BARC 2.0 PRDv1

Team Yao and "Friends":

Yuan Yao (Lead)	yuanyao00@ucsb.edu
Chandler Forrest (Scribe)	chandlerforrest@ucsb.edu
Jake Bliss	zakarybliss@ucsb.edu
Vamsi Kalidindi	vkalidindi@ucsb.edu
Alex Ngo	alexanderngo@ucsb.edu

Problem:

A significant portion of becoming an adult in modern America is learning financial responsibility. One of the hurdles that young adults face while focusing on school is taking control and managing their money. It becomes difficult to live within your means when your school expenses are abstracted away online. The ideal solution would provide useful statistics to students based on their spending habits, show exactly where their money is going, while being able to deposit and withdraw money instantly from virtually any source.

To provide context, one of the first things UCSB students learn during their freshman orientation is how to properly pay their dues through UCSB's online payment portal, Billing Accounts Receivable Collection Unit (BARC). BARC keeps track of each student's tuition, on campus rent, health insurance, dining commons meal plans, residence hall printing, lab materials, and other seemingly random fees. It is confusing to see breakdowns of your feeds, hard to get money in and out, and provides a frustrating user experience.

BARC's biggest issue is its inability to show users exactly how money is being spent on campus. As students, its very crucial to filter where money is being spent, so that you can save money and budget appropriately. On BARC, all your charges are just pushed into a big list, which cannot be manipulated or filtered. Furthermore, you can't see how much is spent for a given month or period of time. This is an issue because students should be able to see when they're spending the most money, or for any given quarter that they are enrolled in. If they see that their expenses are higher for a given quarter, they can choose to cut those costs during the following quarter.

Another one of BARC's biggest issues is the way money flows in and out of the system. In order to put money into your BARC account. There are three options - credit card, electronic checks, or international payment. Paying with credit card is ridiculous because it charges a 2.75% fee for the transaction. If you are paying in-state tuition for a year, that's an extra $300. International payment makes your complete the process inside another application instead of handling it natively. This leaves the only sensible way to pay is through an electronic check which makes you enter your full banking information every time you try to complete a transaction. Finally, in order to pay for a due, you actually check it out into a shopping basket, and pay for it using one of the three options. It gives you the impression that you are given a choice whether to pay or not. When you have a positive balance - which if you can figure it out, means the school actually *owes* you money - they mail you a check. A large improvement we aim to create is making the process of taking money in and out of BARC as simple as possible while generating a more sincere way of paying for dues.

Furthermore, BARC doesn't let you export any of this data into a more viewable format, such as a CSV file. If the user wants to further manipulate this data in Excel or any other software, it's not possible. This is something worth tackling, because students shouldn't be solely dependent on BARC's bill tracker to see how much they're spending. Within BARC's financial expenses, it's also hard to keep track of which expense falls into a certain category, such as a restaurant expense, or quarterly gym fee. Everything is added together as a whole bill, which doesn't help filter where money is going. Therefore, one of our biggest priorities to combat this issue is to provide analytics with the data being saved into the new payment application

We want to make payments for on campus charges as simple as possible. If you are purchasing something like a cup of coffee, the entire interaction should take less than 20 seconds. We aim to do this by creating a simple companion app that uses your phone as the payment system. Any purchase made on campus would be reflected for students to see and reflect upon at a later time.

Innovation and Science:

One of our innovation about our product is the NFC/QR payments on campus. Since our goal is to unify and simplify the entire money flow system for universities, it makes sense for us to figure out a simple way where students can make purchases on campus directly from their UniPay account with a snap of a finger. To accomplish this, we need to develop a companion mobile App for students. We will have NFC stands set up at the cashier counter. When students make purchases, they can easily tap their phone to the stands and the payments will be automatically billed to the students account. During the research process, we found out that Android Phone can simulate NFC signals. So this is a great start for us to play around with this technology where we would have two Android device, one emitting signals, mimicking the NFC stands at the cashier counter, and another phone receiving the signal, mimicking when students use their phones to pay. This requires utilizing the Connectivity library coming from Android framework.

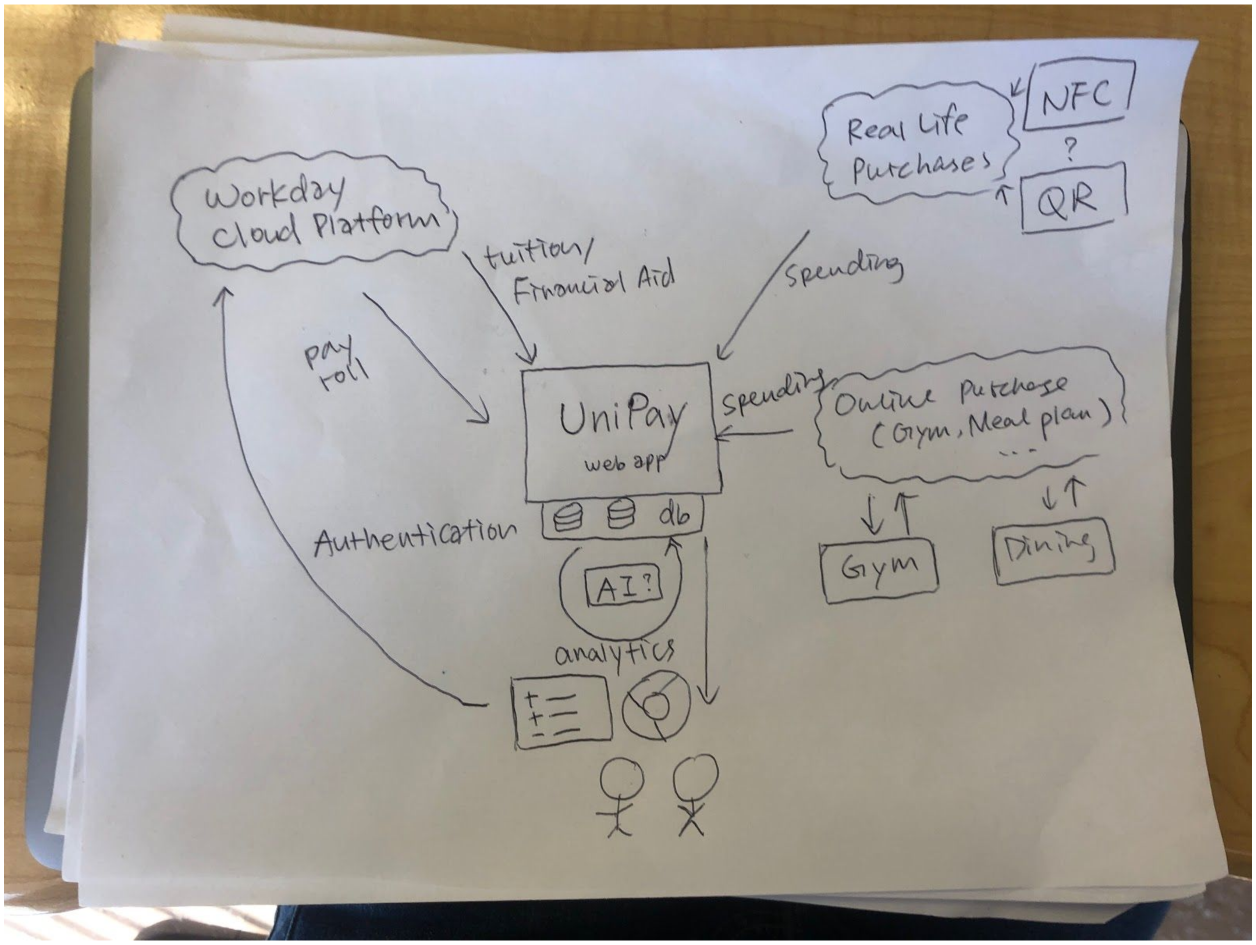
As of the the web app, we are using the combination of ReactJS + NodeJS. By defining a single language for front end and back end, all five of our team members can code review each other's code and learn more than just our own parts. In addition, NodeJS backend provides a significant amounts of community contributed frameworks. As for our plan right, we are using Express and MongoDB for our backend routing and storing services.

In the future, after we have implemented everything we have planned, we would like to explore the possibilities with machine learning to generate suggestions on how to improve students' spending habits. We could also do some data manipulation and comparisons to let the student have a better idea on how much they are spending compare to their peers.

Core Technical Advances:

Our System provides a unified, easy to use platform for all student finances. To accomplish this, we draw financial information from Workday Cloud Platform, on campus purchases, and online purchases related the university. We then analyze this information, generate informative analytics and display them to the user. Our system also uses artificial intelligence to track student's financial habits and provide suggestions. These suggestions are intended to help guide the student's spending so they can budget appropriately and maintain healthy financial habits. To integrate on campus purchases into our web app, we provide an Android app which allows student to make payments at on campus stores via Near Field Communication or with a QR code.

System Architecture Overview:



Use Cases:

Use case: Basic Analytics
Actors: Current Student, Account database, Workday API
Precondition:
Student is logged in and the account has collected information (there is some from of data to actually analyze in the account database)
Flow of Events:
Basic Path:
1) Student is currently logged into the app
2) Student spends money/receives money
3) Account database is updated by retrieving data from Workday API or from other spending
4) Account database analyzed and results are shown (decide what form it is displayed in)

Alternative Paths:
1) A. Student has no data in the database, an error is shown about not having enough data for analysis (exceptional case that must be handled)
2) A. Problems connecting to the workday api (exceptional case that must be handled)

Postcondition:
Some form of analysis is shown in the app for the user to view or an error is shown

Use case: Financial History
Actors: Current Student, Account database
Precondition:
Student is logged in
Flow of Events:
Basic Path:
1) Student is currently logged into the app
2) Student clicks on financial history
3) Page should pop-up showing past input/output based on account database

Alternative Paths:
Postcondition:
Page that shows a list of financial history should be displayed to the user.

Use Case: Real Life Purchases (Bookstores, Small shops, Coffee stand)
Actors:
Student who wants buy stuff on campus, Merchandise, Student account databases, Mobile app
Precondition:
Students are using our system and they would like to make purchases on campus
Flow of Events:
Basic Path:
1) Student is logged in with their account on their phone
2) Student goes to the store and buys stuff
3) Head to the checkout desk, the cashier will put the value into the NFC stand
4) Student opens the app, scan their phone to the NFC stand
5) Their balance is automatically deducted.

Alternative path:
1) A. Student doesn't have enough money to cover the transaction (exceptional case that must be handled)
   B. Student is notified that they don't have enough money.
   C. Transaction is declined
2) A. Error connecting to the database (exceptional case that must be handled)
   B. Error message pops up asking the user to try again.

Postcondition:
Student has successfully paid for the their transaction and can check their transactions immediately or transaction has not been made and the correct error message has popped up informing the user.

Use Case: School-Wide Purchases (Gym, Meal Plan)
Actors: Enrolled Student
Precondition:
User has been charged for a meal plan, gym membership, or another school services
Flow of Events:
Basic Path:
1) Student is enrolled in UCSB.
2) Students get a flat fee for certain services, such as gym membership
3) Students choose which Meal Plan they want, and system charges student based on meal plan choice. (More meals means greater sum charged)
4) System stores student service charges into database.

Alternative Paths:
1. Student doesn't choose meal plan, in which case the system will place a standard charge for gym and other school services.

Postcondition:
Student account balance has been updated to include standard school-wide charges. Analytics will use this data to give students the appropriate suggestions.

Use Case: Checking UCSB Balance
Actors: Student
Precondition:
Student is already logged into the system and chooses "check balance" option
Flow of Events:
Basic Path:
1) System retrieves student account from the Student Account Database
2) System displays the balance attribute from the student account to the student.

Postcondition: Student's balance has been correctly displayed.

Use case: User Login on Website
Actors: Enrolled Student, Workday Authentication API, Student Database, Application Server
Preconditions: Student has a pre-existing account which is created and provided by the school. Student is also not currently logged in to the website
Flow of Events:
Basic Path:
1) Application server presents the login screen
2) Enrolled student enters both their correct school email address and correct password
3) Application server sends credentials to the Workday Authentication API
4) Workday Authentication API checks that both the email address and password match a user within the Student Database
5) User is properly authenticated and Workday API sends a callback to the Application Server, giving the "ok"

Alternative Paths:
1) If an enrolled student has forgotten their password, the student is presented a seperate screen to send an email to their school email address to reset it
2) Student closes the web app page and does not choose to log in
3) Student enters wrong password or wrong email and is asked to re-enter credentials after Workday has failed to authenticate them

Postcondition:
Student user is given access to the main web application page

Use case: Check Spending From A Specific Month
Actors: Enrolled Student, Student Database, Application Server, Main Web App Page
Preconditions: Student has a pre-existing account, logged in, and has data from the chosen month
Flow of Events:
Basic Path:
1) User clicks on the drop down menu which displays month names
2) User then clicks on the specific month they wish to view transaction data from
3) Main Web Page redirects to a monthly view, there would also be options to refine by day, week, or just view the entire month

Alternative Paths:
1) Use clicks on month which they weren't active. It shows no spending data.

Postcondition: Student user has full view of their monthly spending for chosen month

Use case: Logout From Account on Web App
Actors: Enrolled Student, Application Server, Main Web App Page
Preconditions: Student has a pre-existing account and is logged in
Flow of Events:
Basic Path:
1) Student clicks on the settings icon
2) Student clicks on the the logout button from the settings menu
3) Web App notifies the Application Server and changes the status of the user

Alternative Paths:
Postcondition: The status of the Student is changes in the application server to "logged out" and the login screen is displayed on the webapp

Use case: Logout From Account on Web App
Actors: Enrolled Student, Application Server, Main Web App Page
Preconditions: Student has a pre-existing account and is logged in
Flow of Events:
Basic Path:
1) Student clicks on the settings icon
2) Student clicks on the the logout button from the settings menu
3) Web App notifies the Application Server and changes the status of the user

Alternative Paths:
Postcondition: The status of the Student is changes in the application server to "logged out" and the login screen is displayed on the webapp

Use Case: Generate graph to show where expenses are being made.
Actors: User, Workday API, Application Server, Main Web Page
Preconditions: User is already enrolled at UCSB. User has already been charged for services.
Flow of Events:
Basic Path:

1. User logs into application.
2. User clicks on Spending toolbar.
3. User clicks on specific month or year of spending
4. User can generate a graph to show proportional data for costs such as meal plan, school services, restaurant expenses etc.

Alternative Paths: No alternative paths.
Postcondition: Student can see a pie graph or bar graph that shows expenses proportional to spending. Can be filtered based on day, month, and year.

Use Case: Export data to CSV
Actors: User, Workday API, Application Server, Main Web Page.
Preconditions: User is already enrolled at UCSB. User has accumulated charges to their account that can be viewed.
Flow of Events:
Basic Path:

1) User has logged onto Main web page.
2) User goes to drop down menu, and clicks on Spending toolbar.
3) User can filter data based on time, area of expense, and location.
4) Application will output data to user, which the user can then manipulate.
5) User can click "export data" which will then export all the chosen information into a CSV file.

Alternative Paths:

1) Student has no charges for a given month, in which case no data can be exported for that given period of time.

Postcondition: Student has received data exported from the main web app, into a CSV file.

Technologies Employed:

1. React.js
2. Node.js
3. Github
4. Trello
5. MongoDB
6. Postman
7. Jest