Exploring the problem gives opportunities to start addressing some of the first *ISMG Criterion: Retrieving and comprehending*, especially:

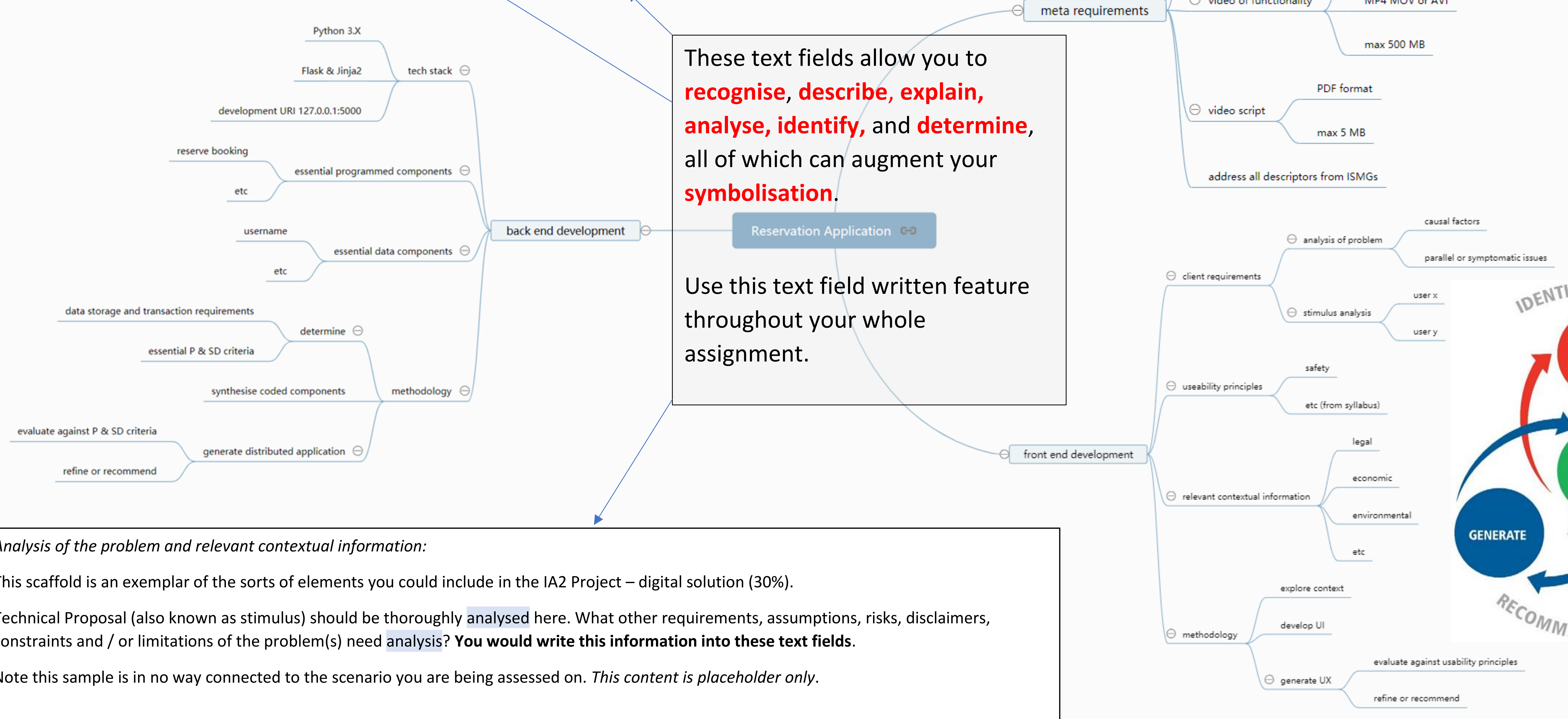
- Recognition and description of:
  - **Programming elements**
  - **UI components**
  - **Useability principles**
- Symbolisation and explanation of:
  - **programming information and ideas**
  - **interrelationships between UX and data**

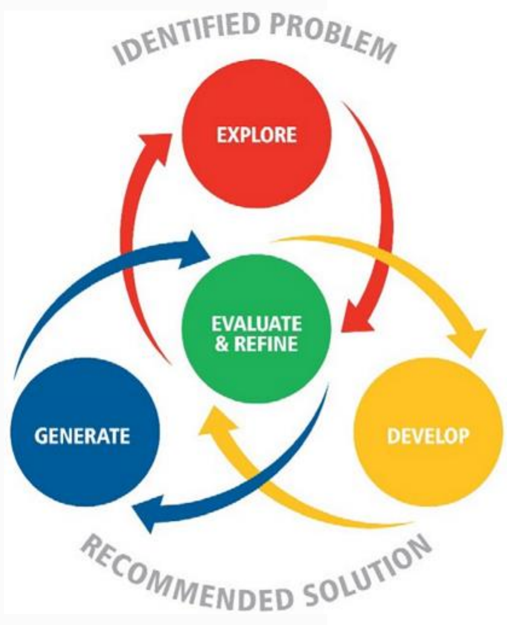
Opportunities also exist here for evidencing the second *ISMG Criterion: Analysing*, especially:

- Analysis of problem and contextual information to identify:
  - **UI elements and features**
  - **Data**
  - **Programmed components**
  - **How these inter-relate**
- Determination of:
  - **Solution requirements**

These text fields allow you to **recognise**, **describe**, **explain**, **analyse**, **identify**, and **determine**, all of which can augment your **symbolisation**.

Use this text field written feature throughout your whole assignment.





Analysis of the problem and relevant contextual information:

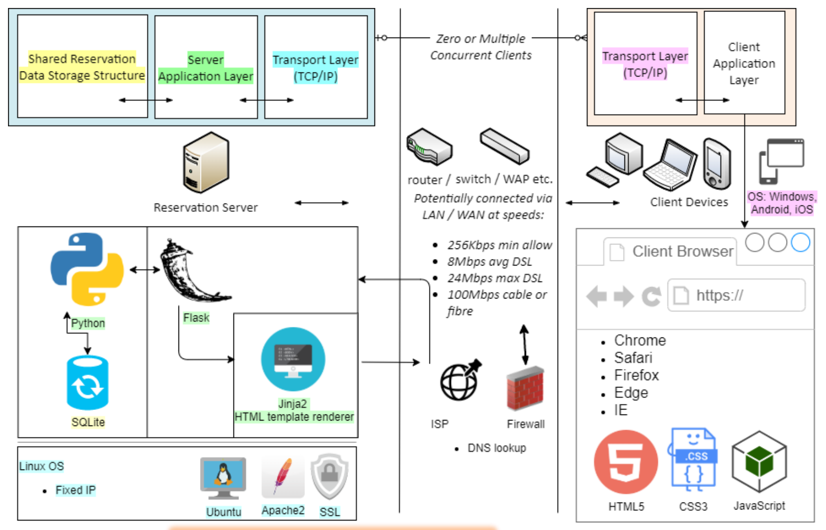
This scaffold is an exemplar of the sorts of elements you could include in the IA2 Project – digital solution (30%).

Technical Proposal (also known as stimulus) should be thoroughly analysed here. What other requirements, assumptions, risks, disclaimers, constraints and / or limitations of the problem(s) need analysis? **You would write this information into these text fields**.

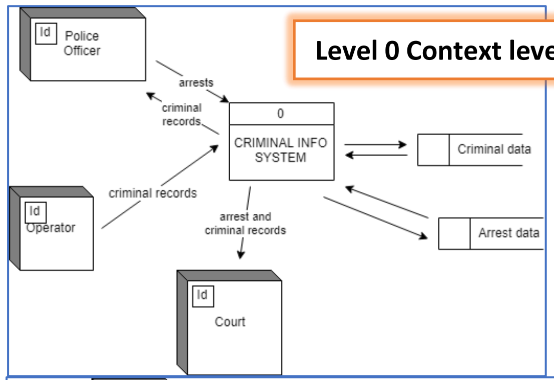
Note this sample is in no way connected to the scenario you are being assessed on. *This content is placeholder only*.

Continue your recognition, description, symbolisation, and explanation throughout the entire document using these text fields.

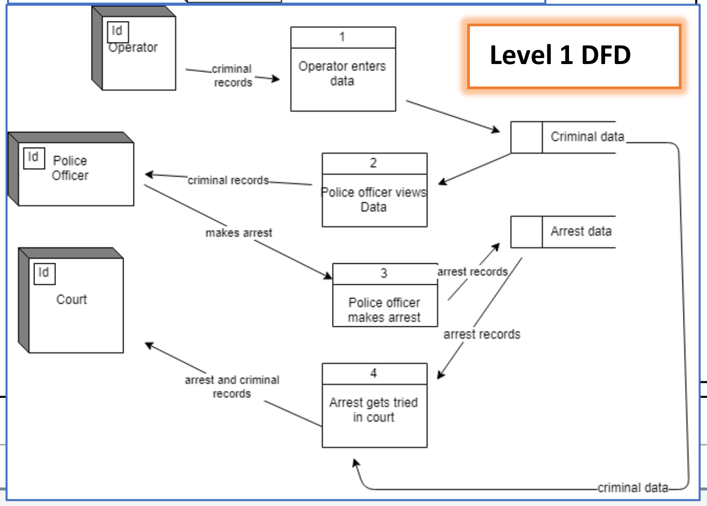
It is recommended here that you really focus on an astute determination of essential **prescribed** and **self-determined** criteria. Include technical criteria such as normalization, scalability, algorithmic complexity etc.



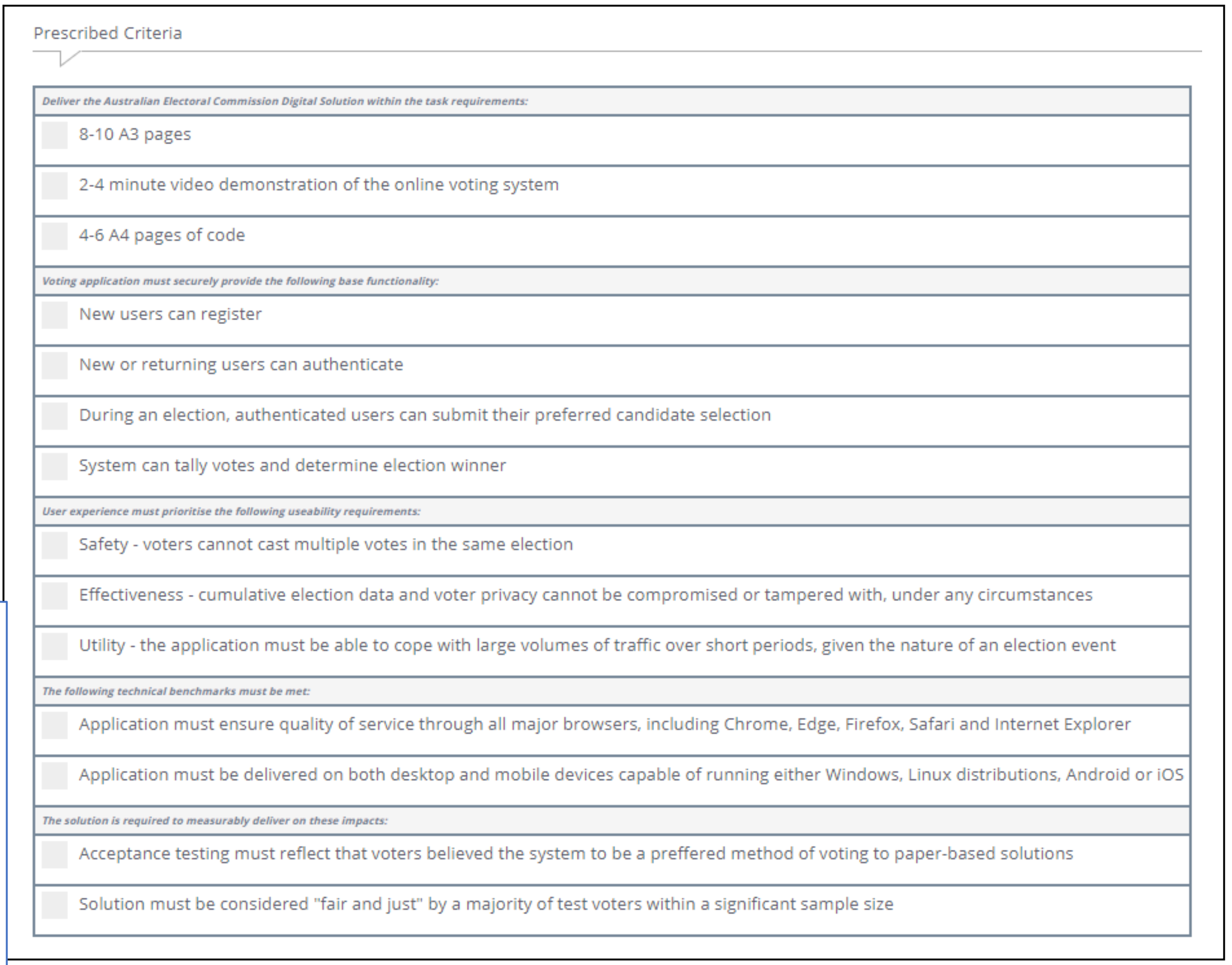
System Architecture



Level 0 Context level DFD



Level 1 DFD



Prescribed Criteria

Deliver the Australian Electoral Commission Digital Solution within the task requirements:

☐ 8-10 A3 pages

☐ 2-4 minute video demonstration of the online voting system

☐ 4-6 A4 pages of code

Voting application must securely provide the following base functionality:

☐ New users can register

☐ New or returning users can authenticate

☐ During an election, authenticated users can submit their preferred candidate selection

☐ System can tally votes and determine election winner

User experience must prioritise the following useability requirements:

☐ Safety - voters cannot cast multiple votes in the same election

☐ Effectiveness - cumulative election data and voter privacy cannot be compromised or tampered with, under any circumstances

☐ Utility - the application must be able to cope with large volumes of traffic over short periods, given the nature of an election event

The following technical benchmarks must be met:

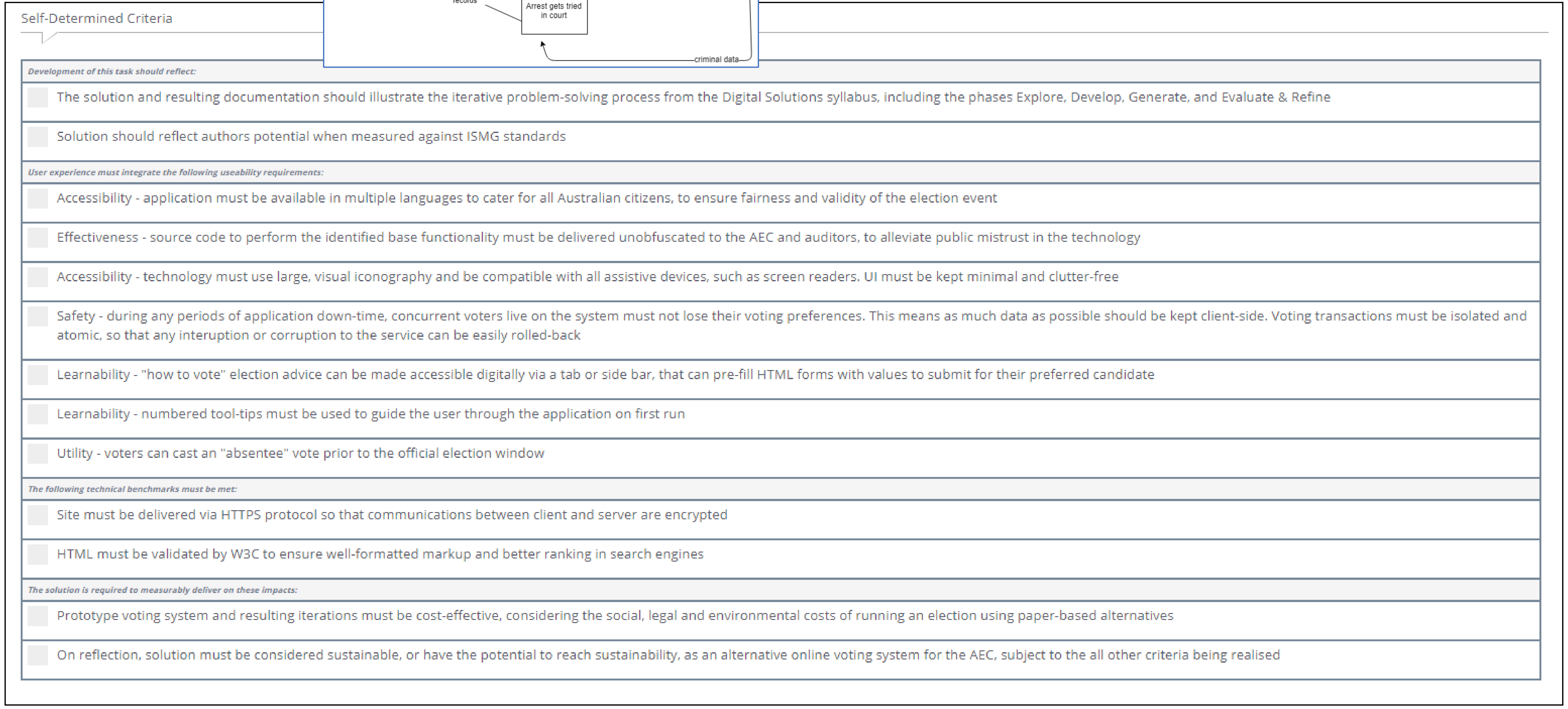
☐ Application must ensure quality of service through all major browsers, including Chrome, Edge, Firefox, Safari and Internet Explorer

☐ Application must be delivered on both desktop and mobile devices capable of running either Windows, Linux distributions, Android or iOS

The solution is required to measurably deliver on these impacts:

☐ Acceptance testing must reflect that voters believed the system to be a preffered method of voting to paper-based solutions

☐ Solution must be considered "fair and just" by a majority of test voters within a significant sample size



Self-Determined Criteria

Development of this task should reflect:

☐ The solution and resulting documentation should illustrate the iterative problem-solving process from the Digital Solutions syllabus, including the phases Explore, Develop, Generate, and Evaluate & Refine

☐ Solution should reflect authors potential when measured against ISMG standards

User experience must integrate the following useability requirements:

☐ Accessibility - application must be available in multiple languages to cater for all Australian citizens, to ensure fairness and validity of the election event

☐ Effectiveness - source code to perform the identified base functionality must be delivered unobfuscated to the AEC and auditors, to alleviate public mistrust in the technology

☐ Accessibility - technology must use large, visual iconography and be compatible with all assistive devices, such as screen readers. UI must be kept minimal and clutter-free

☐ Safety - during any periods of application down-time, concurrent voters live on the system must not lose their voting preferences. This means as much data as possible should be kept client-side. Voting transactions must be isolated and atomic, so that any interuption or corruption to the service can be easily rolled-back

☐ Learnability - "how to vote" election advice can be made accessible digitally via a tab or side bar, that can pre-fill HTML forms with values to submit for their preferred candidate

☐ Learnability - numbered tool-tips must be used to guide the user through the application on first run

☐ Utility - voters can cast an "absentee" vote prior to the official election window

The following technical benchmarks must be met:

☐ Site must be delivered via HTTPS protocol so that communications between client and server are encrypted

☐ HTML must be validated by W3C to ensure well-formatted markup and better ranking in search engines

The solution is required to measurably deliver on these impacts:

☐ Prototype voting system and resulting iterations must be cost-effective, considering the social, legal and environmental costs of running an election using paper-based alternatives

☐ On reflection, solution must be considered sustainable, or have the potential to reach sustainability, as an alternative online voting system for the AEC, subject to the all other criteria being realised

Accessibility – site to be coded using *HTML Semantic Elements* for ease of screen-reader or assistive device recognition:

```
<header>
<nav>  <section>  <aside>
                  <article>
<footer>
```

Source: https://www.w3schools.com/html/html5_semantic_elements.asp

Safety – Breadcrumb trail illustrates site depth and enables users to recover by navigating their way to a previously visited higher level resource.

Utility – sort results by list view (default) or grid (tile) view, like mobile. Display results alphabetically or chronologically. This functionality enables the user to navigate lost items if a search cannot be found.

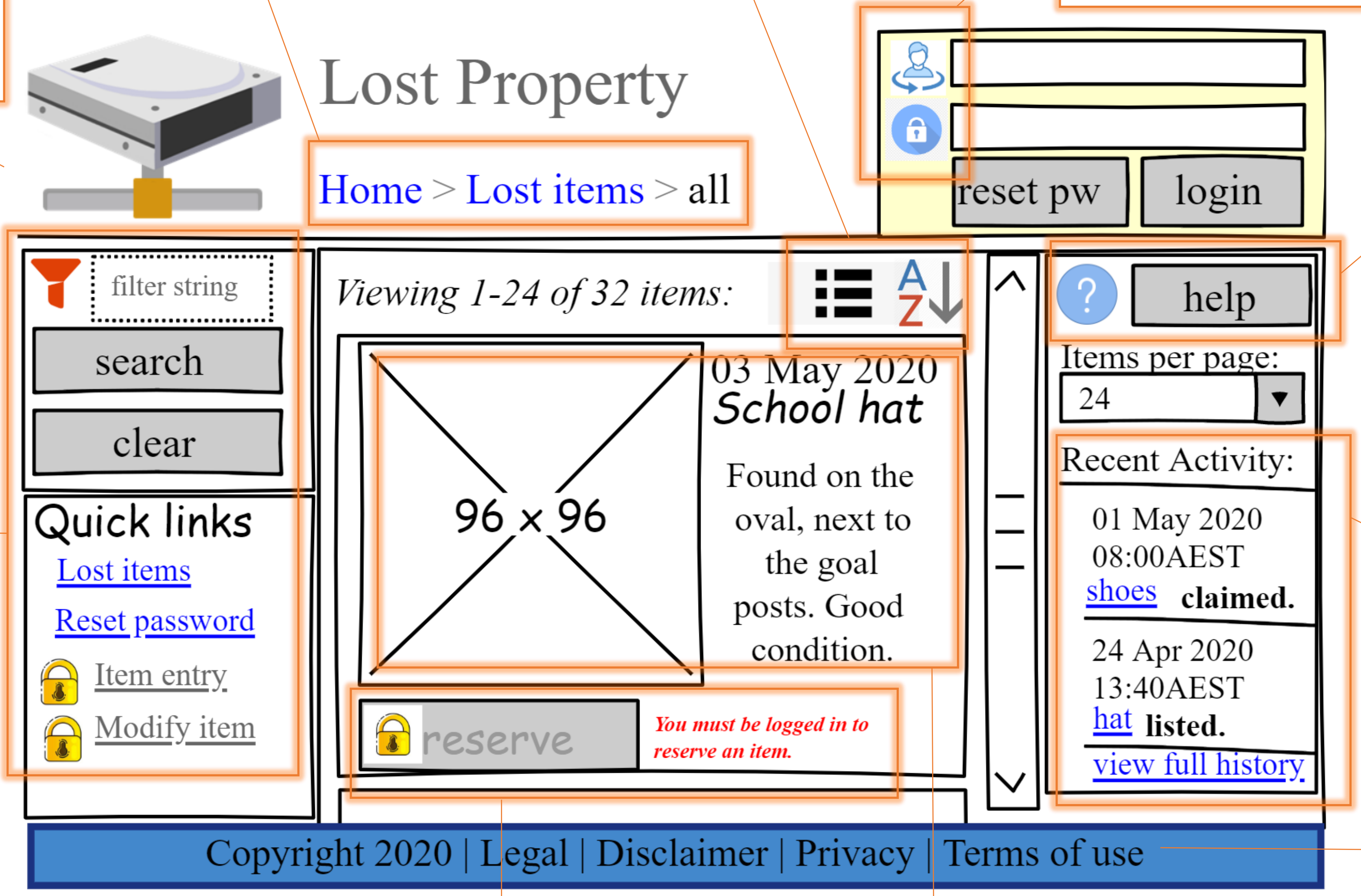
Learnability – consistent use of visual, recognisable iconography throughout site to provide familiarity for first time users.

Learnability – online help accessible, will launch links to interactive tutorials or contacts for further assistance.

Effectiveness – lets users quickly search for a lost item using a filter, whether their account exists or is logged in to save time.

Quick links for quick access to areas that are frequently accessed (which can be determined by site metrics).

Page results can be sorted by items per page (not circled), with pagination used at the top of the section to indicate the records being browsed.



Found on the oval, next to the goal posts. Good condition.

You must be logged in to reserve an item.

01 May 2020 08:00AEST shoes **claimed.**

24 Apr 2020 13:40AEST hat **listed.**

Effectiveness – history is backed up via cloud provider and will be restored on event of server crash. Transaction log adds a layer of security insofar as false transactions can be easily recognised and flagged via display.

Useability principles: principles used to improve the user experience, including:

- *accessibility*: ability to be used by many different people, even people with disabilities
- *effectiveness*: ability of users to use the system to do the work they need to do, includes reliability
- *safety*: ability for users to make errors and recover from the mistake
- *utility*: ability of the system to provide all the functionality that users need
- *learnability*: how easy a system is to learn.

Queensland Curriculum & Assessment Authority, "Digital Solutions 2019 v1.2 General Senior Syllabus". (2020). Retrieved DD Month YYYY, from https://www.qcaa.qld.edu.au/downloads/senior-qce/syllabuses/snr_digital_solutions_19_syll.pdf

Accessibility – terms of use shows compliance of site with accessibility guidelines identified in Technical Proposal (stimulus).

Safety – error message in red to illustrate steps to resolve problem. Visual iconography (padlock) to indicate secure access.

Utility – Lost Property management application delivers a solution to the Technical Proposal by displaying images, descriptions, and metadata (such as date time found, logged etc) of lost items.

Accessibility – containers stack elements for mobile using a responsive CSS framework:

`<header>`
`<nav>`
`<section>`
`<aside>`
`<footer>`

Note the semantic tag article is disregarded, as the article tag is used for independent, self-contained content irrespective of the page. Source: https://www.w3schools.com/tags/tag_article.asp

Learnability – button sizes enlarged on mobile devices to enable easier clicking and navigation.

Iconography consistent with desktop experience to enable seamless transition to mobile application.

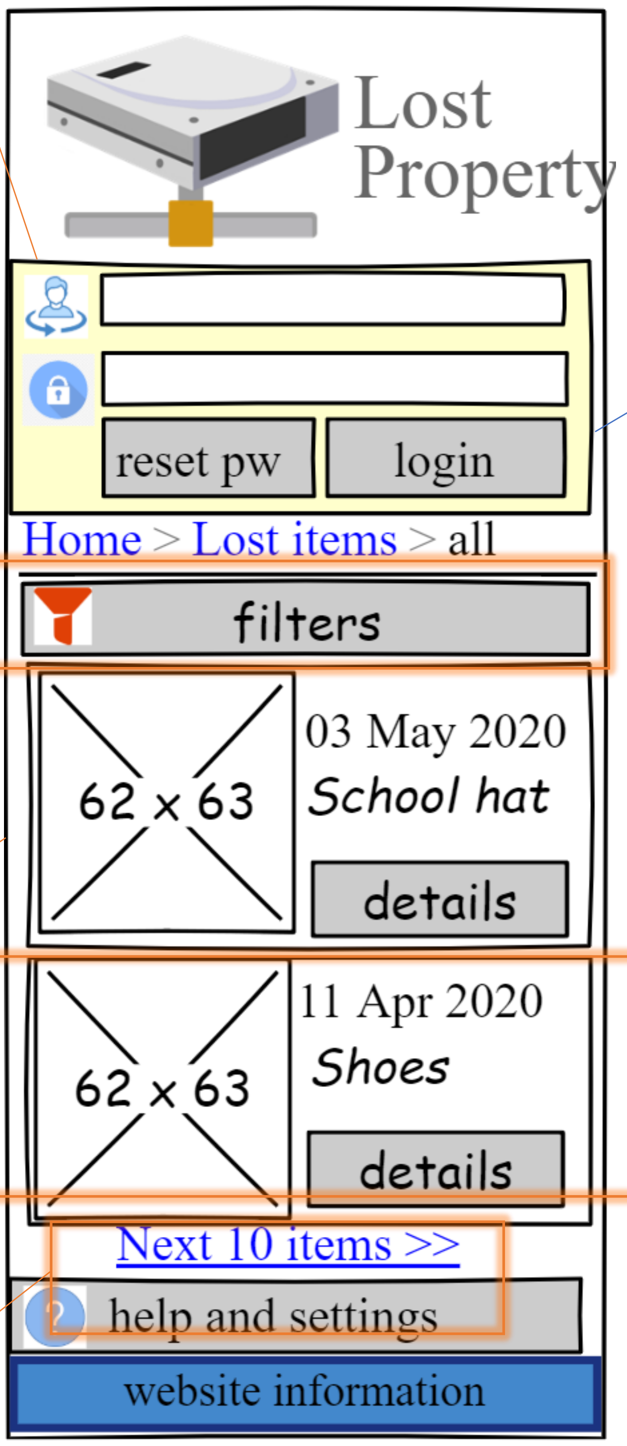
Stacking of containers using responsive framework enables most frequented areas to appear at top of stack, enabling success in finding and navigating via mobile or tablet device for first time.

Safety – vertical scrollbar removed, as browsing on a mobile device will make use of swipe screen gestures. Accessing a scrollbar on mobile is difficult as these are precision elements intended for pointing device, which can lead to scrolling errors.

Accessibility – items displayed in grid view to accommodate shortage in device screen size. Descriptions hidden behind Details link. Images resized as a percentage of viewport width.

Effectiveness – all lost item results still displayed enabling the success of the application on mobile platform. Some default settings assumed, such as 10 items per page, which can be changed in settings

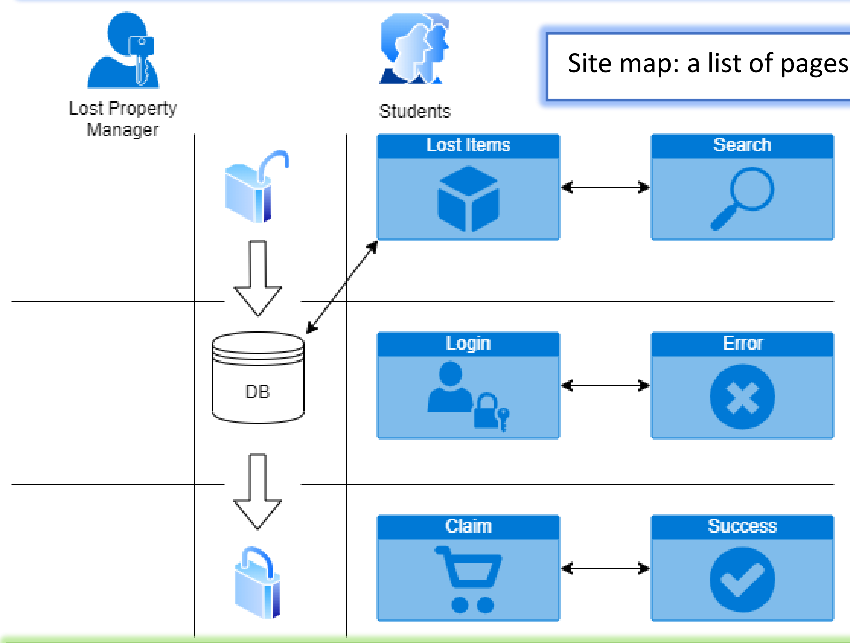
Utility – functionality still available via collapsible containers that can be opened, including as login, filters, help and settings and website information.



Interrelationships between user experiences and data of the digital prototype:

UI element	***Related Table.Field – see ERD below***	***Method or function call - see Algorithms below***
`txtUsername`	`Users.Username`	
`txtPassword`	`Users.Password`	
`btnReset`		`reset_password( username )`
`btnLogin`		`authenticate_user( username, password )`

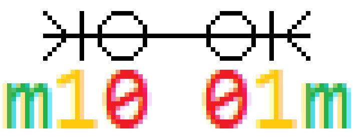
Site map: a list of pages of a web site.



Data and programmed components and their inter-relationships to the structure of the low-fidelity prototype digital solution:

Interrelationships with UX – Programmed components, algorithms, and generation of code:		
Programmed components	*Algorithm name*	*Code*
`reset_password()`	resetPassword	Found under app.route("/reset") Line 52
`authenticate_user()`	login	Not yet implemented
Interrelationships with UX – Programmed components with data structures and SQL statements:		
Programmed components	*Data structure in ERD*	*SQL statement*
`reset_password()`	Users table	~~UPDATE user SQL Statement #1~~ (needs renaming)

The Crow's Feet Notation shows relationship **modality** and **cardinality**:



The **modality** refers to the *minimum* number of times an instance of an entity can be associated with instances in the related entity. It is shown by the *inner-bound* as a **zero** or **one** (**0** or **|**).

The **cardinality** refers to the *maximum* number of times an instance of an entity can be associated with instances in the related entity. It is shown by the *outer-bound* (i.e. the one touching the other entity) as a **one** or **many** (**|** or **<**).

Each new record or tuple in the Distribution table will be associated with one (and only one) Article.

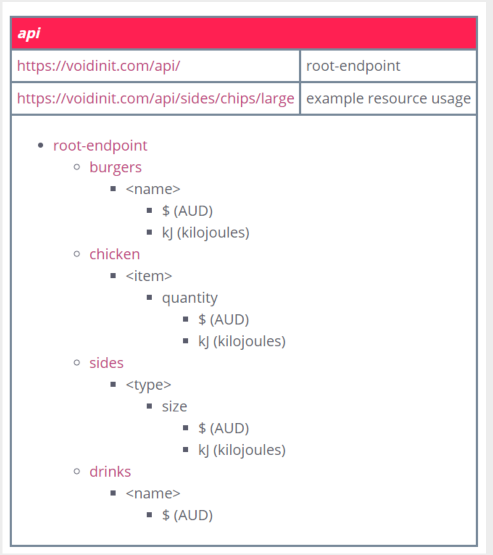
This lower bound modality constraint enforces the logic that if an instance an Article exists, it **must** also be Distributed.

This is assuming all Articles are written for *at least one* Website, on the premise that a Website will initially commission an Article to be written.

In addition to this, an Article may also be Distributed to many different Websites.

JSON / API data source analysis

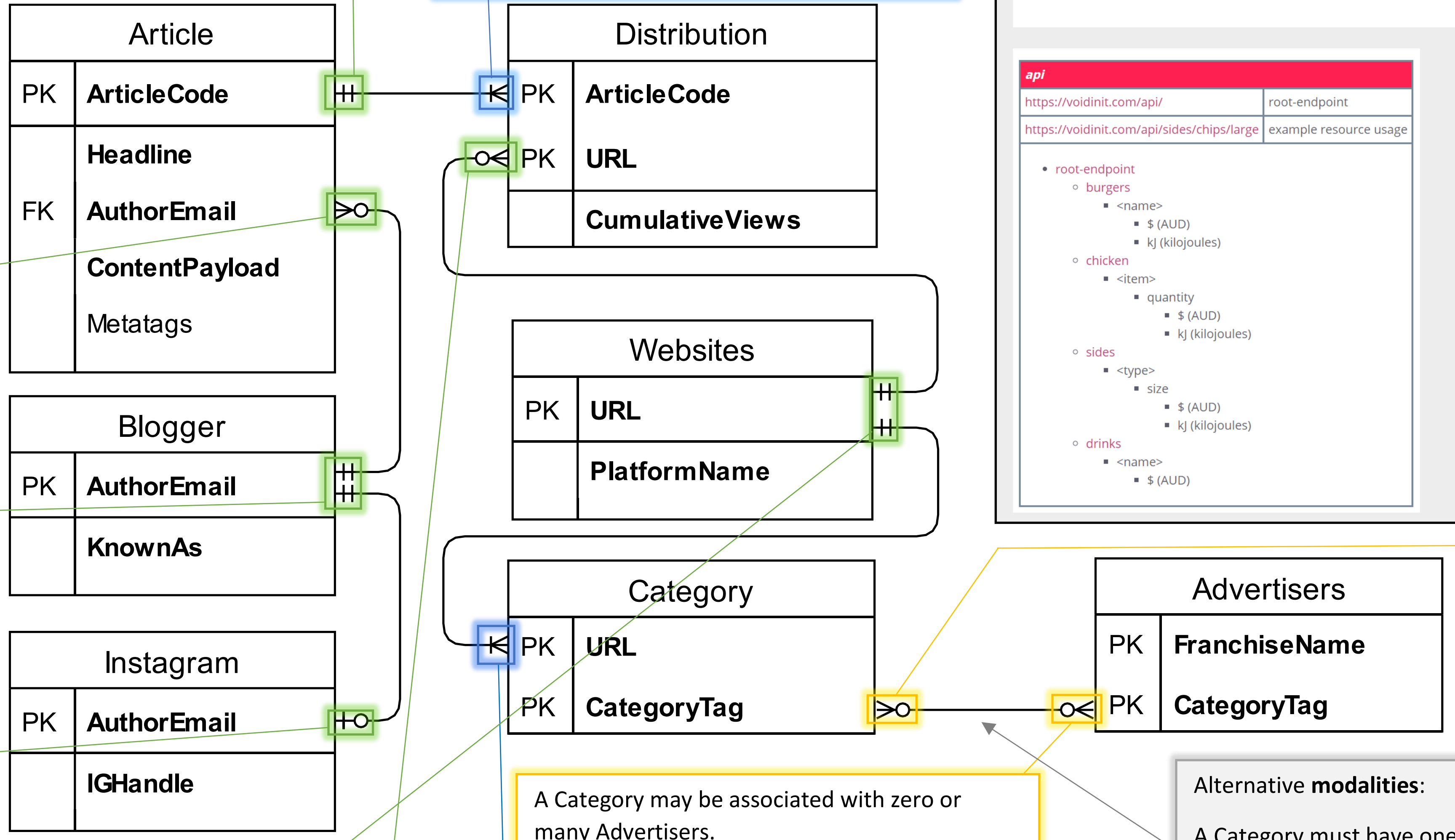
```
{"burgers": {"zinger": {"kJ": 1779, "$": 5.95}, "original_fillet": {"kJ": 1666,
"$": 5.95}, "zinger_stacker": {"kJ": 2996, "$": 8.95}, "zinger_bacon_and_cheese":
{"kJ": 3574, "$": 9.95}, "original_bacon_and_cheese": {"kJ": 4169, "$": 12.45},
"bbq_bacon_stacker": {"kJ": 2975, "$": 9.95}, "double_tender": {"kJ": 2127, "$":
4.95}}, "chicken": {"original_recipe": {"21_piece": {"kJ": 18853, "$": 34.95},
"5_piece": {"kJ": 4489, "$": 12.95}}, "nuggets": {"24": {"kJ": 5675, "$": 10},
"10": {"kJ": 2292, "$": 8.95}, "6": {"kJ": 1306, "$": 5.95}}}, "sides":
{"potato_and_gravy": {"large": {"kJ": 1318, "$": 5.95}, "regular": {"kJ": 296,
"$": 3.75}}, "chips": {"large": {"kJ": 2376, "$": 4.95}, "regular": {"kJ": 1188,
"$": 2.95}}, "dinner_roll": {"kJ": 508, "$": 0.95}}, "drinks": {"7up": {"$":
3.1}, "mountain_dew": {"$": 3.1}, "pepsi": {"$": 3.1}}}
```



api

https://voidinit.com/api/	root-endpoint
https://voidinit.com/api/sides/chips/large	example resource usage

- root-endpoint
  - burgers
    - <name>
      - $ (AUD)
      - kJ (kilojoules)
  - chicken
    - <item>
      - quantity
        - $ (AUD)
        - kJ (kilojoules)
  - sides
    - <type>
      - size
        - $ (AUD)
        - kJ (kilojoules)
  - drinks
    - <name>
      - $ (AUD)



Bloggers may be recorded in the Blogger table without ever having written an Article. The same Blogger may also write multiple Articles.

Each Article must be written by one (and only one) Blogger.

Any Instagram handles that exists must be attached to only one Blogger.

A Blogger may have an Instagram handle. Then again, *they may not*.

This relationship allows a maximum of one Instagram Handle (IGHandle) per Blogger.

Each individual record or row in the Distribution table will be linked to one (and only one) Website.

Similarly, each Category tag made will be linked to one and one only URL (per tag).

Not all recorded Websites may have published an Article yet. Some Websites may have published many different Articles.

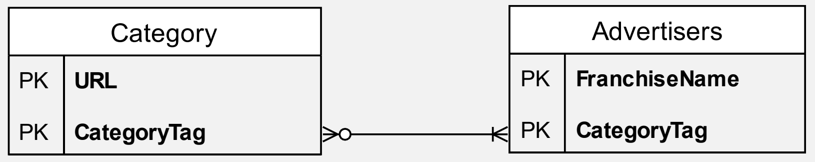
Every Website **must** be tagged with *at least one* Category. A Website may also be tagged with many Categories. The composite key allows different Website URLs to be tagged with the same Category.

A Category may be associated with zero or many Advertisers.

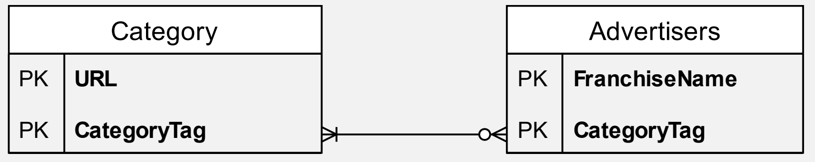
An Advertiser may be associated with zero or many Categories.

Alternative **modalities**:

A Category must have one associated Advertiser for that Category:



An Advertiser can only exist if they advertise at least one Category:



```sql
CREATE TABLE Article (
  ArticleCode INTEGER NOT NULL UNIQUE PRIMARY KEY AUTOINCREMENT,
  Headline CHAR(50) NOT NULL,
  AuthorEmail CHAR(128) NOT NULL,
  ContentPayload TEXT NOT NULL,
  Metatags TEXT
);

CREATE TABLE Websites (
  URL CHAR(128) NOT NULL UNIQUE PRIMARY KEY,
  PlatformName CHAR(50) NOT NULL
);

CREATE TABLE Distribution (
  ArticleCode INTEGER NOT NULL,
  URL CHAR(128) NOT NULL,
  PRIMARY KEY (ArticleCode, URL)
);
```

```sql
DELETE FROM Websites WHERE PlatformName == "YouTube";
```

```sql
UPDATE Websites SET URL = "m.facebook.com" WHERE URL == "facebook.com";
```

```sql
CREATE TABLE Distribution (
  ArticleCode INTEGER NOT NULL,
  URL CHAR(128) NOT NULL,
  PRIMARY KEY (ArticleCode, URL)
  FOREIGN KEY (ArticleCode) REFERENCES Article (ArticleCode) ON DELETE CASCADE ON UPDATE CASCADE
  FOREIGN KEY (URL) REFERENCES Websites (URL) ON DELETE CASCADE ON UPDATE CASCADE
);
```

```sql
INSERT INTO Article (Headline, AuthorEmail, ContentPayload) VALUES ("Hi!", "jo@jo.com", "<h1>Hello World</h1>");
INSERT INTO Websites (URL, PlatformName) VALUES ("facebook.com", "Facebook");
INSERT INTO Websites (URL, PlatformName) VALUES ("youtube.com", "YouTube");
INSERT INTO Distribution (ArticleCode, URL) VALUES (1, "youtube.com");
```

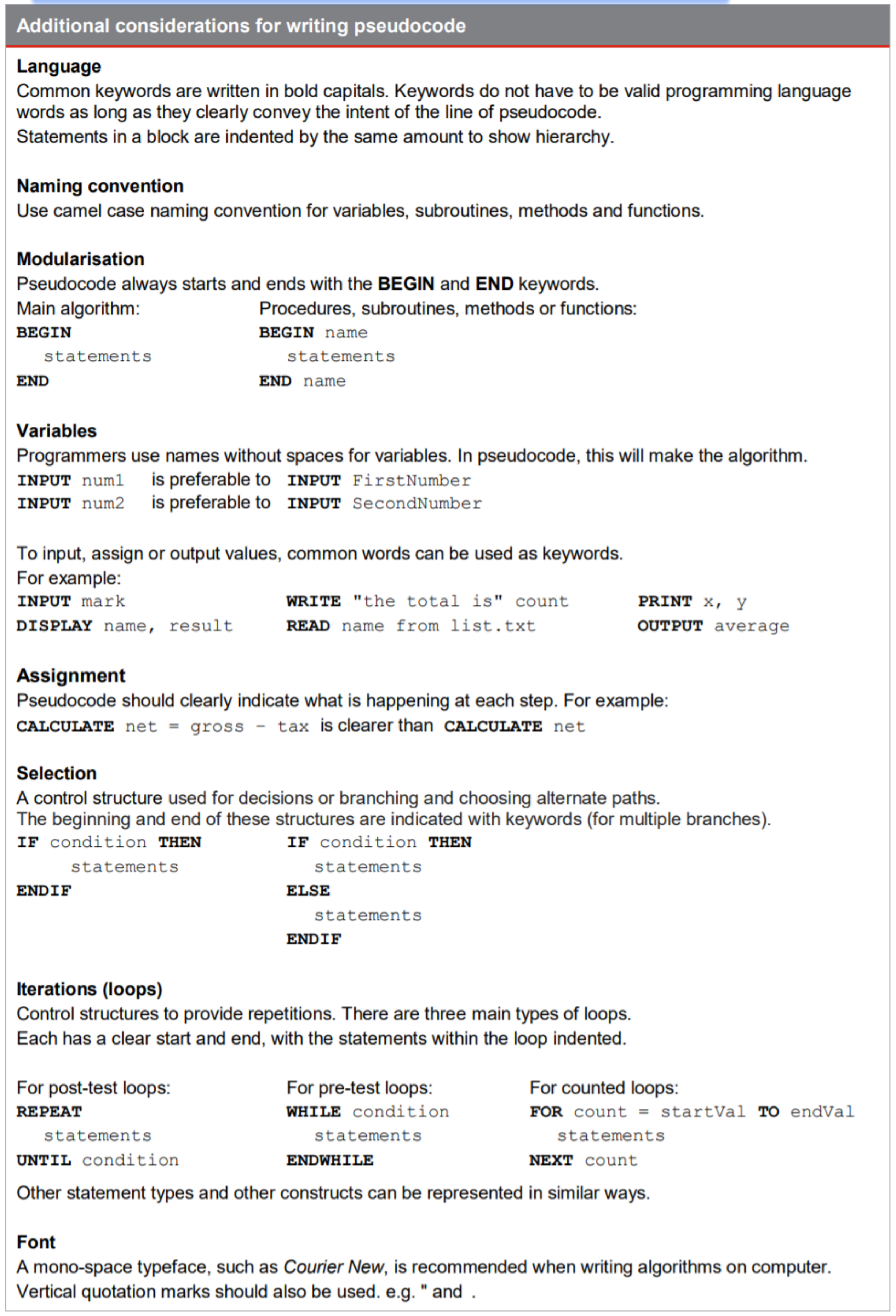
SQL Statements

- **CREATE**
- **INSERT**
- **UPDATE**
- **DELETE**
- **SELECT**

Table metadata

Article					
Name	Data type	Default value	Size limit or bound constraint	Use	Notes
ArticleCode	integer			Unique article identifier	Increments automatically
Headline	string		128 characters	Article headline	
AuthorEmail	string		128 characters	Written by	
ContentPayload	string		Text (no limit)	Article text	Mark-up
Metatags	string	null	128 characters	Used to index article in search engines	optional

Ensure algorithms appear here in a `console` font:



Additional considerations for writing pseudocode

Language
Common keywords are written in bold capitals. Keywords do not have to be valid programming language words as long as they clearly convey the intent of the line of pseudocode.
Statements in a block are indented by the same amount to show hierarchy.

Naming convention
Use camel case naming convention for variables, subroutines, methods and functions.

Modularisation
Pseudocode always starts and ends with the **BEGIN** and **END** keywords.

Main algorithm:
```
BEGIN
  statements
END
```
Procedures, subroutines, methods or functions:
```
BEGIN name
  statements
END name
```

Variables
Programmers use names without spaces for variables. In pseudocode, this will make the algorithm.
`INPUT num1` is preferable to `INPUT FirstNumber`
`INPUT num2` is preferable to `INPUT SecondNumber`

To input, assign or output values, common words can be used as keywords.
For example:
```
INPUT mark               WRITE "the total is" count      PRINT x, y
DISPLAY name, result     READ name from list.txt         OUTPUT average
```

Assignment
Pseudocode should clearly indicate what is happening at each step. For example:
`CALCULATE net = gross - tax` is clearer than `CALCULATE net`

Selection
A control structure used for decisions or branching and choosing alternate paths.
The beginning and end of these structures are indicated with keywords (for multiple branches).
```
IF condition THEN
    statements
ENDIF
```
```
IF condition THEN
    statements
ELSE
    statements
ENDIF
```

Iterations (loops)
Control structures to provide repetitions. There are three main types of loops.
Each has a clear start and end, with the statements within the loop indented.

For post-test loops:
```
REPEAT
  statements
UNTIL condition
```
For pre-test loops:
```
WHILE condition
  statements
ENDWHILE
```
For counted loops:
```
FOR count = startVal TO endVal
  statements
NEXT count
```
Other statement types and other constructs can be represented in similar ways.

Font
A mono-space typeface, such as *Courier New*, is recommended when writing algorithms on computer.
Vertical quotation marks should also be used. e.g. " and .

Simple Algorithms:

- Page navigation
- Login or sign up
- Browse data
- CREATE, INSERT, DELETE, UPDATE

More complex:

- Search
- Sort
- Rank
- Filter
- String manipulation

Very complex:

- Match-making
- Pattern analysis
- Prediction
- ~~Encryption~~

```
BEGIN resetPassword
INPUT username
EXECUTE "
  UPDATE Users
  SET Password = 'password'
  WHERE username = username;
"
END resetPassword
```

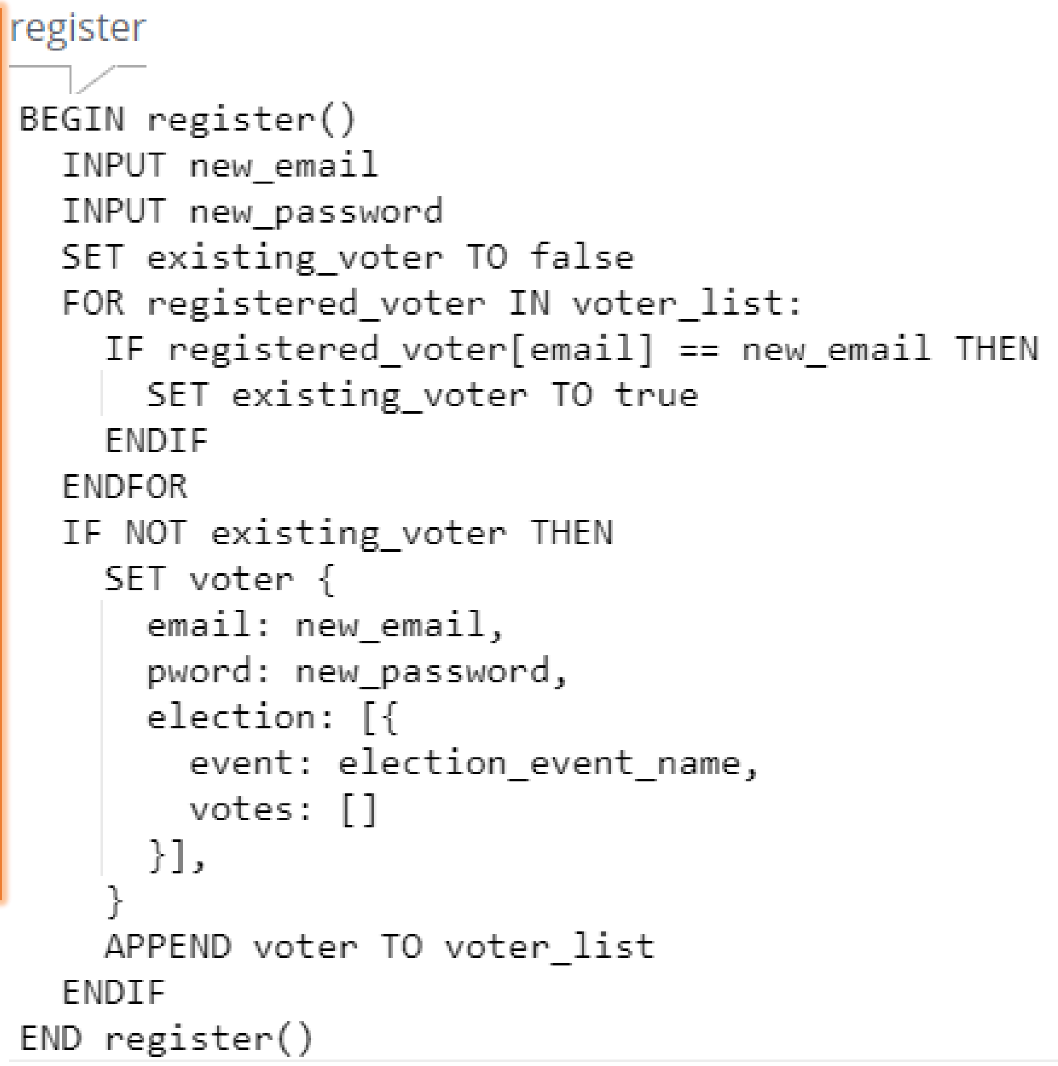
SQL Statement #1

Algorithm: step-by-step procedure required to solve a problem.

Pseudocode: a type of descriptive algorithm that is a mixture of everyday language.

With your algorithms, aim for:

1. Modularisation (break them up, not 1 big algorithm)
2. Consistency of words – don't change (e.g. mixing INPUT and GET is a bad idea)
3. Clear wording – algorithms are not Python code, they are *language independent* so none of this:
   ```
   app.route("/etc")
   def something():
   ```
4. Complexity. Try and include:
   a. Algorithms that loop
   b. Algorithms that branch
5. Try and incorporate either a list or dictionary data structure (as shown)
6. Consistency of indenting – it really, really matters.



register

```
BEGIN register()
  INPUT new_email
  INPUT new_password
  SET existing_voter TO false
  FOR registered_voter IN voter_list:
    IF registered_voter[email] == new_email THEN
      SET existing_voter TO true
    ENDIF
  ENDFOR
  IF NOT existing_voter THEN
    SET voter {
      email: new_email,
      pword: new_password,
      election: [{
        event: election_event_name,
        votes: []
      }],
    }
    APPEND voter TO voter_list
  ENDIF
END register()
```

determine_winner

```
BEGIN determine_winner()
  #set first candidate as default leader on initial run:
  SET leading_candidate TO election_event_candidates[0][name]
  SET leading_rank TO election_event_candidates[0][result][rank]
  FOR candidate IN election_event_candidates:
    IF candidate[result][rank] > leading_rank THEN
      SET leading_candidate TO candidate[name]
    ENDIF
  ENDFOR
  RETURN leading_candidate
END determine_winner()
```

voting

```
BEGIN voting()
  INPUT close_voting_timedate
  SET now_timedate TO SYSTEM.TIME
  WHILE now_timedate < close_voting_timedate
    voter = authenticate()
    IF voter IS NOT NULL THEN
      FOR candidate IN election_event_candidates:
        INPUT preference_ranking
        SET voter[election][votes][preference_ranking] TO candidate[name]
      ENDFOR
    ENDIF
    SET now_timedate TO SYSTEM.TIME
  END WHILE
END voting()
```

authenticate

```
BEGIN authenticate()
  INPUT login_attempt_email
  INPUT login_attempt_password
  SET authenticated_voter TO NULL
  FOR registered_voter IN voter_list:
    IF registered_voter[email] == login_attempt_email THEN
      IF registered_voter[pword] == login_attempt_password THEN
        SET authenticated_voter TO registered_voter
      ENDIF
    ENDIF
  ENDFOR
  RETURN authenticated_voter
ND authenticate()
```

launch

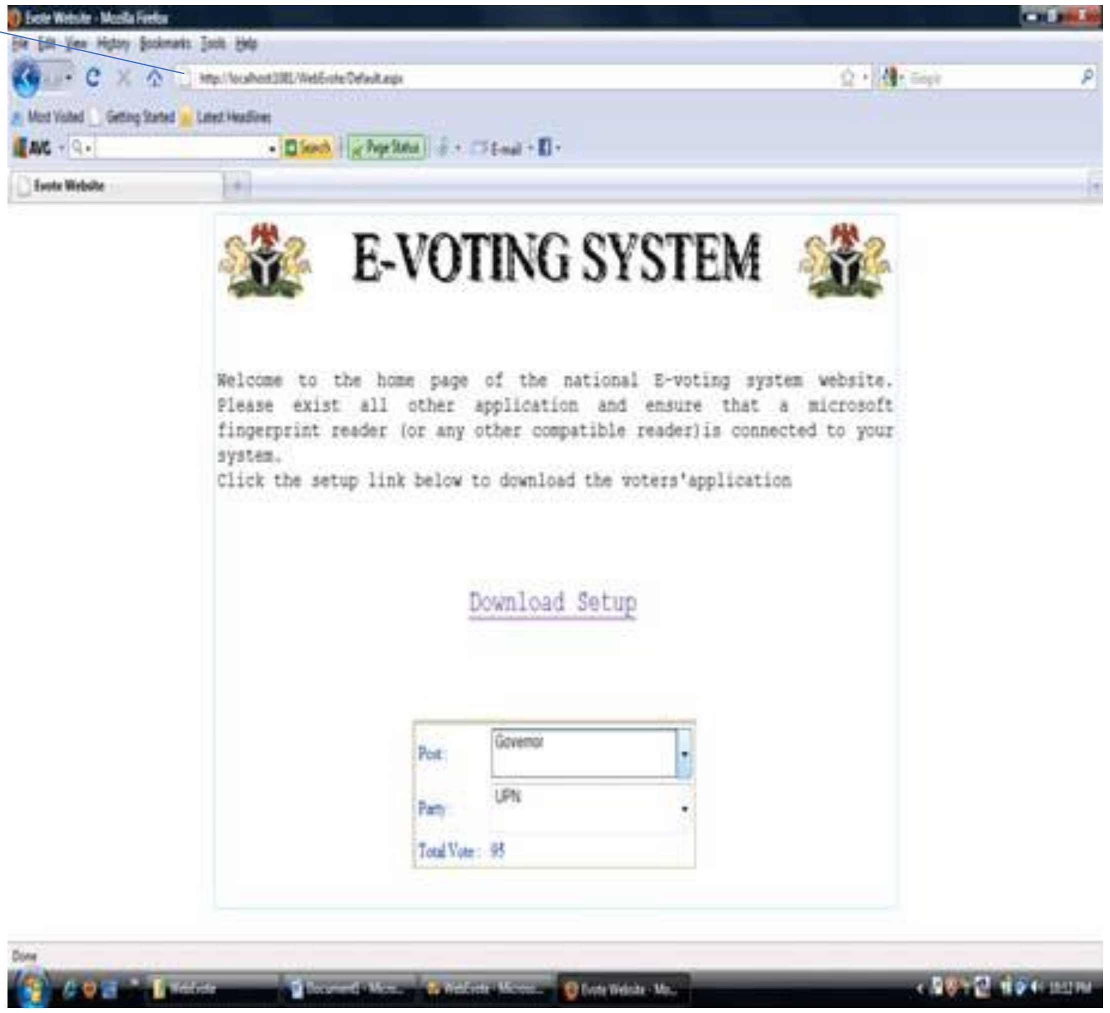
```
BEGIN launch(prereg_voters)
  IF election_event_name UNDEFINED THEN
    INPUT GLOBAL election_event_name
  ENDIF
  IF election_event_candidates UNDEFINED THEN
    SET GLOBAL election_event_candidates TO {}
    WHILE INPUT candidate_name
      SET candidate {
        name: candidate_name,
        result: {
          rank: 0
        }
      }
      APPEND candidate TO election_event_candidates
    ENDWHILE
  ENDIF
  IF voter_list UNDEFINED THEN
    SET GLOBAL voter_list TO []
  ENDIF

  #populate existing voter list into application data structure:

  IF prereg_voters DEFINED THEN
    FOR each_voter IN prereg_voters:
      SET voter {
        email: prereg_voters[email],
        pword: prereg_voters[password],
        election: [{
          event: election_event_name,
          votes: []
        }],
      }
      APPEND voter TO voter_list
    ENDFOR
  ENDIF
END launch()
```

Launch

- **More algorithms**
- **Initial builds (aka wire frames)**

tally_votes

```
BEGIN tally_votes()
  FOR vote IN voter_list:
    FOR preference IN RANGE(LENGTH OF election_event_candidates):
      SET chosen_candidate TO vote[election][votes][preference]
      INCREASE election_event_candidates[chosen_candidate][result][rank] BY preference
    ENDFOR
  ENDFOR
END tally_votes()
```



Welcome to the home page of the national E-voting system website. Please exist all other application and ensure that a microsoft fingerprint reader (or any other compatible reader)is connected to your system.
Click the setup link below to download the voters'application

Download Setup

Post: Governor
Party: UPN
Total Vote: 95

```
from flask import *

##################### CREATE DATABASE:
import sqlite3
from sqlstrings import *
import os
if not(os.path.exists("lfs.db")): #on first
launch
  db = sqlite3.connect('lfs.db')
  db.cursor().executescript(create_database)
  db.close()

##################### SETUP EMPLOYEE ACCOUNTS:


users = {
  # username : [ password, full name ]
  "jane@altavista.com":["abc123","Jane Citizen"],
  "wayne@netscape.com":["wayne07","Wayne Smith"],
}

##################### CREATE APP AND SET UP LOGIN LOGOUT FUNCTIONS:


app = Flask(__name__)
app.secret_key = "sssshhhhhhhh!"

@app.route("/")
def root():
  if session.get("logged_in") == True:
    return render_template("home.html", user=session["user"])
  else:
    return render_template("home.html")
```

Annotations

```
@app.route("/login", methods=["POST"])
def login():
  username = request.form["username"]
  password = request.form["password"]
  if username in users:
    if password == users[username][0]:
      session["logged_in"] = True
      session["user"] = users[username][1]
  return redirect("/")

@app.route("/logout")
def logout():
  session.pop("user", None)
  session.pop("logged_in", None)
  return redirect("/")


######################################################
#####################   1. TRACK MY PACKAGE (unsecured):

@app.route("/tracker")
def tracker():
  if session.get("logged_in") == True:
    return render_template("tracker.html", user=session["user"])
  else:
    return render_template("tracker.html")

@app.route("/tracker_display", methods=["GET"])
def tracker_display():
  pid = request.args.get("pack_id")
  db = sqlite3.connect('lfs.db')
  result = db.cursor().execute(tracker_display_sql, (pid,)).fetchone()
```

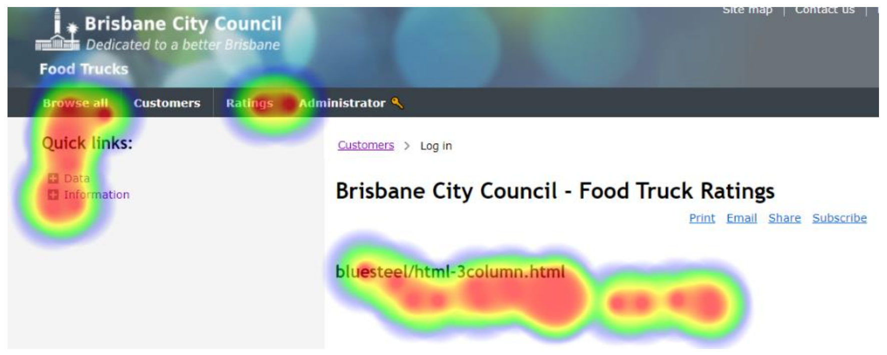
More Annotations

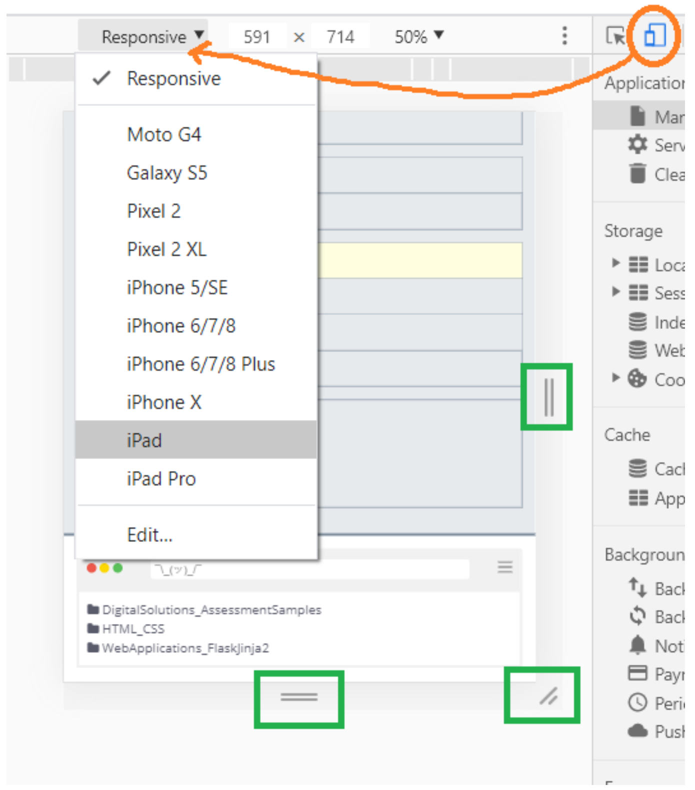
```
  db.close()

  if session.get("logged_in") == True:

    return render_template("tracker_result.html", main=result,
user=session["user"])

  else:

    return render_template("tracker_result.html", main=result)



######################################################

#####################   2. LOG NEW DELIVERY:


@app.route("/customer")

def customer():

  if session.get("logged_in") == True:

    return render_template("new_delivery.html", user=session["user"])

  else:

    return redirect("/") # secured, redirect if not logged in


@app.route("/customer_new_delivery", methods=["POST"])

def customer_new_delivery():

  if session.get("logged_in") == True:

    cust_name = request.form["cust_name"]

    dec_signed = request.form["dec_signed"]

    dest_region = request.form["dest_region"]

    db = sqlite3.connect('lfs.db')

    db.cursor().execute(customer_new_delivery_sql,
(cust_name,dec_signed,dest_region))

    db.commit()

    most_recent_pack_id =
db.cursor().execute(get_most_recent_package_id).fetchone()

    db.close()

    return render_template("new_delivery_result.html",
                           main="Your pack ID: " +
str(most_recent_pack_id[0]),
                           user=session["user"])
  else:
    return redirect("/")

######################################################
#####################   3. DISPATCH DELIVERIES:
@app.route("/dispatch", methods=["GET"])
def dispatch():
  if session.get("logged_in") == True:
    dest_region = request.args.get("region")
    db = sqlite3.connect('lfs.db')
    result = db.cursor().execute(get_delivery_list,
(dest_region,)).fetchall()
    db.close()
    return render_template("dispatch.html", user=session["user"],
            main=str(result), r=dest_region) #r=hidden region
  else:
    return redirect("/")

@app.route("/mark_as_delivering", methods=["POST"])
def mark_as_delivering():
  if session.get("logged_in") == True:
    courier_name = request.form["courier_name"]
    dest_region = request.form["dest_region"].strip()
    db = sqlite3.connect('lfs.db')
    db.cursor().execute(update_delivery, (courier_name,dest_region,))
    db.commit()
    db.close()
  return redirect("/")

######################################################
```

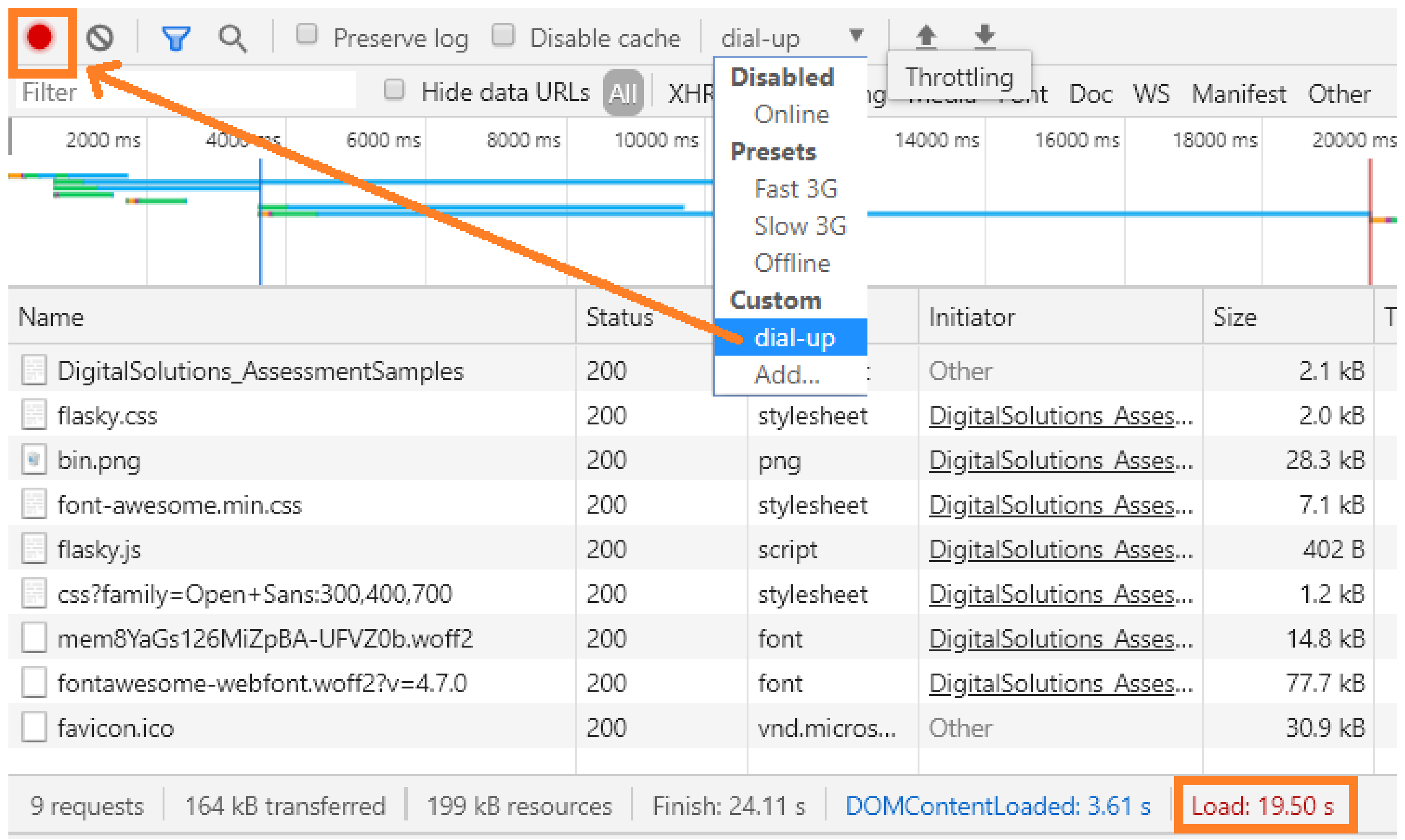
```
#####################   4. DRIVER ASSISTANCE:
@app.route("/delivery")
def delivery():
  if session.get("logged_in") == True:
    return render_template("driver_assistance.html", user=session["user"])
  else:
    return redirect("/")

app.run(debug=True)
```

```
<header>
    <title>Local Freight Services</title>
    <h1 class="page-title">Local Freight Services</h1>
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-
awesome.min.css">
    <link rel="stylesheet" href="{{ url_for('static', filename='lfs.css')
}}">
</header>
<main>
      <div class="intro">
            <h1 class="title">Your local delivery service.</h1>
      </div>
      <div class="account">
        <p class="meta">Current location: <a href="/">home</a><br>

            {% if user is defined %}
                Logged in as: <mark>{{user}}</mark>. <a
href="/logout">logout</a></p>
            {% endif %}

        </div>
      <div class="icons">
        <a href="/"><span class="fa fas fa-home"></span></a>
        <a href="/tracker"><span class="fa fa-map-marker"></span></a>
      </div>
      <div class="links">
            <h3>Links</h3>
            <p><i class="fa fa-map-marker"></i> <a href="/tracker">Track
my delivery</a></p>
        <p><i class="fa fa-lock"></i> <a href="/customer">New delivery
request</a></p>
        <p><i class="fa fa-lock"></i>Warehouse dispatch<br>
            <a href="/dispatch?region=NORTH">NORTH</a>
            <a href="/dispatch?region=EAST">EAST</a>
            <a href="/dispatch?region=WEST">WEST</a>
            <a href="/dispatch?region=SOUTH">SOUTH</a>
      </p>
        <p><i class="fa fa-lock"></i> <a href="/delivery">Driver
assistance</a></p>
      </div>
      <div class="content">
      {% if user is not defined %}
           <form action="/login" method="post">
            <label for="username">Username:</label> <input type="text"
name="username" required><br>
            <label for="password">Password:</label> <input type="password"
name="password" required>
            <input type="submit" value="Login">
        </form>
      {% endif %}
            <p>We have one large warehouse where we operate our entire
delivery service from, which is located in the main street of town. Bring
your articles of postage to our customer service outlet at the front of
the warehouse, where our customer service attendant will receive your
article and tender the delivery fee. Within 24 hours your article is
processed and dispatched by the Delivery Dispatch Manager to a Courier for
delivery.</p>
```

Truncated – CODE HAS BEEN CLIPPED HERE BUT CAN BE SEEN IN THE VIDEO


Brisbane City Council
Dedicated to a better Brisbane
Food Trucks
Browse all
Customers
Ratings
Administrator
Quick links:
Customers > Log in
Brisbane City Council - Food Truck Ratings
Print Email Share Subscribe
bluesteel/html-3column.html





Name	Status		Initiator	Size
DigitalSolutions_AssessmentSamples	200		Other	2.1 kB
flasky.css	200	stylesheet	DigitalSolutions_Asses...	2.0 kB
bin.png	200	png	DigitalSolutions_Asses...	28.3 kB
font-awesome.min.css	200	stylesheet	DigitalSolutions_Asses...	7.1 kB
flasky.js	200	script	DigitalSolutions_Asses...	402 B
css?family=Open+Sans:300,400,700	200	stylesheet	DigitalSolutions_Asses...	1.2 kB
mem8YaGs126MiZpBA-UFVZ0b.woff2	200	font	DigitalSolutions_Asses...	14.8 kB
fontawesome-webfont.woff2?v=4.7.0	200	font	DigitalSolutions_Asses...	77.7 kB
favicon.ico	200	vnd.micros...	Other	30.9 kB

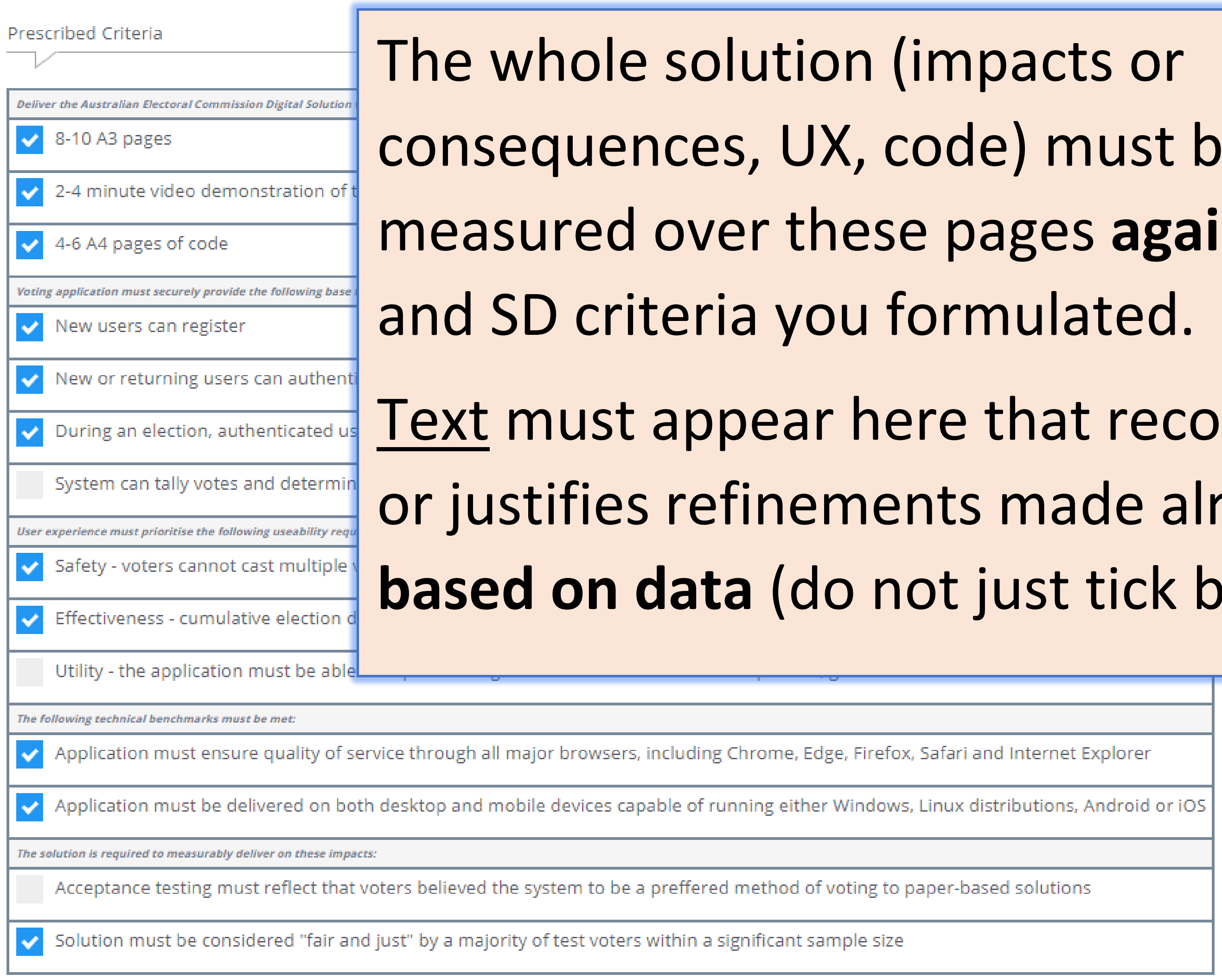
9 requests | 164 kB transferred | 199 kB resources | Finish: 24.11 s | DOMContentLoaded: 3.61 s | Load: 19.50 s

Refinements during development – screenshot errors

More ideas for testing include anything that can determine results for:

- functionality - such as user acceptance testing, making sure solution can handle required tasks
- useability - observe a user, time them to complete a task, and observe any confusion, errors or difficulties
- UI - colour contrast checkers, accessibility testers, test for vision resizing, language translation, screen readers
- compatibility - such as cross browser testing (edge chrome safari firefox ie & mobile browsers)
- performance - such as simulating different internet connections & download speeds
- security - discovering vulnerabilities, SQL injection attack, decoding Flask session cookies, cached credentials, password security, encryption techniques

Evaluation: make an appraisal by weighing up or assessing strengths, implications and limitations; make judgments about ideas, works, solutions or methods in relation to selected criteria; examine and determine the merit, value or significance of something, based on criteria to make refinements and recommendations - justified by data.



Prescribed Criteria

Deliver the Australian Electoral Commission Digital Solution	
✓	8-10 A3 pages
✓	2-4 minute video demonstration of t
✓	4-6 A4 pages of code
Voting application must securely provide the following base	
✓	New users can register
✓	New or returning users can authenti
✓	During an election, authenticated us
	System can tally votes and determin
User experience must prioritise the following useability requ	
✓	Safety - voters cannot cast multiple v
✓	Effectiveness - cumulative election d
	Utility - the application must be able
The following technical benchmarks must be met:	
✓	Application must ensure quality of service through all major browsers, including Chrome, Edge, Firefox, Safari and Internet Explorer
✓	Application must be delivered on both desktop and mobile devices capable of running either Windows, Linux distributions, Android or iOS
The solution is required to measurably deliver on these impacts:	
	Acceptance testing must reflect that voters believed the system to be a preffered method of voting to paper-based solutions
✓	Solution must be considered "fair and just" by a majority of test voters within a significant sample size

The whole solution (impacts or consequences, UX, code) must be measured over these pages **against** the P and SD criteria you formulated.

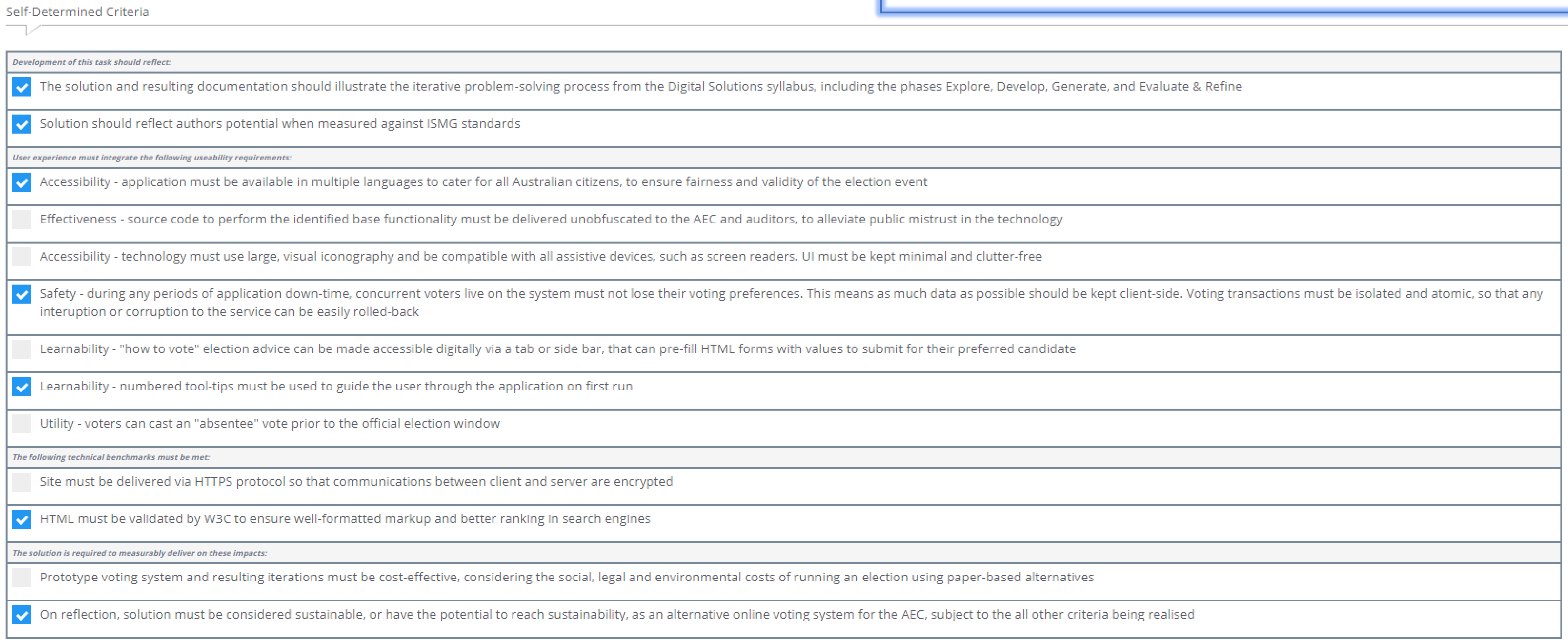
Text must appear here that recommends or justifies refinements made already **based on data** (do not just tick boxes).

critical evaluation of impacts, user experience and coded components and the digital solution against essential prescribed and self-determined criteria to make discerning **refinements** and astute **recommendations** justified by data.

You must have proper reference list / proper referencing format for perfect result:

Queensland Curriculum & Assessment Authority, "Digital Solutions 2019 v1.2 General Senior Syllabus". (2019).

Retrieved DD Month YYYY, from https://www.qcaa.qld.edu.au/downloads/senior-qce/syllabuses/snr_digital_solutions_19_syll.pdf



Self-Determined Criteria

Development of this task should reflect:	
✓	The solution and resulting documentation should illustrate the iterative problem-solving process from the Digital Solutions syllabus, including the phases Explore, Develop, Generate, and Evaluate & Refine
✓	Solution should reflect authors potential when measured against ISMG standards
User experience must integrate the following useability requirements:	
✓	Accessibility - application must be available in multiple languages to cater for all Australian citizens, to ensure fairness and validity of the election event
	Effectiveness - source code to perform the identified base functionality must be delivered unobfuscated to the AEC and auditors, to alleviate public mistrust in the technology
	Accessibility - technology must use large, visual iconography and be compatible with all assistive devices, such as screen readers. UI must be kept minimal and clutter-free
✓	Safety - during any periods of application down-time, concurrent voters live on the system must not lose their voting preferences. This means as much data as possible should be kept client-side. Voting transactions must be isolated and atomic, so that any interuption or corruption to the service can be easily rolled-back
	Learnability - "how to vote" election advice can be made accessible digitally via a tab or side bar, that can pre-fill HTML forms with values to submit for their preferred candidate
✓	Learnability - numbered tool-tips must be used to guide the user through the application on first run
	Utility - voters can cast an "absentee" vote prior to the official election window
The following technical benchmarks must be met:	
	Site must be delivered via HTTPS protocol so that communications between client and server are encrypted
✓	HTML must be validated by W3C to ensure well-formatted markup and better ranking in search engines
The solution is required to measurably deliver on these impacts:	
	Prototype voting system and resulting iterations must be cost-effective, considering the social, legal and environmental costs of running an election using paper-based alternatives
✓	On reflection, solution must be considered sustainable, or have the potential to reach sustainability, as an alternative online voting system for the AEC, subject to the all other criteria being realised