor data (with hardcoded customer ID).

16th March-

- Siddhant added the api call to fetch profile data for users (with hardcoded customer ID).

- Pratham added the api call to fetch notification data for users (with hardcoded customer ID).

18th March-

- Aditya added the authentication for login and logout.

19th March-

- Siddhant wrote the api call for the 'update profile' page (customer side)
- Siddhant made some styling changes for the entire application and updated the overview page.
- Siddhant added vendor data, notification, transaction history callback.
- Aditya wrote the vendor transaction data api call.
- Aditya added the api calls for 'All Dues', 'Profile', 'Update Profile' on the vendor side.
- Siddhant added api calls to make instant and add dues transactions (customer side).

20th March-

- Kalika pushed the 'credits' page

BACKEND Implementation log

5th March:

- Harsh and Pulkit wrote the first draft of User and transaction models.

6th March

- Harsh revamped the CustomUser, Vendor, Customer, Wallet and Transaction models, with logic for creation and saving of models.

7th March

- Pulkit implemented the Notification model for sending notifications on new user profile creation (vendor/customer), transaction success and transaction failure.
- Harsh updated the django admin panel for the Transaction, User, and Wallet models.

9th March

- Harsh setup the initial views and urls for API calls in the backend manager.
- Pulkit implemented logic for pending dues functionality
- Monil added the views for the notification functionality as well as started to work on the Issues model.

10th March

- Harsh and Monil added the views for pending dues(showing total dues for each sender-receiver pair)
- Monil implemented an issue model for getting the issues of users regarding any transaction

11th March

- Pulkit improved the Issue() model, implemented an issue resolution system and added notifications for Issues creation / resolution

14th March

- Harsh implemented the User register, login, logout Auth on the DjangoRESTFramework for the CustomUsers in the application.

15th March

- Harsh implemented the User Password Reset in case of the 'forgot password' scenario on the DjangoRESTFramework.

16th March

- Harsh implemented the protection of API routes from unauthorized users to protect data of users
- Harsh implemented the backend service for updating the profile and POSTing transactions.

19th March

- Harsh implemented the clear all dues functionality from the backend (clear dues and clear all dues)
- Harsh added the (since the functionality is abstracted out, this is just a mimic of a banking request) Add Balance to wallet functionality.
- Harsh wrote the backend service to fetch data to be displayed on the Overview Navbar and Table.