Rohan Kapoor, Austin Kramer, Everett Hu, Yuriy Tytla
rkapoor6, ajkrame2, ehu6, tytla2
04/28/2014

Team Thinking of a Better Name – Project Fastest Food

Contents

Rohan Kapoor, Austin Kramer, Everett Hu, Yuriy Tytla
rkapoor6, ajkrame2, ehu6, tytla2
04/28/2014

Description

Project Fastest Food is a database driven application that helps college students find the fastest food delivery service available to them. It runs on a LAMP stack (Linux server, Apache webserver, MySQL database, and PHP). Project Fastest Food uses: PHP with the Twig template engine, HTML5, Foundation 5 CSS framework, and client side JavaScript.

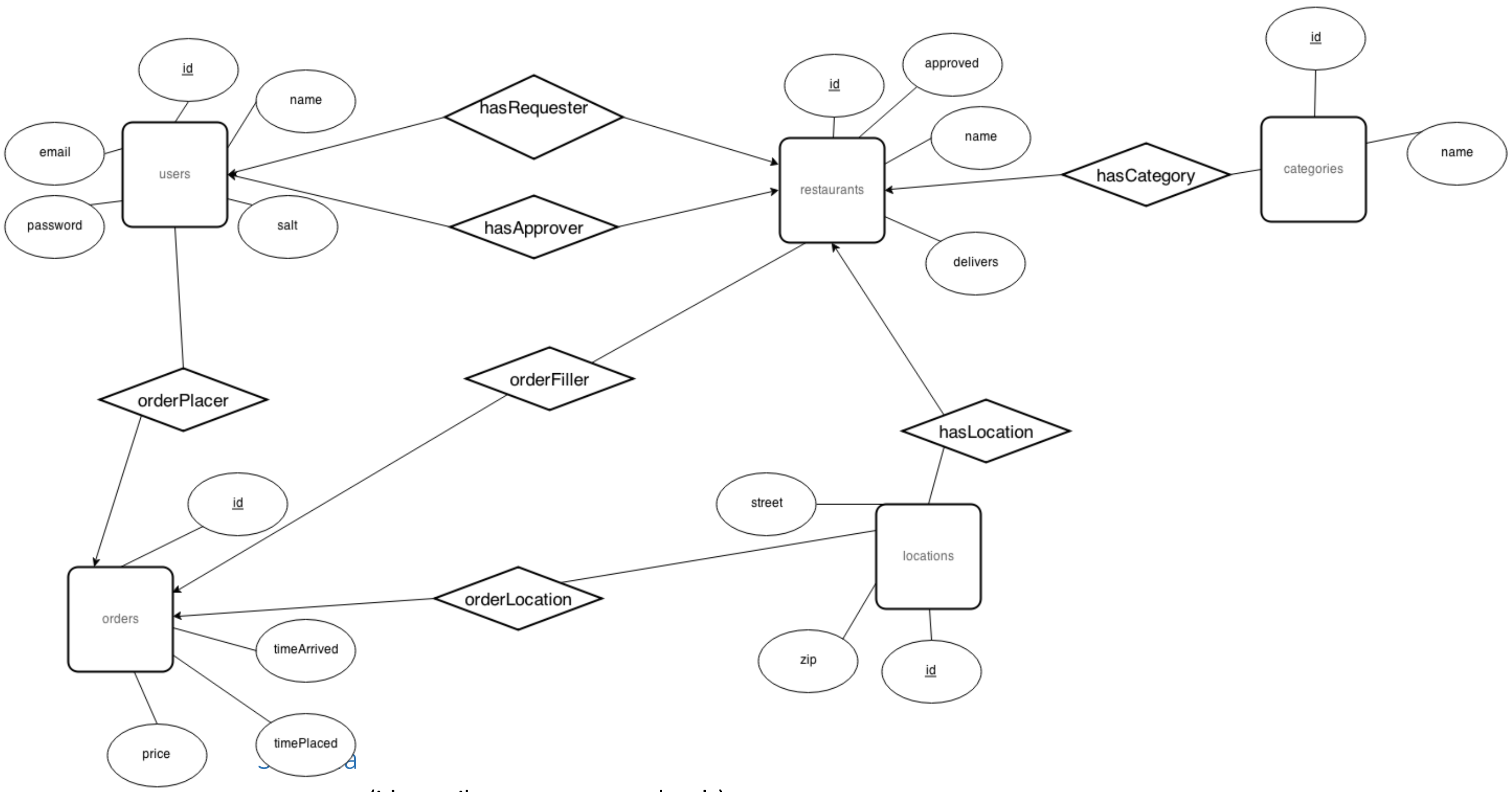
Utility

Many college students live in dorms and don't have access to kitchens to cook their own food. They have to rely on food delivery services to eat and when they are hungry, they want their food to arrive quickly. Our website gives users the best chance at eating quickly by recommending them restaurants that have consistently made fast deliveries.

Database Specifics

We use a MySQL database with several tables to store our data. The ER Diagram and Schema (below) provide more details.

ER Diagram



- **users**(<u>id</u>, email, name, password, salt)
    - id is the primary key
- **restaurants**(<u>id</u>, name, location, category, delivers, approved, requester, approver)
    - id is the primary key

- location is a foreign key to the locations table
- category is a foreign key to the categories table
- requester is a foreign key to the users table
- approver is a foreign key to the users table

- **orders**(id, timeArrived, timePlaced, price, userID, restaurant, location)
  - id is the primary key
  - userID is a foreign key to the users table
  - restaurant is a foreign key to the restaurants table
  - location is a foreign key to the locations table
- **locations**(id, street, zip)
  - id is the primary key
- **categories**(id, name)
  - id is the primary key

Data Collection

We asked our friends to become test users and log their orders from the past few months. This gave us a large enough dataset to build the algorithms for recommendation. Once we built our GrubHub parser, this vastly simplified the data entry process for our friends.

Functionality

Basic Functions

- User Account Creation/Login – All passwords are hashed with randomized salts for a secure experience
- Adding/Editing/Deleting Orders – We make sure to verify that the user can only modify and delete their own orders to prevent malicious activity
- Adding/Viewing/Searching Restaurants – We verify that any restaurants added are legitimate (manually). This verification has been turned off for the demo. When viewing restaurants, we allow users to specify a type of restaurant and then search the database to find restaurants matching that type.

Advanced Functions

- Recommendation Engine – Based on the overall order history, we calculate the average delivery time per restaurant and use that to provide a sorted list indicating which restaurants are the most likely to deliver quickly to your location. This utilizes a view which is regenerated when necessary.
- GrubHub Email Parser – One of the ways to record orders is by copying and pasting an email from GrubHub. Whenever you order food through GrubHub, they send you an email containing most of the information pertinent to us. We've implemented a solution that allows users to copy and paste that email into a text box on the site which then parses the contents to prefill the order form with the details found. We then allow the user to manually confirm before adding the data to the database.

Explain one basic function: Edit Order

After a user has logged an order, they have the ability to edit certain fields already stored in the database in case they made mistakes. We start the process by knowing the id of the order that the user wants to edit as well as the id of the user for authorization purposes.

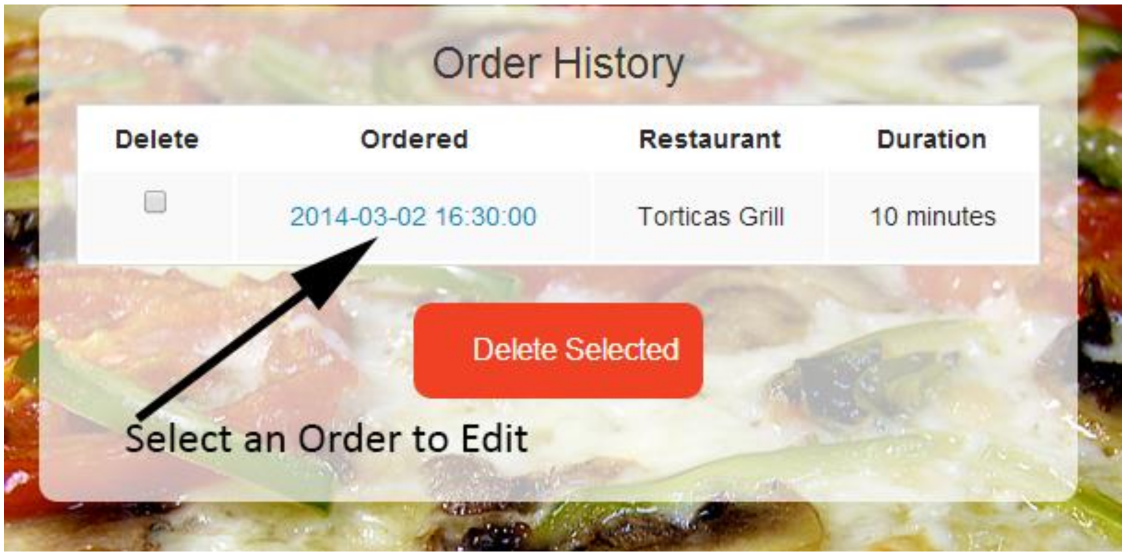
We use SQL Prepared Statements to protect against injection.

```
$stmt = $mysql_con->prepare("SELECT orderID, id, timePlaced, timeArrived, name, durationSeconds, location, price
        FROM restaurants NATURAL JOIN (
            SELECT id AS orderID, restaurant AS id, timePlaced, timeArrived, price,
                TIMESTAMPDIFF(second, timePlaced, timeArrived) as durationSeconds
            FROM orders where userID=? AND id=?
        ) AS DeliverTimes");

$stmt->bind_param('ii', $id, $orderID);
$stmt->execute();

$result = $stmt->get_result();
```

This result is passed on to the template which displays the editable fields. Once the user has edited them, we POST the data to a processing file which runs the following query after sanitizing the user input to protect against injection.

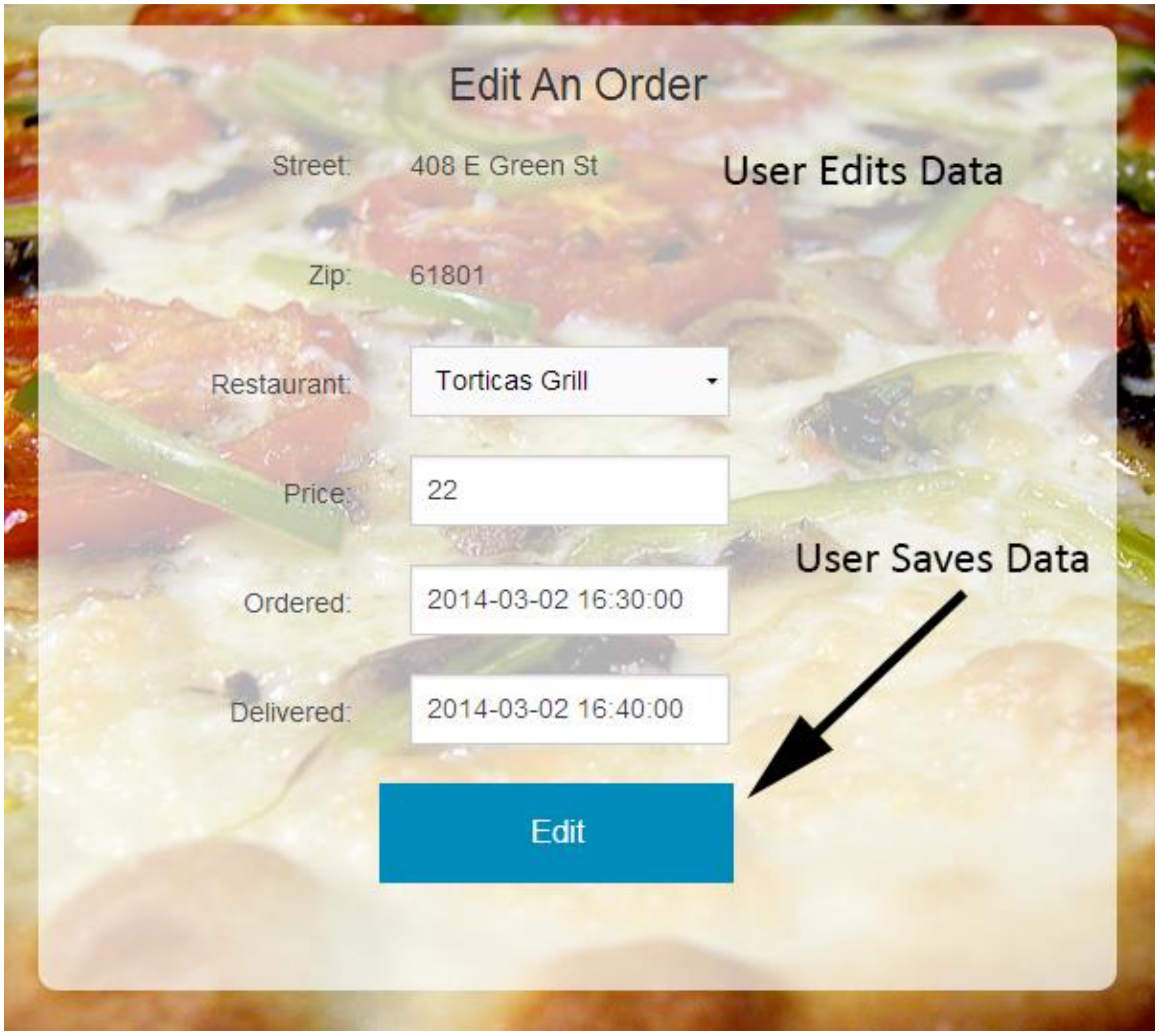
```
$prep_stmt = "UPDATE orders SET restaurant=?, price=?, timePlaced=?, timeArrived=?
    WHERE id=? AND userID=?";
$stmt = $mysql_con->prepare($prep_stmt);

$uid = $_SESSION['user_id'];

if ($stmt) {
    $stmt->bind_param('idssii', $restaurant, $price, $time_placed, $time_delivered, $orderID, $uid);
    $stmt->execute();
}
```

Here's the corresponding user workflow:



Delete	Ordered	Restaurant	Duration
☐	2014-03-02 16:30:00	Torticas Grill	10 minutes

Rohan Kapoor, Austin Kramer, Everett Hu, Yuriy Tytla
rkapoor6, ajkrame2, ehu6, tytla2
04/28/2014



Edit An Order

Street: 408 E Green St

Zip: 61801

Restaurant: Torticas Grill

Price: 22

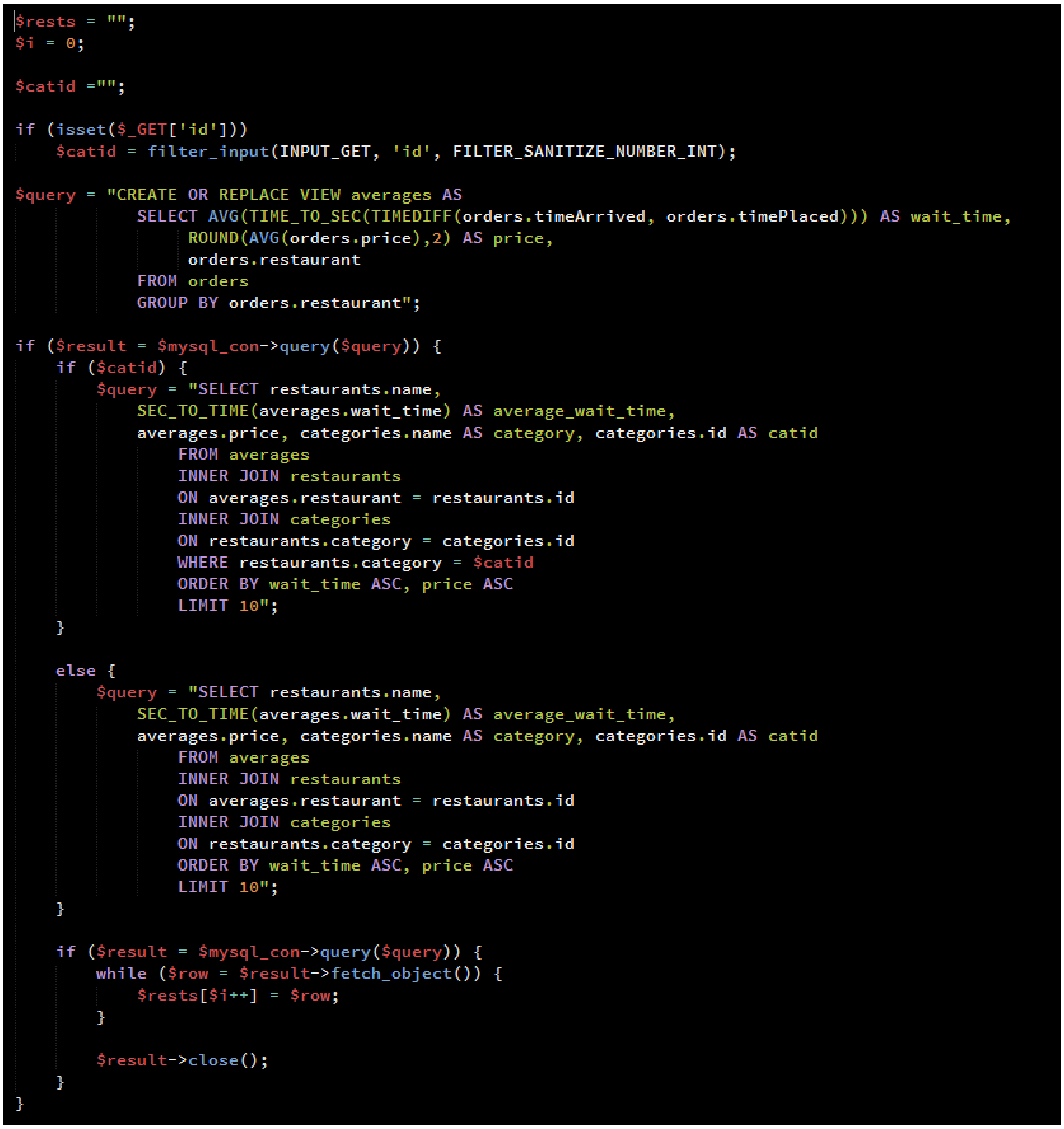
Ordered: 2014-03-02 16:30:00

Delivered: 2014-03-02 16:40:00

Advanced Functions

Recommendation Engine

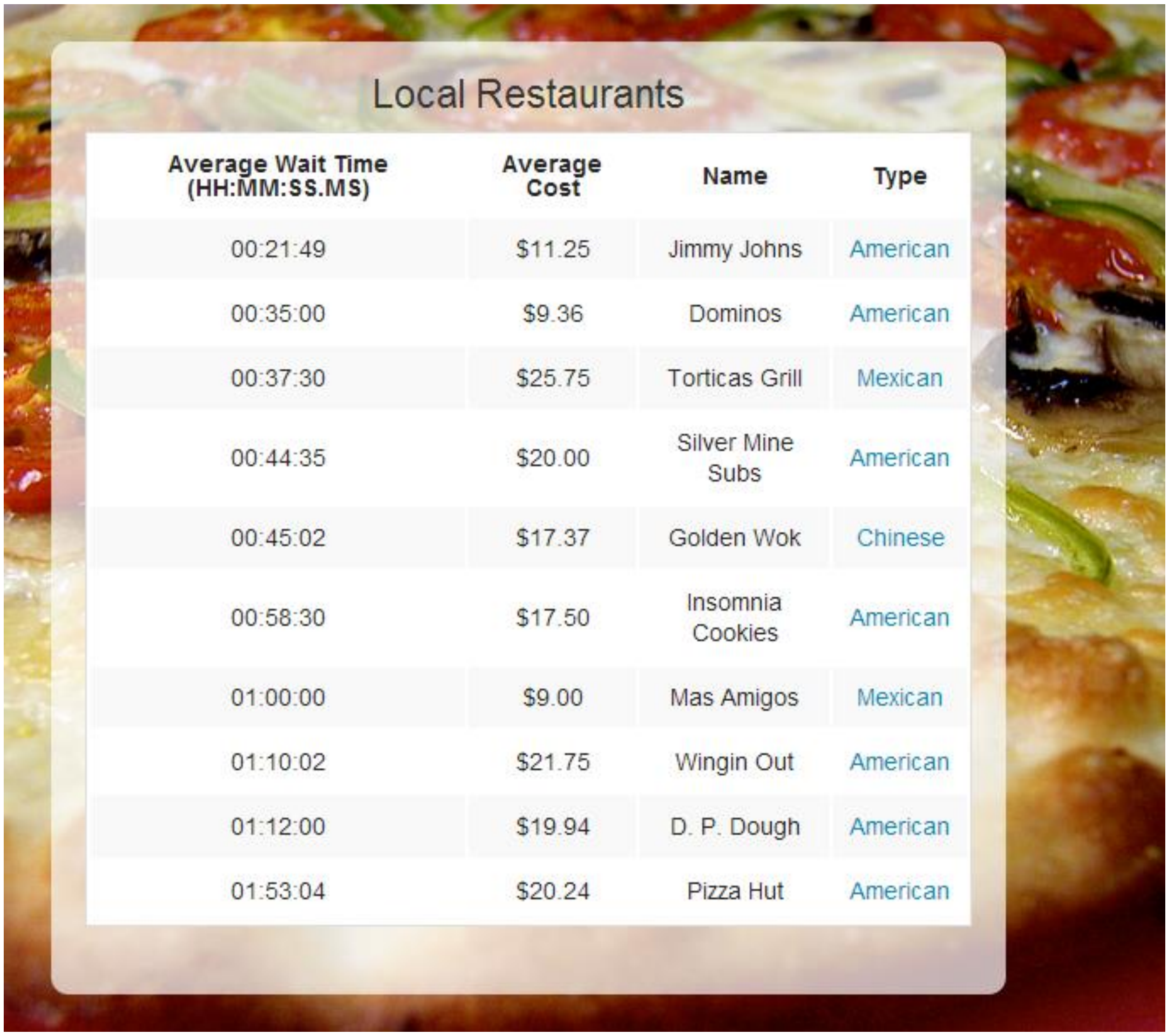
We do not use SQL Prepared Statements for the recommendation engine because we are not using any user input.



```
$rests = "";
$i = 0;

$catid ="";

if (isset($_GET['id']))
    $catid = filter_input(INPUT_GET, 'id', FILTER_SANITIZE_NUMBER_INT);

$query = "CREATE OR REPLACE VIEW averages AS
            SELECT AVG(TIME_TO_SEC(TIMEDIFF(orders.timeArrived, orders.timePlaced))) AS wait_time,
                 ROUND(AVG(orders.price),2) AS price,
                 orders.restaurant
            FROM orders
            GROUP BY orders.restaurant";

if ($result = $mysql_con->query($query)) {
    if ($catid) {
        $query = "SELECT restaurants.name,
            SEC_TO_TIME(averages.wait_time) AS average_wait_time,
            averages.price, categories.name AS category, categories.id AS catid
                FROM averages
                INNER JOIN restaurants
                ON averages.restaurant = restaurants.id
                INNER JOIN categories
                ON restaurants.category = categories.id
                WHERE restaurants.category = $catid
                ORDER BY wait_time ASC, price ASC
                LIMIT 10";
    }

    else {
        $query = "SELECT restaurants.name,
            SEC_TO_TIME(averages.wait_time) AS average_wait_time,
            averages.price, categories.name AS category, categories.id AS catid
                FROM averages
                INNER JOIN restaurants
                ON averages.restaurant = restaurants.id
                INNER JOIN categories
                ON restaurants.category = categories.id
                ORDER BY wait_time ASC, price ASC
                LIMIT 10";
    }

    if ($result = $mysql_con->query($query)) {
        while ($row = $result->fetch_object()) {
            $rests[$i++] = $row;
        }

        $result->close();
    }
}
```

The Recommendation Engine has two parts:

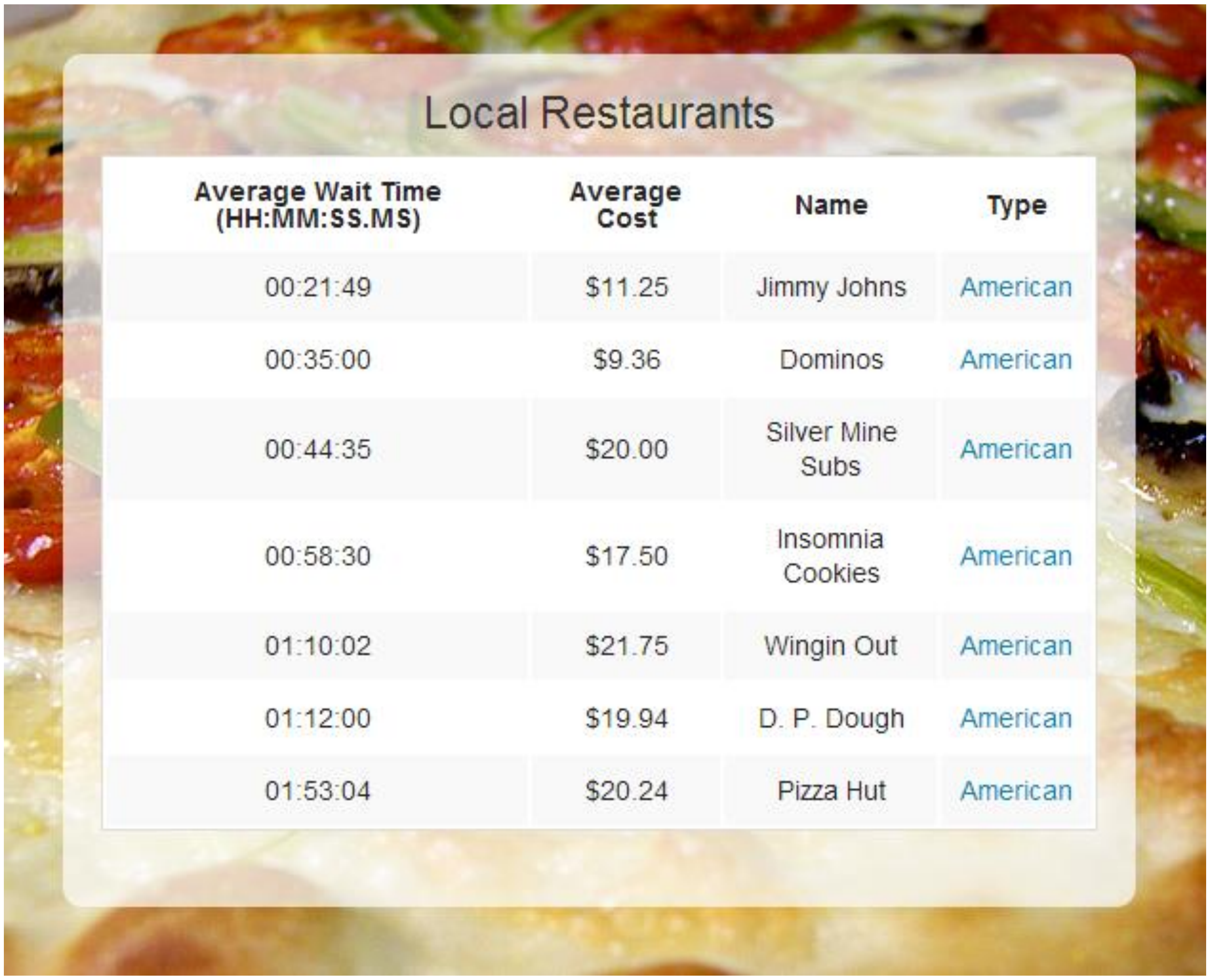
- The first part generates a view which has the average wait time of all restaurants, using data from all of the orders.
- The second part selects from that view restaurants from either all categories (if the user just wants the fastest food) or from a specific category (if the user wants the fastest food from a category) and then sorts it by the delivery time and the price in ascending order. We've limited this to 10 results so that the user isn't given too many choices as the amount of data in the database increases.



Local Restaurants

Average Wait Time (HH:MM:SS.MS)	Average Cost	Name	Type
00:21:49	$11.25	Jimmy Johns	American
00:35:00	$9.36	Dominos	American
00:37:30	$25.75	Torticas Grill	Mexican
00:44:35	$20.00	Silver Mine Subs	American
00:45:02	$17.37	Golden Wok	Chinese
00:58:30	$17.50	Insomnia Cookies	American
01:00:00	$9.00	Mas Amigos	Mexican
01:10:02	$21.75	Wingin Out	American
01:12:00	$19.94	D. P. Dough	American
01:53:04	$20.24	Pizza Hut	American

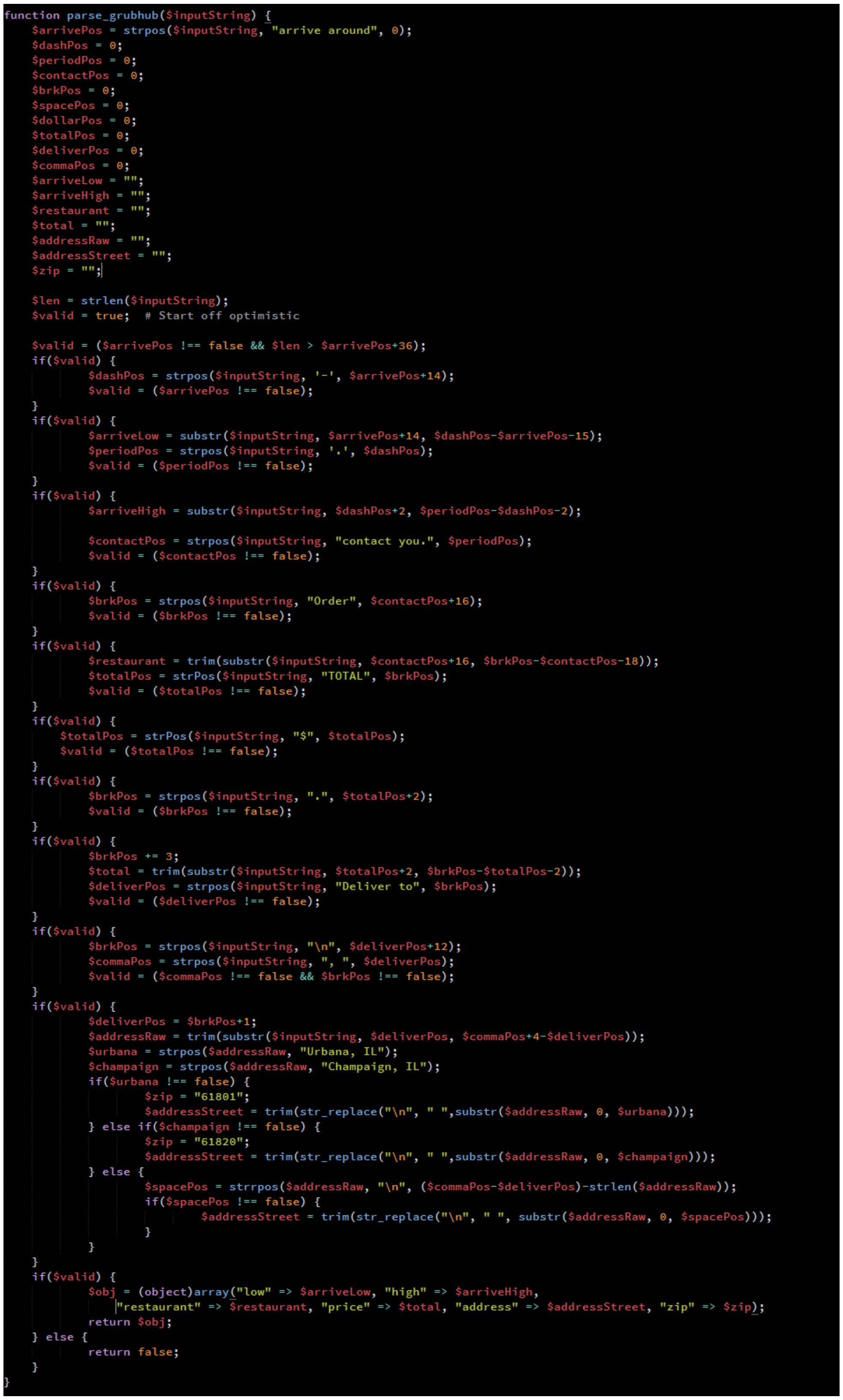
If the user clicks on a category, they get a list of restaurants from that category only.



Local Restaurants

Average Wait Time (HH:MM:SS.MS)	Average Cost	Name	Type
00:21:49	$11.25	Jimmy Johns	American
00:35:00	$9.36	Dominos	American
00:44:35	$20.00	Silver Mine Subs	American
00:58:30	$17.50	Insomnia Cookies	American
01:10:02	$21.75	Wingin Out	American
01:12:00	$19.94	D. P. Dough	American
01:53:04	$20.24	Pizza Hut	American

GrubHub Email Parser

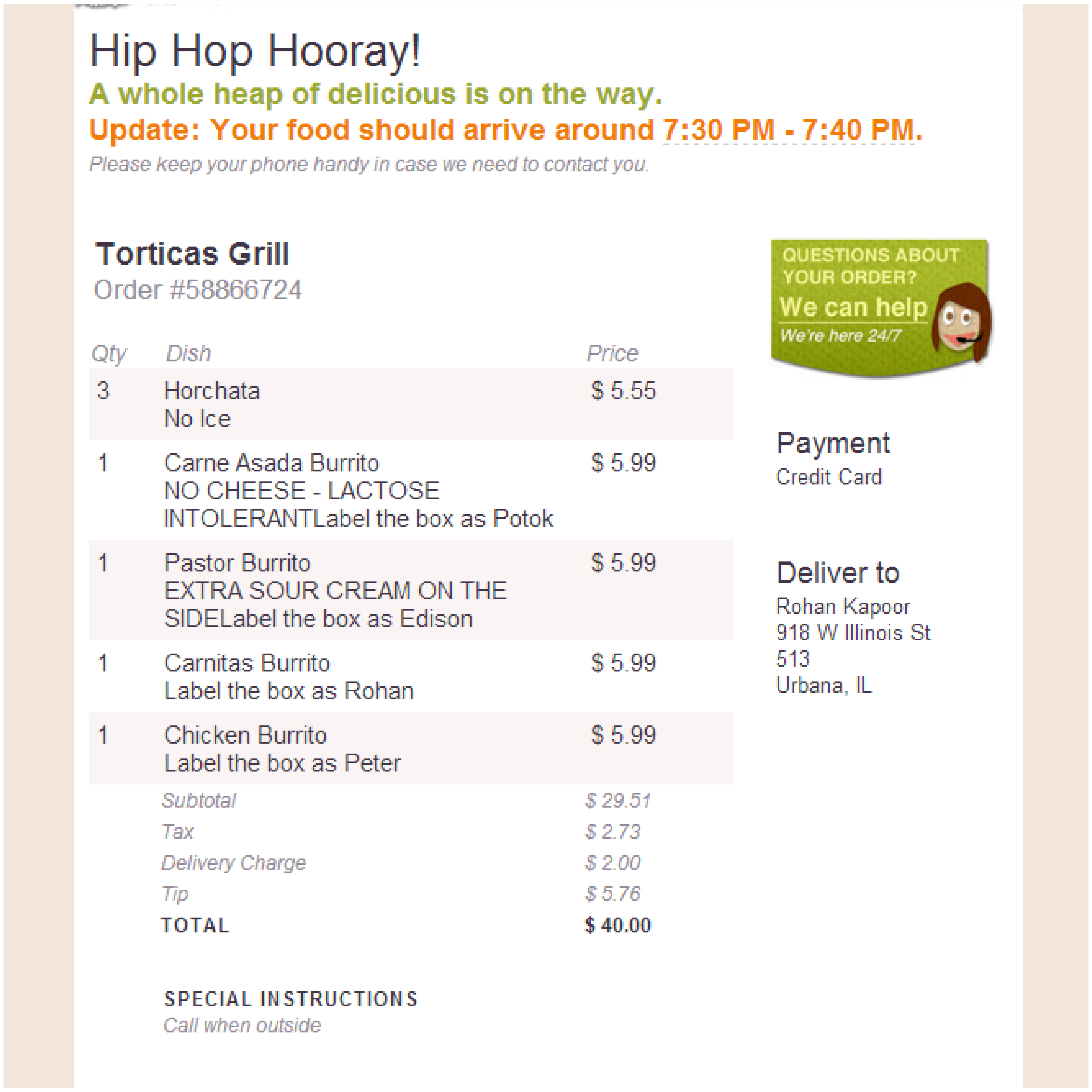


```php
function parse_grubhub($inputString) {
    $arrivePos = strpos($inputString, "arrive around", 0);
    $dashPos = 0;
    $periodPos = 0;
    $contactPos = 0;
    $brkPos = 0;
    $spacePos = 0;
    $dollarPos = 0;
    $totalPos = 0;
    $deliverPos = 0;
    $commaPos = 0;
    $arriveLow = "";
    $arriveHigh = "";
    $restaurant = "";
    $total = "";
    $addressRaw = "";
    $addressStreet = "";
    $zip = "";

    $len = strlen($inputString);
    $valid = true;  # Start off optimistic

    $valid = ($arrivePos !== false && $len > $arrivePos+36);
    if($valid) {
            $dashPos = strpos($inputString, '-', $arrivePos+14);
            $valid = ($arrivePos !== false);
    }
    if($valid) {
            $arriveLow = substr($inputString, $arrivePos+14, $dashPos-$arrivePos-15);
            $periodPos = strpos($inputString, '.', $dashPos);
            $valid = ($periodPos !== false);
    }
    if($valid) {
            $arriveHigh = substr($inputString, $dashPos+2, $periodPos-$dashPos-2);

            $contactPos = strpos($inputString, "contact you.", $periodPos);
            $valid = ($contactPos !== false);
    }
    if($valid) {
            $brkPos = strpos($inputString, "Order", $contactPos+16);
            $valid = ($brkPos !== false);
    }
    if($valid) {
            $restaurant = trim(substr($inputString, $contactPos+16, $brkPos-$contactPos-18));
            $totalPos = strPos($inputString, "TOTAL", $brkPos);
            $valid = ($totalPos !== false);
    }
    if($valid) {
        $totalPos = strPos($inputString, "$", $totalPos);
        $valid = ($totalPos !== false);
    }
    if($valid) {
            $brkPos = strpos($inputString, ".", $totalPos+2);
            $valid = ($brkPos !== false);
    }
    if($valid) {
            $brkPos += 3;
            $total = trim(substr($inputString, $totalPos+2, $brkPos-$totalPos-2));
            $deliverPos = strpos($inputString, "Deliver to", $brkPos);
            $valid = ($deliverPos !== false);
    }
    if($valid) {
            $brkPos = strpos($inputString, "\n", $deliverPos+12);
            $commaPos = strpos($inputString, ", ", $deliverPos);
            $valid = ($commaPos !== false && $brkPos !== false);
    }
    if($valid) {
            $deliverPos = $brkPos+1;
            $addressRaw = trim(substr($inputString, $deliverPos, $commaPos+4-$deliverPos));
            $urbana = strpos($addressRaw, "Urbana, IL");
            $champaign = strpos($addressRaw, "Champaign, IL");
            if($urbana !== false) {
                    $zip = "61801";
                    $addressStreet = trim(str_replace("\n", " ",substr($addressRaw, 0, $urbana)));
            } else if($champaign !== false) {
                    $zip = "61820";
                    $addressStreet = trim(str_replace("\n", " ",substr($addressRaw, 0, $champaign)));
            } else {
                    $spacePos = strrpos($addressRaw, "\n", ($commaPos-$deliverPos)-strlen($addressRaw));
                    if($spacePos !== false) {
                            $addressStreet = trim(str_replace("\n", " ", substr($addressRaw, 0, $spacePos)));
                    }
            }
    }
    if($valid) {
            $obj = (object)array("low" => $arriveLow, "high" => $arriveHigh,
                "restaurant" => $restaurant, "price" => $total, "address" => $addressStreet, "zip" => $zip);
            return $obj;
    } else {
            return false;
    }
}
```

When a user orders food from GrubHub, they send out an email confirmation, which we can use as the source for an order logging entry. The user starts by copying the main body of the email, from the header at the top to the address at the bottom, and pasting it into a text field on our site. We then pull the string from the POST data when they submit, and pass it to the parser.

- The parser starts by checking the string for a couple properties, namely that it's long enough and contains the order confirmation text.
- The parser iterates its way through the file using the PHP string processor functions. It searches for keywords it knows to be present in an email, and looks for relevant bits of information relative to them. This can be tough when you can't anticipate the number of spaces between the word "Total" and the actual price, or when the address may or may not have an extra line containing a room number. We also have to match a city name with a zip code. The current version takes all of this into account. If, at any point in the file, the anticipated constraints are violated, the parser fails.
- If the parser fails, the user is directed to an unfilled form. If it succeeds, the values are passed into the order placement form if they match the database information. The user is free to confirm or modify them before they click "submit."

An example email from GrubHub



Hip Hop Hooray!

A whole heap of delicious is on the way.

Update: Your food should arrive around 7:30 PM - 7:40 PM.

Please keep your phone handy in case we need to contact you.

Torticas Grill

Order #58866724

Qty	Dish	Price
3	Horchata No Ice	$ 5.55
1	Carne Asada Burrito NO CHEESE - LACTOSE INTOLERANTLabel the box as Potok	$ 5.99
1	Pastor Burrito EXTRA SOUR CREAM ON THE SIDELabel the box as Edison	$ 5.99
1	Carnitas Burrito Label the box as Rohan	$ 5.99
1	Chicken Burrito Label the box as Peter	$ 5.99
	Subtotal	*$ 29.51*
	Tax	*$ 2.73*
	Delivery Charge	*$ 2.00*
	Tip	*$ 5.76*
	TOTAL	**$ 40.00**

SPECIAL INSTRUCTIONS

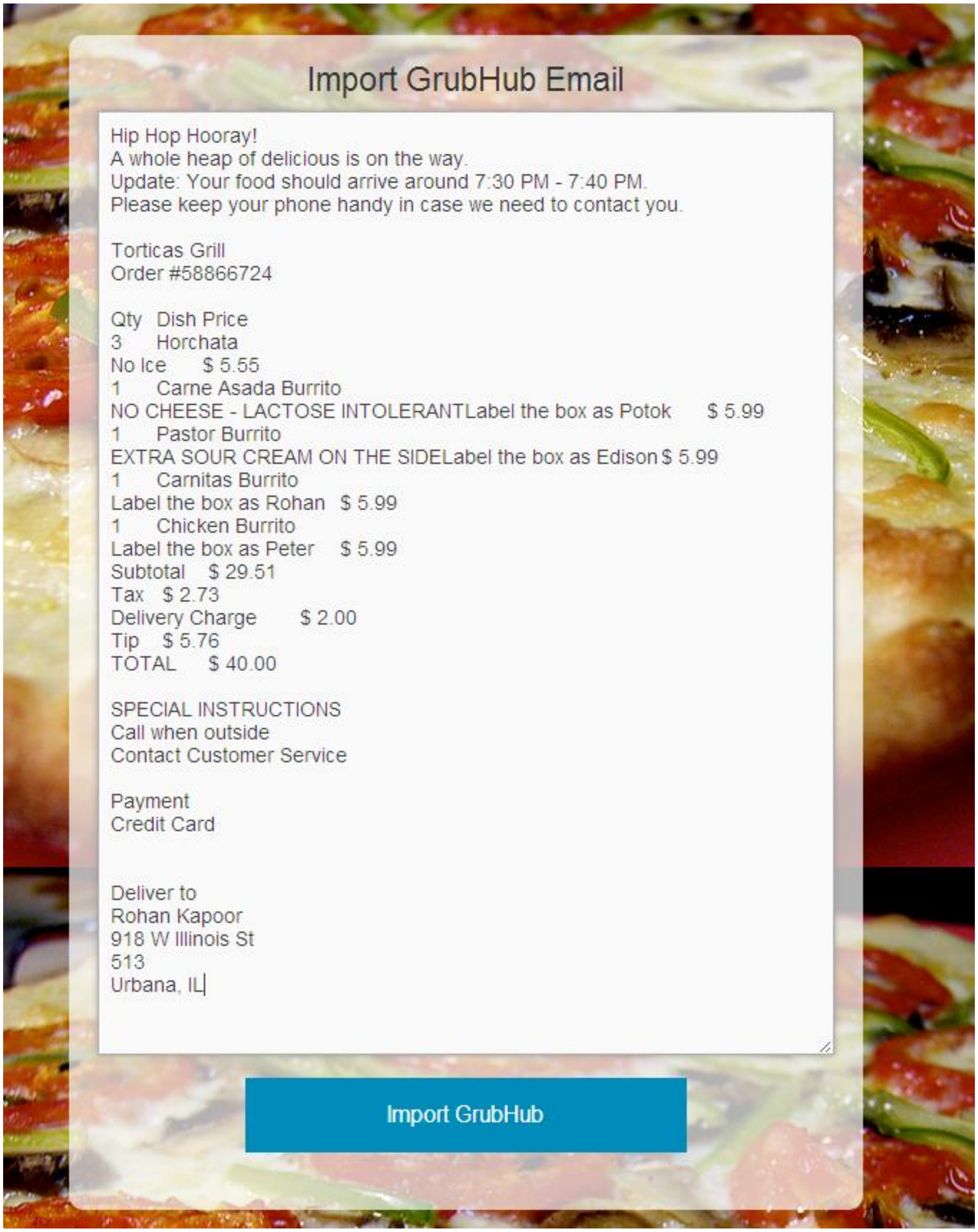
Call when outside

QUESTIONS ABOUT YOUR ORDER?
We can help
We're here 24/7

Payment
Credit Card

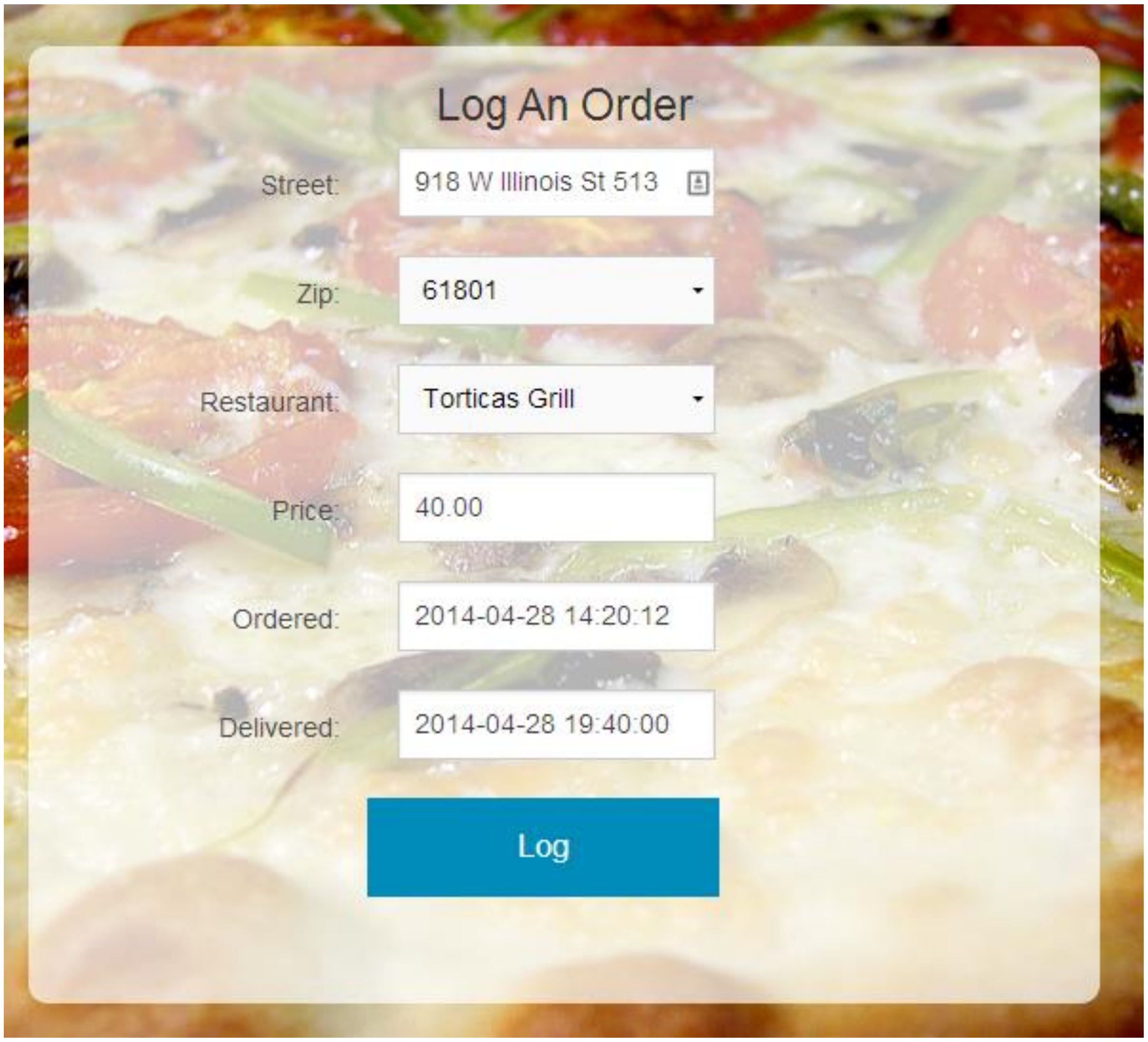
Deliver to
Rohan Kapoor
918 W Illinois St
513
Urbana, IL

The email copied into the textfield



```
Hip Hop Hooray!
A whole heap of delicious is on the way.
Update: Your food should arrive around 7:30 PM - 7:40 PM.
Please keep your phone handy in case we need to contact you.

Torticas Grill
Order #58866724

Qty  Dish Price
3     Horchata
No Ice     $ 5.55
1     Carne Asada Burrito
NO CHEESE - LACTOSE INTOLERANTLabel the box as Potok     $ 5.99
1     Pastor Burrito
EXTRA SOUR CREAM ON THE SIDELabel the box as Edison $ 5.99
1     Carnitas Burrito
Label the box as Rohan  $ 5.99
1     Chicken Burrito
Label the box as Peter    $ 5.99
Subtotal    $ 29.51
Tax   $ 2.73
Delivery Charge      $ 2.00
Tip   $ 5.76
TOTAL     $ 40.00

SPECIAL INSTRUCTIONS
Call when outside
Contact Customer Service

Payment
Credit Card


Deliver to
Rohan Kapoor
918 W Illinois St
513
Urbana, IL
```

After clicking the “Import GrubHub” button



Log An Order

Street:	918 W Illinois St 513
Zip:	61801
Restaurant:	Torticas Grill
Price:	40.00
Ordered:	2014-04-28 14:20:12
Delivered:	2014-04-28 19:40:00

Technical Challenge

Building the GrubHub parser was our technical challenge. In any situation where you have to scrape external data, you need to make assumptions based on what you see and hope that they are the correct assumptions. You have no idea what "rules" are being used to generate that data and how the format could vary between instances. In our particular case, we learned that spaces, tabs, and line breaks are not the same thing even though visually they can appear the same. To a parser, they are totally different and any parsing efforts need to account for the variety of different spacing types that could be used. If the user does not provide the entirety of the email, the parser cannot function and needs to fail cleanly. This requires error checking at nearly every step.

Spec Changes

As the project evolved, we realized that our initial schema did not allow us to store all of the data needed to function. This required many ALTER TABLE statements to bring the database into line with what the code expected.

In our original specifications we had assumed that the administrator would be responsible for adding restaurants. This does not scale well at all and so we made a user-facing page to delegate that responsibility. We were also forced to cut back on minor features within the recommendation engine as the current implementation would have required us to start over to accommodate them.

Almost everything else was as originally planned.

Division of Labor

Rohan Kapoor: Created all of the database schemas and was responsible for integrating everything into a single, cohesive package. Worked with Austin to implement a viable design and build all basic functionality.

Austin Kramer: Worked with Rohan to implement a viable design and built the GrubHub parser. Assisted during all debugging and development efforts. Assembled the demo video from Yuriy's clips.

Everett Hu: Responsible for building the recommendation engine. Helped with gathering the initial scope of the project (ER Diagrams, Schema).

Yuriy Tytla: Responsible for data gathering and demoing the product. He "shot" the demo video and was the voice behind it. Assisted with site design and initial scope of project.