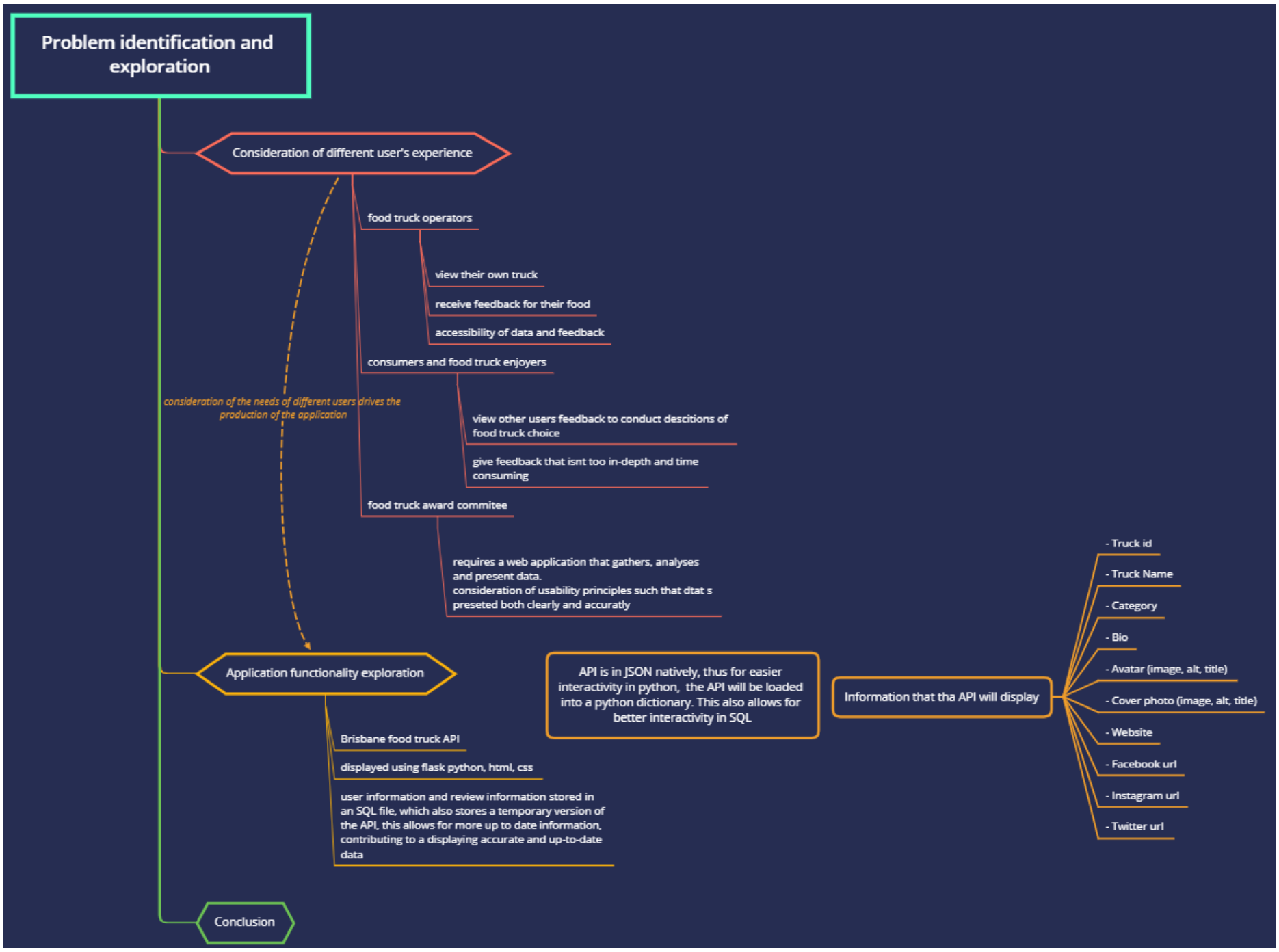
Digital Solutions IA2 – Hunter Kruger-Ilingworth

Problem identification and exploration

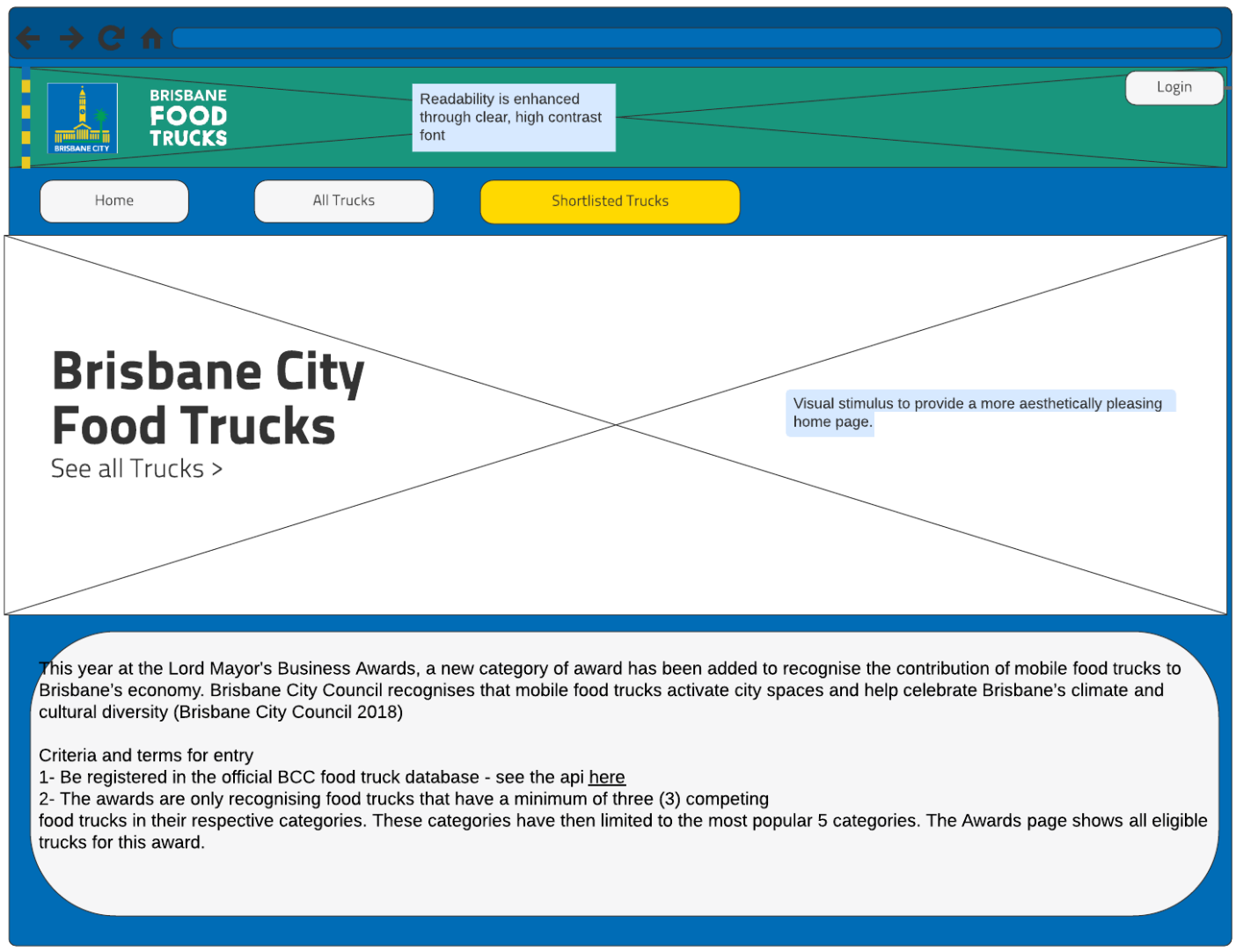
Mind Map



Prescribed criteria	Self determined criteria
1- Generation of digital solution that a satisfies the task description. It must import the food truck API content into an SQL table, and displays this data in a logical and user-friendly manor 2- Allow for Fair and Accurate user reviews	1- Cater to users required functionality, and allow for ease of use 2- Considered incorporation of useability principles a) Utility b) Portability c) responsiveness d) learnability / effectiveness 3- Make considerations for the User experience 4- Accuracy and adaptability of code with regards to a potential change in the API 5- Efficiency of code through optimisation of processing power and internet usage

Annotated GUI Designs

Hunter Kruger-Ilingworth | May 17, 2021



Legal Jargon to ensure legal compliance

Use of existing brisbane food truck logo demonstrates cohesion and legitimises the website as a platform for Showing the official food trucks and their eligibility for awards

The taslk descripttion states users can register for the online polling system with their email address, and supply a display name, password, and choose to receive recommendations (or not). If a user
chooses to receive email recommendations, they may wish to filter by a preferred food truck category (optional, where default is overall positive ratings from all categories)

Users will be able to login or register,
Accounts will not be able to share emails so that repeat votes are less prevailant in polling data, enhancing the effectiveness of the solution

Use of banner image, both on above the menu bar and under the menu bar enhances viewer retention and engagingness through more visual stimulus whilst not being a purposeless distraction.

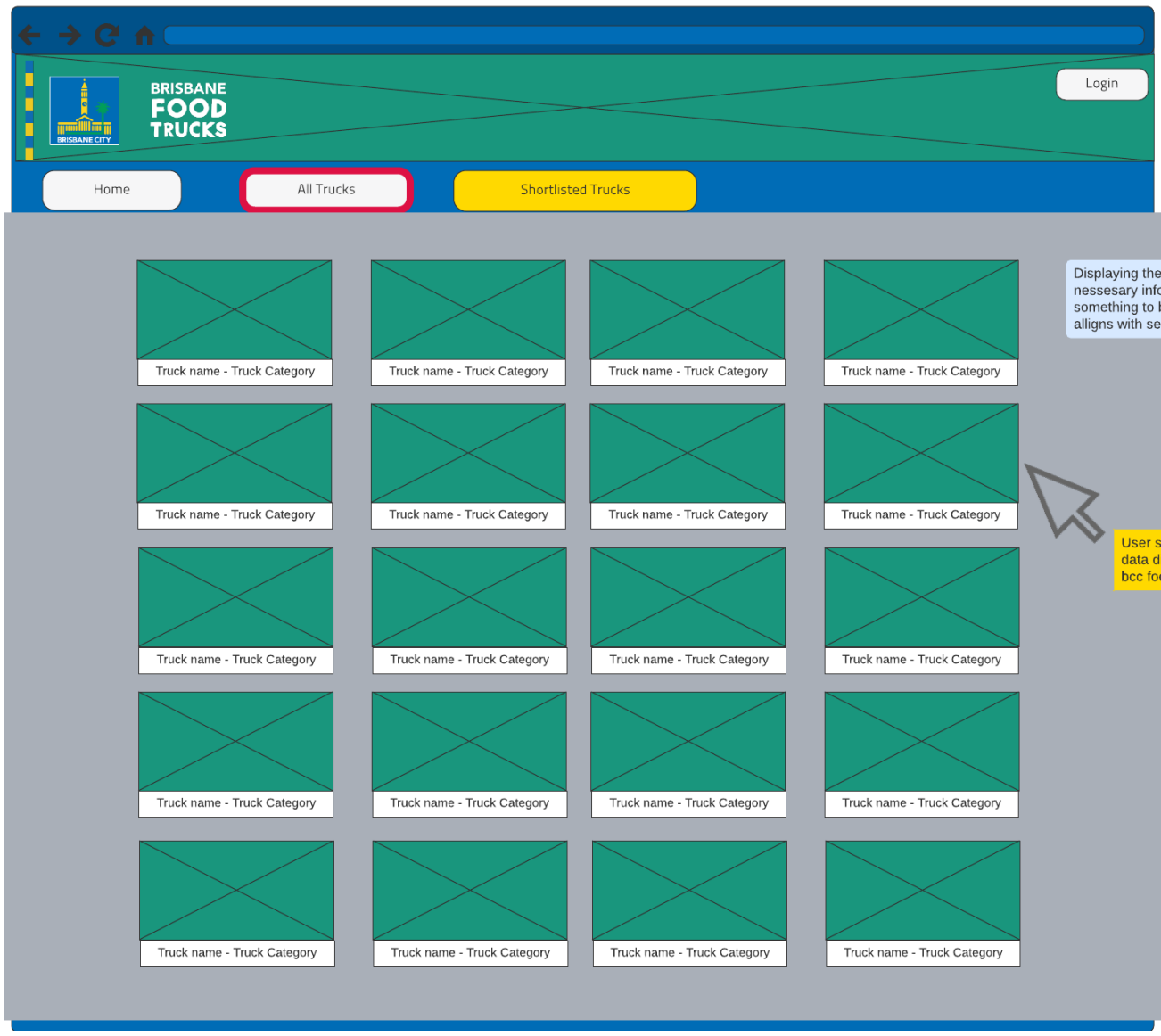
Highlighting the Shortlisted Trucks in yellow will direct the users attention towards it, thus enhancing the user expeirence and overall effectiveness of the website

Simplistic web design limits visual clutter, which fits a more modern style of web design, whilst leaving room for expansion (SPC2) - furthermore the colours that were used were the ones existing in the Brisbane city counel logo which makes it better fit the context of the task at hand and suits the needs of the clientele.

Inclusion of legal information at the bottom of the webpage ensures that the solution is legally compliant

Efficient displaying of data on high contrast colours enhance the effectiency in the conveying of information

text is accessible at the index page of the webiste and informs users of the purpose and functionality of the website and also provides users with the nessesary information



Displaying the truck name and category displays nessesary information, and the cover image provides something to break up the monotony of the text - this alligns with self determined criteria 3 (SD3)

User selects a food truck and is taken to its page, the data displayed on each page comes directly from the bcc food truck API

Food Truck Api - first datapoint

This is the information that can be displayed, to enhance the utility of the web application, it is reccommended that most if not all of this information is utilised

"truck_id":"66",
"name":"King of the Wings",
"category":"American",
"bio":"International award winning Food truck. Australia's first American style chicken wing food truck!! \r\nOffering 5 unique flavours of succulent, flavoursome, crispy wings!! Taste the hype!\r\nRepresented Australia in New York in 2016 for the Worlds Biggest Buffalo Wing Festival, current holder of Best Food Truck Brisbane (Eat Drink Awards 2018)... \r\nThis truck is a must try for wing lovers!!",
"avatar": {
"src":"https://www.bnefoodtrucks.com.au/sites/foodtrucks/files/styles/avatarlarge/public/images/King%20of%20the%20Wings/king_of_the_wings_logo_0.jpg?itok=mvKcHHD6",
"alt":"",
"title":""
},
"cover_photo": {
"src":"https://www.bnefoodtrucks.com.au/sites/foodtrucks/files/styles/header/public/images/King%20of%20the%20Wings/image.jpeg?itok=GDIv3sIa",
"alt":"",
"title":""
},
"website":"",
"facebook_url":"https://www.facebook.com/kingofthewingsbrisbane/",
"instagram_handle":"wingstigram ",
"twitter_handle":"kingofthewings1"

BRISBANE FOOD TRUCKS

Login

Home | All Trucks | Shortlisted Trucks

Food Truck Name

Food truck category

Food truck Facebook, instagram, twitter and website links

Food truck biography (text which describes the food truck and is made by the regitere of the food truck) ----------
---

Leave a Review for (food truck name)

Speed of Service ★ ☆ ☆

Qualty and taste ★ ★ ☆

Value for money ★ ★ ★

quick and reliable star rating system incentivises people to leave a review, providing an *optional* text box to leave a more detailed review enhances the utlility of the website, through more detailed feedback this enhances the usebaility principle of effecitveness. This text box option was made optional to cater for people who have less time to leave a review, enhancing the useability principles of accessibility

(self determined criteria 1 & 2)

User reviews *number of user reviews*

providing the number of reviews gives feedback as to thepopularity of the truck which is a useful metric for adjuducators of the awrds and the truck owners themselves enhanicng utility

Barry Thomas
'I thought it was fairly average in my opinion. The owner was kind and welcoming though!

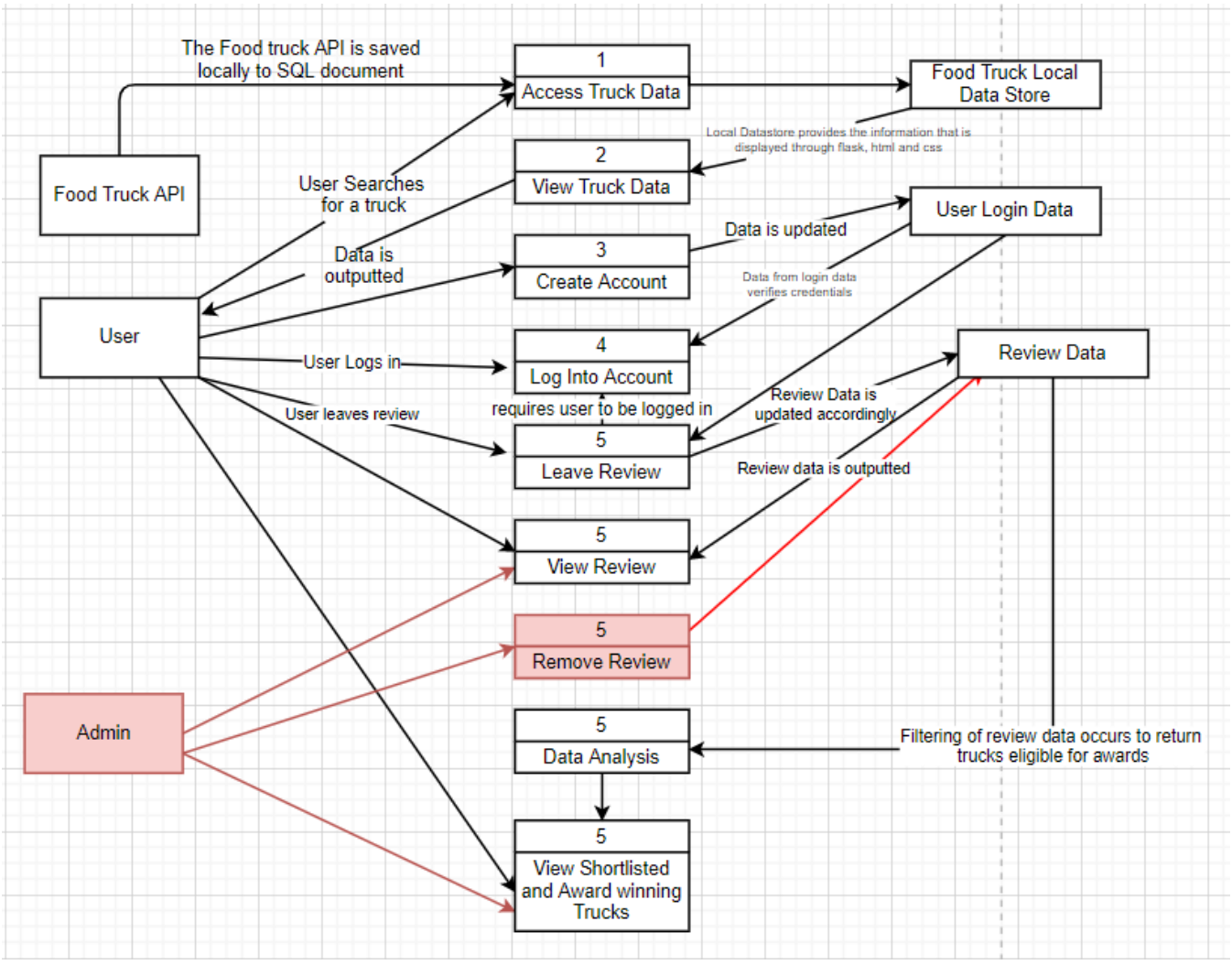
Speed of Service ★ ☆ ☆
Qualty and taste ★ ★ ☆
Value for money ★ ★ ★

Barry Thomas' Brother
I concur with my brothers prior statement

Speed of Service ★ ☆ ☆
Qualty and taste ★ ★ ☆
Value for money ★ ★ ★

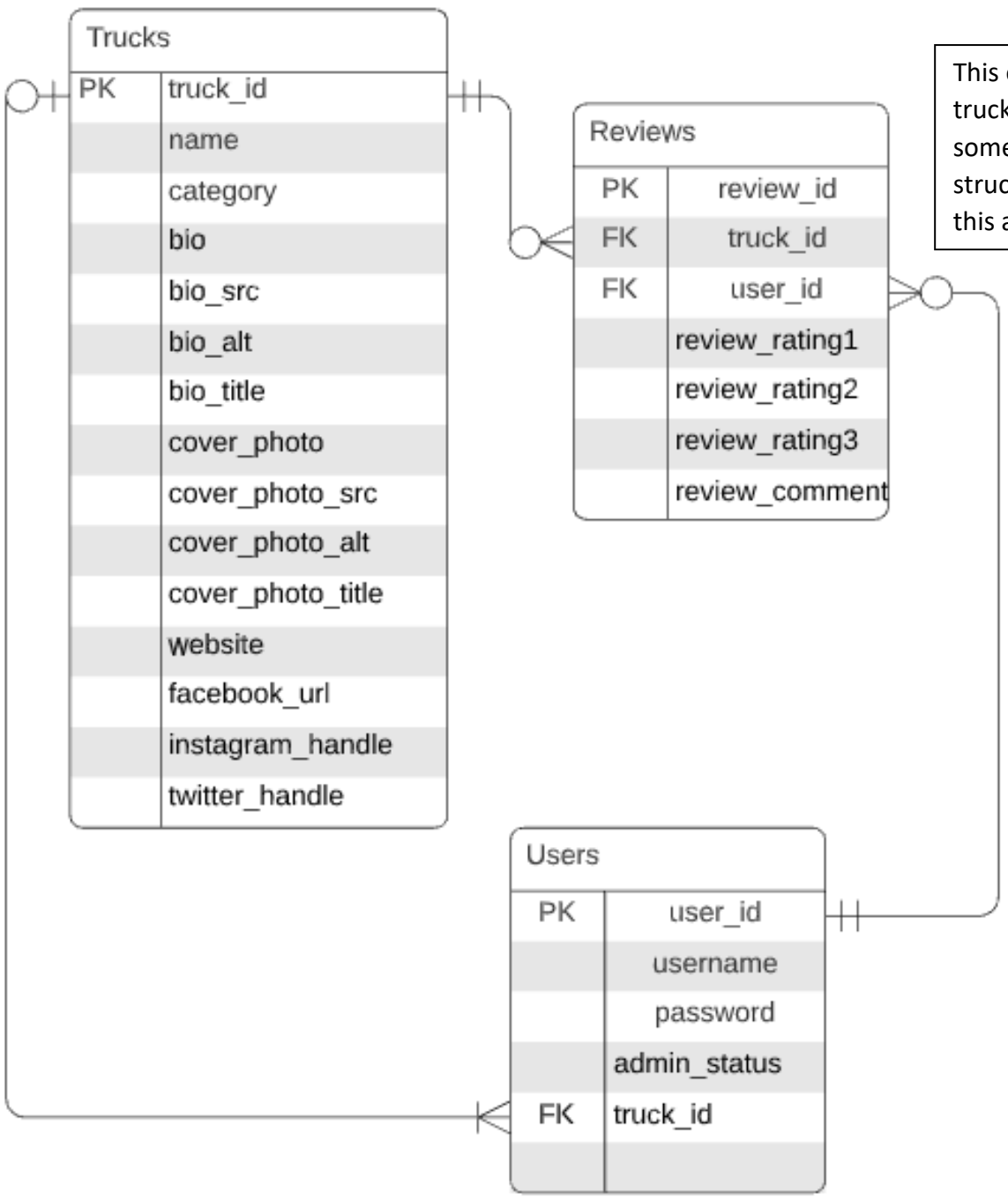
The proviision of reviews allows for a reliable metric through which to base the awards on. The criteria for the awards are as follows

Data Flow Diagram



Components of the DFD in red signify elements which are recommended to be implemented in a future iteration of the digital solution to enhance the overall utility of the application.

Entity relationship diagram



This entity relationship diagram limits the amount of truck a user can have to 1. To allow for cases where someone owns multiple food trucks, the data structure should be re-evaluated to accommodate for this and could be implemented in a future iteration.

Algorithms

Algorithm to import data from API to SQL

SUB **IMPORT_DATA**()

```
BEGIN
      ACCESS food truck api
      CONNECT to SQL trucks table
      FOR truck in food truck api:
            Save truck_id to local SQL
            Save name to local SQL
            Save category to local SQL
            Save bio to local SQL
            Save avatar_src to local SQL
            Save avatar_alt to local SQL
            Save avatar_title to local SQL
            Save cover_photo_src to local SQL
            Save cover_photo_alt to local SQL
            Save cover_photo_title to local SQL
            Save website to local SQL
            Save facebook_url to local SQL
            Save Instagram_handle to local SQL
            Save twitter_handle to local SQL
END
```

Algorithm to update SQL data

```
BEGIN
      IF SQL file exists:
            DELETE existing truck data
            IMPORT_DATA()
      ELSE:
            CREATE truck data table
            IMPORT_DATA()
END
```

Algorithm to register user

```
BEGIN
      truck_id = null #by default
      User input username
      User input password
      User input email
      User input truck_id if applicable #for users who own a truck
      IF email input contains BOTH "@" AND ".":
            validity of email input = TRUE
            user_id = (Select most recent user_id) + 1
            CONNECT to SQL table users
            INSERT INTO users(user_id,truck_id,username, password, email)
      ELSE:
            Return user to registration page+ message "invalid email"
END
```

#While this may not be 100% foolproof, this will be effective in filtering a vast majority of invalid responses. Refinement of this system is strongly recommended in future revisions of the digital solution.
Furthermore, in future solutions verifying that the details entered aren't already in the database should be done, as entering the same people multiple times is inefficient with regards to storage. This may also lead to repeat votes, highly detrimenting the integrity of the voting system

Algorithm to login a user

```
BEGIN
      logged_in = "false"
      User input username
      User input password
      IF username input matches row in user database:
            IF password in that row corresponds to password input:
                  User is verified
                  Return webpage where logged in == "true"
                  #user may now leave reviews
      IF logged_in == "false":
            Return user to registration page+ message "invalid email"

END
```

#in a future revision, two factor authentication can be implemented to enhance the security of the digital solution. Furthermore, encryption of the digital solution could be implemented to further enhance security.

Algorithm to leave a review

Check if user is logged in

Blah blah blah include sql queries

```
BEGIN
#logged_in variable is true or false depending on the outcome of the previous algorithm
      IF logged_in == "true":
            truck_id =
            User input review_rating1#speed of service
            User input review_rating2#quality and taste
            User input review_rating3#value for money
            User input review_comment

            CONNECT to SQL reviews table
            review_id = (Select most recent review's review_id) + 1
            INSERT INTO reviews (review_id, truck_id, user_id, review_rating1, review_rating2,
review_rating3, review_comment)



      ELSE: #IF logged_in == "false"

END
```

Algorithm to Filter Trucks by Award Eligibility

Process the polled data, calculate category scores, and generate results and recommendations

```
BEGIN
      CONNECT to trucks table via SQL
      SELECT Categories column from trucks
      SORT results by the number of trucks under each category
      LIMIT the selection by 5
      #a list of five eligible categories have been selected
      SAVE SQL selection as categories

      For category in categories:
            Return all data from trucks where category = category
            #i.e return all truck data with corresponding category

END
```

Generation of Code

```
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning) #removal of warnings make te
sting clearner

#check if file truck.db exists
path = 'D:\\Desktop Files\\hkrugerili\\Desktop\\everything digital solutions\\---
- FIA2 FOOD TRUCK\\CODE\\truck.db'
if os.path.isfile(path) == False:
  try:
    db = sqlite3.connect('D:\\Desktop Files\\hkrugerili\\Desktop\\everything digital solutions\\
---- FIA2 FOOD TRUCK\\CODE\\truck.db')
    cursor = db.cursor()
    cursor.execute("CREATE TABLE fruck ( `truck_id` INTEGER, `name` TEXT, `category` TEXT, `bio`
 TEXT, `avatar_src` TEXT, `avatar_alt` TEXT, `avatar_title` TEXT, `cover_photo_src` TEXT, `cover
_photo_alt` TEXT, `cover_photo_title` TEXT, `website` TEXT, `facebook_url` TEXT, `instagram_hand
le` TEXT, `twitter_handle` TEXT )")
    cursor.execute("CREATE TABLE reviews ( `review_id` INTEGER, `truck_id` INTEGER, `user_id` IN
TEGER, `review_rating1` INTEGER, `review_rating2` INTEGER, `review_rating3` INTEGER, `review_com
ment` TEXT )")
    insertreview = """INSERT INTO reviews (review_id, truck_id, user_id, review_rating1, review_
rating2, review_rating3, review_comment)
    VALUES (?, ?, ?, ?, ?, ?, ?)"""
    cursor.execute(insertreview,(test, test, test, test, test, test, poop))
    db.commit()


  except Exception as error:
    db.rollback()
    print("!!!!!!!!!!!!!!! Something went wrong in the creation of the database: " + str(error))
  finally:
    db.close()
else: #This file does indeed exist, however the contents of the table must be removed to assure
that it is as up to date as possible
  os.remove(path)
  try:
    db = sqlite3.connect('D:\\Desktop Files\\hkrugerili\\Desktop\\everything digital solutions\\
---- FIA2 FOOD TRUCK\\CODE\\truck.db')
    cursor = db.cursor()
    cursor.execute("CREATE TABLE fruck ( `truck_id` INTEGER, `name` TEXT, `category` TEXT, `bio`
 TEXT, `avatar_src` TEXT, `avatar_alt` TEXT, `avatar_title` TEXT, `cover_photo_src` TEXT, `cover
_photo_alt` TEXT, `cover_photo_title` TEXT, `website` TEXT, `facebook_url` TEXT, `instagram_hand
le` TEXT, `twitter_handle` TEXT )")
    cursor.execute("CREATE TABLE reviews ( `review_id` INTEGER, `truck_id` INTEGER, `user_id` IN
TEGER, `review_rating1` INTEGER, `review_rating2` INTEGER, `review_rating3` INTEGER, `review_com
ment` TEXT )")
    cursor.execute("CREATE TABLE login ( `user_id` INTEGER, `truck_id` INTEGER, `username` TEXT,
 `password` TEXT )")
    #some initial test data must be inserted to allow for future code to work


    insertreview = '''INSERT INTO reviews (review_id, truck_id, user_id, review_rating1, review_
rating2, review_rating3, review_comment)
                      VALUES (?, ?, ?, ?, ?, ?, ?)'''
    cursor.execute(insertreview,(0,"null", "null", "null", "null", "null", "null"))


    insertlogin = '''INSERT INTO login (user_id, truck_id, username, password)
                      VALUES (?, ?, ?, ?)'''
    cursor.execute(insertlogin,(0,0, "test", "123"))
    db.commit()
  except Exception as error:
    db.rollback()
    print("!!!!!!!!!!!!!!! Something went wrong in the creation of the database: " + str(error))
  finally:
```

```
    db.close()

#The following code gets the API and converts it into a dictionatry, this dictionary is then
systematically sorted into the SQL file which can be then used for displaying and interactions
with the user.


answer = requests.get("https://www.bnefoodtrucks.com.au/api/1/trucks", verify=False)
#convert to Python dictionary:
trucks = json.loads(answer.text)

#this assumes that a table is there witht he neccessary colums, as such, a new table will be cre
ated before this and it will be EMPTY woo

insertionSQLstring = '''INSERT INTO fruck (truck_id, name, category, bio, avatar_src, avatar_alt
, avatar_title, cover_photo_src, cover_photo_alt, cover_photo_title, website, facebook_url, inst
agram_handle, twitter_handle)
                      VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)'''

try:
  n=0
  for x in trucks:
    for y in x:
      #looping through truck_id, avatar, social media and all that stuff KEYS VALUES AND ALL
      db = sqlite3.connect('D:\\Desktop Files\\hkrugerili\\Desktop\\everything digital solutions
\\---- FIA2 FOOD TRUCK\\CODE\\truck.db')
      cursor = db.cursor()
      #set up parameters for INSERT:
      if str(y) == "truck_id":##################
        truck_id = x[y]




        review_num = random.randint(3, 10)
#####GENERATES A RANDOM NUMBER OD REVIEWS FOR DEMONSTRATIONAL PURPOSES
        z = 0
        while z < review_num:
          try:

            insertreview = '''INSERT INTO reviews (review_id, truck_id, user_id, review_rating1,
 review_rating2, review_rating3, review_comment)
                        VALUES (?, ?, ?, ?, ?, ?, ?)'''
            db2 = sqlite3.connect('D:\\Desktop Files\\hkrugerili\\Desktop\\everything digital so
lutions\\---- FIA2 FOOD TRUCK\\CODE\\truck.db')

            review_cursor = db2.cursor()
            review_id = review_cursor.execute("SELECT review_id FROM reviews ORDER BY review_id
DESC").fetchone()
            review_id = int(review_id[0]) + 1

            truck_id = x[y]
            user_id = "bot"

            review_rating1 = random.randint(-1,1)
            review_rating2 = random.randint(-1,1)
            review_rating3 = random.randint(-1,1)

            sum_review = review_rating1+review_rating2+review_rating3
            if (sum_review/3) >= 0:
              review_comment = "my experience was positive overall"
            else:
              review_comment = "I believe there are better options out there to be honest"

            review_cursor.execute(insertreview,(review_id, truck_id, user_id, review_rating1, re
view_rating2, review_rating3, review_comment))
            db2.commit()
```

```
            z = z+1
          except Exception as error:
            db2.rollback()
            print(str(error))
            break
          finally:
            db2.close()


      elif str(y) == "name":###################
        name = x[y]
      elif str(y) == "category":#################
        category = x[y]
      elif str(y) == "bio":######################
        bio = x[y]
      elif str(y) == "avatar":##########################
        for z in x[y]:
          if str(z) == "src":
            avatar_src = x[y][z]
          elif str(z) == "alt":
            avatar_alt = x[y][z]
          elif str(z) == "title":
            avatar_title = x[y][z]
      elif str(y) == "cover_photo": ##################
        for z in x[y]:
          if str(z) == "src":
            cover_photo_src = x[y][z]
          elif str(z) == "alt":
            cover_photo_alt = x[y][z]
          elif str(z) == "title":
            cover_photo_title = x[y][z]
      elif str(y) == "website":######################
        if x[y] is None:
          website == "null"
        else:
          website = x[y]

      elif str(y) == "facebook_url":################
        facebook_url= x[y]
      elif str(y) == "instagram_handle":################
        instagram_handle = x[y]
      elif str(y) == "twitter_handle":#################
        twitter_handle= x[y]
        cursor.execute(insertionSQLstring,(truck_id, name, category, bio, avatar_src, avatar_alt
, avatar_title, cover_photo_src, cover_photo_alt, cover_photo_title, website, facebook_url, inst
agram_handle, twitter_handle))
      db.commit()
except Exception as error:
  db.rollback()
  print( str(error))

finally:
  db.close()

print("SQL processes successful")


logged = "false" #user is not logged in, and therefore has slightly limited functionality
app = Flask(__name__)
app.secret_key = "session" #session variable for the website

@app.route("/")
def main():
  try:
      db = sqlite3.connect('D:\\Desktop Files\\hkrugerili\\Desktop\\everything digital solutions
\\---- FIA2 FOOD TRUCK\\CODE\\truck.db')
      db.row_factory = sqlite3.Row
      cursor = db.cursor()
```

```
      database = cursor.execute("SELECT * FROM fruck").fetchall()
      categories = cursor.execute("SELECT DISTINCT category FROM fruck").fetchall()
      db.commit()
      print(logged)
  except Exception as error:
    db.rollback()
    print( str(error))
  finally:
    db.close()
  return render_template("index.html", database = database, categories = categories, login_statu
s = logged)



@app.route("/awards")
def award():
  try:
      db = sqlite3.connect('D:\\Desktop Files\\hkrugerili\\Desktop\\everything digital solutions
\\---- FIA2 FOOD TRUCK\\CODE\\truck.db')
      db.row_factory = sqlite3.Row
      cursor = db.cursor()

      eligible = cursor.execute("SELECT category from fruck GROUP BY category ORDER BY count(cat
egory) DESC LIMIT 5").fetchall()
      n=0
      for x in eligible:
        n=n+1
        print("Woo Hoo Number "+ str(n))
        if n == 1:
          cate1 = x[0]
          print(cate1)
        elif n == 2:
          cate2 = x[0]
          print(cate2)
        elif n == 3:
          cate3 = x[0]
          print(cate3)
        elif n == 4:
          cate4 = x[0]
          print(cate4)
        elif n == 5:
          cate5 = x[0]
          print(cate5)

      cat_1 = cursor.execute("SELECT * FROM fruck WHERE category ==?", (cate1,)).fetchall()
      cat_2 = cursor.execute("SELECT * FROM fruck WHERE category ==?", (cate2,)).fetchall()
      cat_3 = cursor.execute("SELECT * FROM fruck WHERE category ==?", (cate3,)).fetchall()
      cat_4 = cursor.execute("SELECT * FROM fruck WHERE category ==?", (cate4,)).fetchall()
      cat_5 = cursor.execute("SELECT * FROM fruck WHERE category ==?", (cate5,)).fetchall()



      db.commit()
      print(logged)
      print("Hello world")



  except Exception as error:
    db.rollback()
    print( str(error))
  finally:
    db.close()
    return render_template("awards.html", login_status = logged, cat1=cat_1, cat2=cat_2, cat3=ca
t_3, cat4=cat_4, cat5=cat_5)
```

```python
@app.route("/login")
def login():
  return render_template("login.html")

@app.route("/login-verify", methods=['GET', 'POST'])
def verify():
  try:

    u_input = request.form["username"]
    p_input = request.form["password"]

    db = sqlite3.connect('D:\\Desktop Files\\hkrugerili\\Desktop\\everything digital solutions\\
---- FIA2 FOOD TRUCK\\CODE\\truck.db')
    db.row_factory = sqlite3.Row
    cursor = db.cursor()
    login_SQL = cursor.execute("SELECT * FROM login").fetchall()
    db.commit()
    global logged
    for entry in login_SQL:
      if u_input == entry[2] and p_input == entry[3]:
        print("valid details entered")
        logged = "true"
        print(logged)
        return redirect("/") #joe
      else:
        return render_template("login.html")
    #####MAKE SURE THAT YOU CANT HAVE THE SAME USERNAME AS SOMEONE ESLE
    db.commit()
  except Exception as error:
    db.rollback()
    print( str(error))
  finally:
    db.close()


@app.route("/register")
def reg():
  return render_template("register.html")

@app.route("/register-verify", methods=['GET', 'POST'])
def regv():

  try:

    u_input = request.form["username"]
    p_input = request.form["password"]
    e_input = request.form["email"]
    print(e_input)
    if e_input.find('@')!=-1 and e_input.find('.')!=-1:
      print("valid email!")
      db = sqlite3.connect('D:\\Desktop Files\\hkrugerili\\Desktop\\everything digital solutions
\\---- FIA2 FOOD TRUCK\\CODE\\truck.db')
      db.row_factory = sqlite3.Row
      cursor = db.cursor()

      insertlogin = '''INSERT INTO login (user_id, truck_id, username, password)
                       VALUES (?, ?, ?, ?)'''
      user_id = cursor.execute("SELECT user_id FROM login ORDER BY user_id DESC").fetchone()
      user_id = int(user_id[0]) + 1
      cursor.execute(insertlogin,(user_id, "null", u_input, p_input))
      db.commit()
      global logged
      logged = "true"
      return redirect("/")
```

```
    else:
      return render_template("register.html")
  except Exception as error:
    db.rollback()
    print( str(error))
  finally:
    db.close()
  return render_template("register.html")


@app.route("/filter/<category>")
def filter(category):
  try:
      db = sqlite3.connect('D:\\Desktop Files\\hkrugerili\\Desktop\\everything digital solutions
\\---- FIA2 FOOD TRUCK\\CODE\\truck.db')
      db.row_factory = sqlite3.Row
      cursor = db.cursor()
      filtered = cursor.execute("SELECT * FROM fruck WHERE category == ?", (category,)).fetchall
()
      categories = cursor.execute("SELECT DISTINCT category FROM fruck").fetchall()

  except Exception as error:
    db.rollback()
    print( str(error))
  finally:
    db.close()
  return render_template("index.html", database = filtered, categories = categories)


@app.route("/trucks/<truckID>")
def truckInfo(truckID):
  try:
    tid = truckID
    db = sqlite3.connect('D:\\Desktop Files\\hkrugerili\\Desktop\\everything digital solutions\\
---- FIA2 FOOD TRUCK\\CODE\\truck.db')
    db.row_factory = sqlite3.Row
    cursor = db.cursor()
    database = cursor.execute("SELECT * FROM fruck WHERE truck_id == ? ", (truckID,)).fetchall()
    review = cursor.execute("SELECT * FROM reviews WHERE truck_id == ? ", (truckID,)).fetchall()

  except Exception as error:
    db.rollback()
    print("!!!!!!!!!!!!!!! Something went wrong in def main thing sql game" + str(error))
  finally:
    db.close()
  return render_template("trucks.html", database = database, review = review, tid=tid, login_sta
tus = logged)


@app.route("/trucks/<truckID>/review_submit", methods=['GET', 'POST'])
def truckreview(truckID):
    truck_id = truckID
    user_id = "test"
    review_rating1 = request.form["radio1"]
    review_rating2 = request.form["radio2"]
    review_rating3 = request.form["radio3"]
    review_comment = request.form["comment"]
    try:

      db = sqlite3.connect('D:\\Desktop Files\\hkrugerili\\Desktop\\everything digital solutions
\\---- FIA2 FOOD TRUCK\\CODE\\truck.db')
      db.row_factory = sqlite3.Row
      cursor = db.cursor()

      insertreview = '''INSERT INTO reviews (review_id, truck_id, user_id, review_rating1, revie
w_rating2, review_rating3, review_comment)
```

```
                        VALUES (?, ?, ?, ?, ?, ?, ?)'''
      review_id = cursor.execute("SELECT review_id FROM reviews ORDER BY review_id DESC").fetcho
ne()
      review_id = int(review_id[0]) + 1
      cursor.execute(insertreview,(review_id, truck_id, user_id, review_rating1, review_rating2,
 review_rating3, review_comment))

      db.commit()
    except Exception as error:
      db.rollback()
      print( str(error))
    finally:
      db.close()
    url = "/trucks/" + str(truckID)
    return redirect(url)

@app.route("/info")
def info():
  return render_template("info.html")
app.run(debug=True)
```

Evaluation and testing of the code

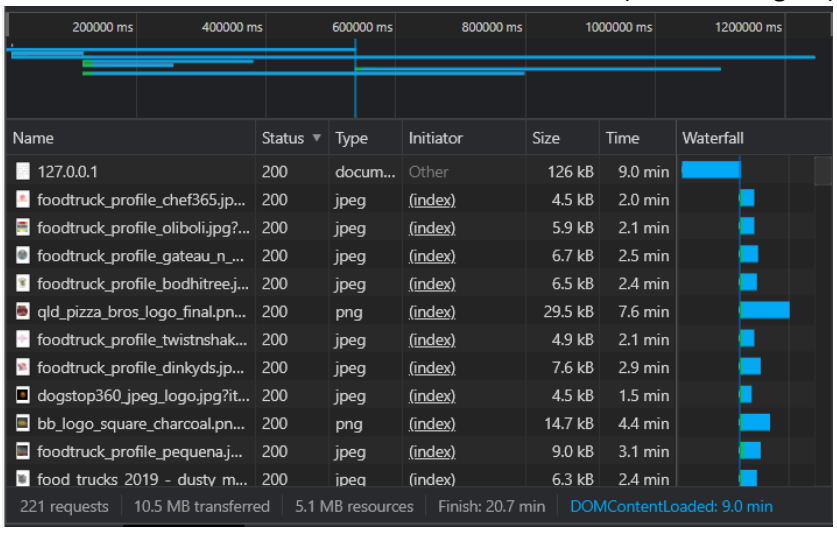
Synthesising and evaluating [9–10] Critical evaluation of impacts and coded components and the digital solution against essential prescribed and self-determined criteria to make discerning refinements and astute recommendations justified by data

Prescribed criteria	Evaluation and recommendations and refinements (justified by data where applicable)
1- Generation of digital solution that a satisfies the task description. It must import the food truck API content into an SQL table, and displays this data in a logical and user-friendly manor	This criterion has been satisfied through - The importing of the API into a local SQL database - The displaying of this data In a logical and user-friendly manour, which is achieved through the consideration of useability principles, see self determined criterion
2- Allow for Fair and Accurate user reviews	Fair and accurate user reviews are ensured through making sure that repeat reviews are disallowed ensure food truck review data reflects the truth. Accuracy was ensured through the 3-scale rating system where users pick between -1, 0 and 1 as outlined in the task description. Accuracy was enhanced through implementation of an optional textbox where users can enter in information relevant for constructive criticism and their overall experience

Self Determined criteria	Evaluation and recommendations and refinements (justified by data where applicable)
1- Cater to users 'required functionality, and allow for ease of use	Three hypothetical users were **<u>outlined in the problem exploration:</u>** Walter • Food truck owner, Walter. He requires customer feedback, accessible even through limited computer experience • Customer, Shelly. She requires a review system which is fast to leave feedback • Food truck awards committee chairwoman, Naomi. She too, requires review data on all trucks in an easy to access format Components of the code were made to provide for the needs of these potential clientele. For Walter, customer feedback was integrated into the digital solution through a review system – whereby registered users can review any truck in the database. For Shelly, the review system was made to suit a quick 'tap and go' mentality. This was to incentivise more reviews, as lengthy mandatory explanations of a customers experience would otherwise be a disincentive. A chord has been struck to suit both clientele, as more in-depth constructive feedback is optional through a textbox that can be filled out by a user who may have spare time. However, since it is not mandatory, people like shelly who value a quicker review experience will also enjoy this system. Therefore, it is recommended that this system is maintained for future iterations of the digital solution. To cater for Naomi's needs, review data was made accessible for users via the truck page, however, Naomi requires large scale comparisons between trucks to conduct her work, therefore it is strongly recommended that a future version of the digital solution include a truck overview page, where trucks can be compared on a larger scale by rating, review count and other metrics which can be useful and would greatly enhance the user experience of both truck owners, and award adjudicators through enhanced utility.
2- Considered incorporation of useability principles e) Utility f) Portability g) responsiveness	Useability principles were vital in guiding the web design for the digital solution and were explained at length in the annotated GUI designs Examples of enhancing utility is shown through the full incorporation and displaying of data from the bcc food trucks API, and the incorporation of a reviews which are conducted by visitors to the website. It is recommended that to further enhance the useability principle of utility to th user, that a back-end administrator system is implemented, similar to the one illustrated in the data flow diagram whereby inappropriate or otherwise unfair reviews can be removed by website administrators, and other statistics are made viewable to allow for comparisons and large-scale data management.

h) learnability / effectiveness	Portability was seldom considered Data suggests that 70% of web traffic comes from phones, therefore it is strongly recommended the digital solution includes a mobile version to allow for people to review on the go, this would increase accessibility and overall be of great benefit to the digital solution through more collection of valuable review data. https://techjury.net/blog/what-percentage-of-internet-traffic-is-mobile/#:~:text=Up%20to%2070%20percent%20of%20web%20traffic%20comes%20from%20mobile%20devices. Responsiveness was considered through the highlight of navigation buttons upon mouse hover, this enhances the user experience by giving more tactile feedback to user inputs. Learnability and effectiveness were considered through the minimalist approach to web design that was taken for this digital solution. As this becoming the standard, user will find familiarity and ease of use within this digital solution. Components which were incorporated which follow web design standards include: Login in the top right, logos in the top left, navigation at the top of the screen and accessible through all pages of the website and legal information as a footer on each page and hyperlinked banners which take to the respective truck pages. It is recommended that useability principles are considered when incorporating new features into the digital solution to enhance the user experience.
3- Make considerations for the User experience	Considerations have been made for the user experience as discussed previously. The minimalist approach to the web design decreases visual clutter, which enhances the useability of the website. Learnability was incorporated as well to enhance the user experience through a higher innate understanding of the website through familiarity.
4- Accuracy and adaptability of code	Accuracy and adaptability of the code was ensured through the implementation of the data updating algorithm whereby an SQL document is created to populate the latest instance of the API for processing through python, and thus displaying through HTML and CSS. This coded component is accurate, as it successfully achieves this task, and no information is lost or otherwise convoluted It is recommended this algorithm is maintained throughout future iterations of this digital solution as the algorithm ensures adaptability through constant updating of the SQL table each time the program is run. Adaptability can be enhanced through alteration of the SQL statement which returns the content for the awards page: `SELECT category from fruck GROUP BY category ORDER BY count(category) DESC LIMIT 5` While at the time of writing, this outputs the same thing as outlined in the task description, however the task description specifies that trucks with less than three trucks in a category should be filtered out. The current code does not account for a change in the database whereby many trucks are reduced and leaves categories that conflict with these parameters. As such in future iterations, it is recommended that the SQL statement which returns the awards page content is altered such that categories with less than 3 trucks are excluded entirely.
5- Efficiency of code through optimisation of processing power and internet usage	Efficiency of code needs to be optimised to provide the fastest experience for users. While considerations have been made for optimisation of code, visual stimulus in the form of image have been incorporated on a large scale from a large scale, this is particularly evident on the home page, where every truck's avatar and banner image is displayed. This would mean a very fast internet connection would be necessary for using the digital solution efficiently. It is highly recommended that optimisations are made, this can be done by downscaling the resolution of the images loaded which would reduce the network load considerable. This recommendation is highly justified, as data gathered from chrome network throttling showed that not only was most of the time loading dedicated to the images alone, it also showed that with a slow internet connection of 56 kb/s, comparable to dial-up speed the

website took 20.7 minutes in total to load all assets (see below figure).



This extreme wait time shows that an imageless version of the application should be available for users who do not have fast internet – because it has been shown from the above figure that practically all the network load and wait time is because it is loading images.