

FSAD Project

Phase 1 - Architecture and Design

Database Design

The User's details collection, the authentication for "*Admins*" and the contact details for "*Students/Parents*".

Users

| Field Name | Data Type | Validation and or Notes |
|--------------|---------------|---|
| id | Int (Primary) | Given in increments of 1 starting with 10011. (Or this could be auto-generated and incremented by MySQL, this is to be reviewed later). Used for login. |
| name | String | Data Required. Must be Full name. |
| role | String | Enum: ['admin', 'student', 'parent'] Defaults to 'student' |
| email | Varchar | Data Required. Must be unique. Used for login Used as contact |
| password | Varchar | Data Required. Store as a bcrypt hash, never as plain test. |
| phone_number | Int | Data Optional. Used as Contact |

Tutors

| Field Name | Data Type | Validation and or Notes |
|-----------------|-----------------|---|
| id | Int (Foreign) | Given in increments of 1 starting with 20021. (Or this could be auto-generated and incremented by MySQL, this is to be reviewed later). Used for login. |
| name | String | Data Required. Must be Full name. |
| email | Varchar | Data Required. Must be unique. Used for login Used as contact |
| password | Varchar | Data Required. Store as a bcrypt hash, never as plain text. |
| phone_number | Int | Data Optional. Used as Contact |
| Bio | String | Data Required. Short description for profile. |
| Profile_picture | String | URL string set to image file. |

| Field Name | Data Type | Validation and or Notes |
|-------------------|------------------|---|
| Qualifications | Varchar | <p>Data Required.</p> <p>Metric to assess tutors capability.</p> <p>A-levels/GCSE or other professional qualifications.</p> |
| DBS | String | <p>Data Required.</p> <p>Must be updated to recent check.</p> |
| SubjectID | Int (Foreign) | <p>Array["Maths", "English", "Science"].</p> <p>Selected by tutor.</p> <p>One to one link with subjects in booking table.</p> |

Sessions/Bookings

| Field Name | Data Type | Validation and or Notes |
|-------------------|------------------|---|
| id | Int (Foreign) | Auto-generated |
| SubjectID | Int (Foreign) | One to one link with subjects in tutor table. |
| StudentID | Int (Foreign) | <p>User id must exist.</p> <p>Reference linking session/booking to student.</p> |
| TutorID | Int (Foreign) | <p>User id must exist.</p> <p>Reference linking to tutor.</p> |
| Available_Dates | Date | Set by tutor |
| Time_Slots | Int | Set by tutor |
| Tutor_Mode | String | In-person or online class with tutor |
| Status | String | Enum: ['Pending', |

| | | |
|-------------------|-----|--|
| | | 'Confirmed', 'Cancelled' Default to "Pending". |
| Sessions_Attended | Int | Shows number of attended sessions. Should link to calendar to show the days of attended sessions. |
| Dates_Booked | Int | Data Required. |

Resources

| Field Name | Data Type | Validation and or Notes |
|------------|-----------------|--|
| id | Int (Foreign) | Auto-generated |
| title | String | Required. Example. "Year 11 Mathematics Scheme" |
| type | String | Enum: ['Free', 'Paid']. |
| subject | String | Category for filtering. |
| file_Url | String | Required. Path to downloadable file(PDF). |
| UploadBy | Int | Reference User. |

I chose a relational database(MySQL) structure over NoSQL because of the assignment requirements, giving a good standard quality whilst not going too far with unnecessary features. MySQL allows me to have multiple, complex relationships between the tables and makes it easier to query the database due to the predefined schema set and its structured nature format.

What I will showcase on the homepage of the mobile view website version:

| Feature | Position |
|--|---|
| The "Academic Wizard" name | at the Left side of the Header |
| Hamburger menu that encloses: <ul style="list-style-type: none"> • "Find a tutor" • "Book a session" • "Study resources" • "Admin" • "Help and faq" | at the right side of the Header |
| A "Login" | in the header |
| A Banner with a motto | at the right side of the page |
| A "book a session" button | On the bottom of the banner to the left |
| A Scroller | at the right side of the page |
| Testimonials | displayed for the rest of the screen |
| "About us" | at the left side of the footer of the page |
| "Contact info " | at the right side of the footer of the page |

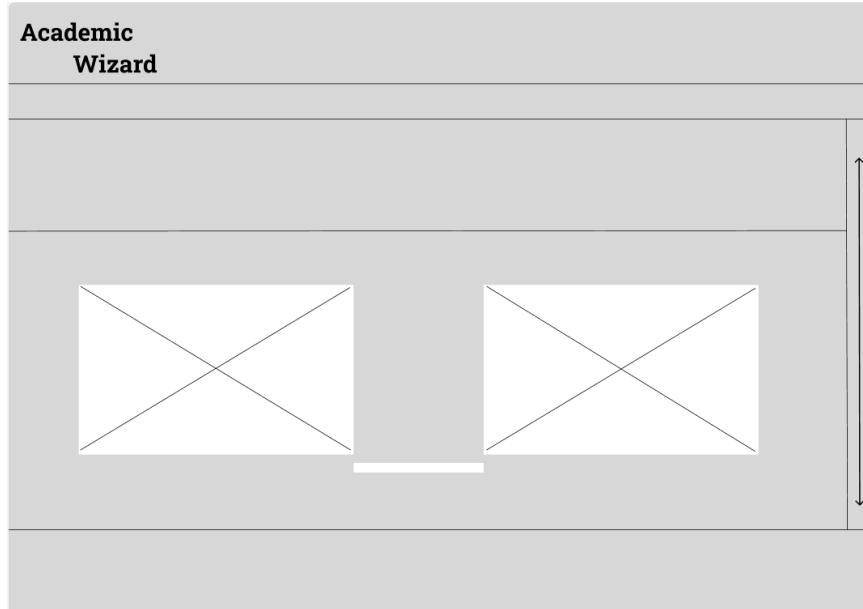
The mobile view website created by Figma works well. Some changes I had in mind for it were:

- Make the testimonials move horizontal like college website rather than vertically
- Remove the Subjects list.

Task 2

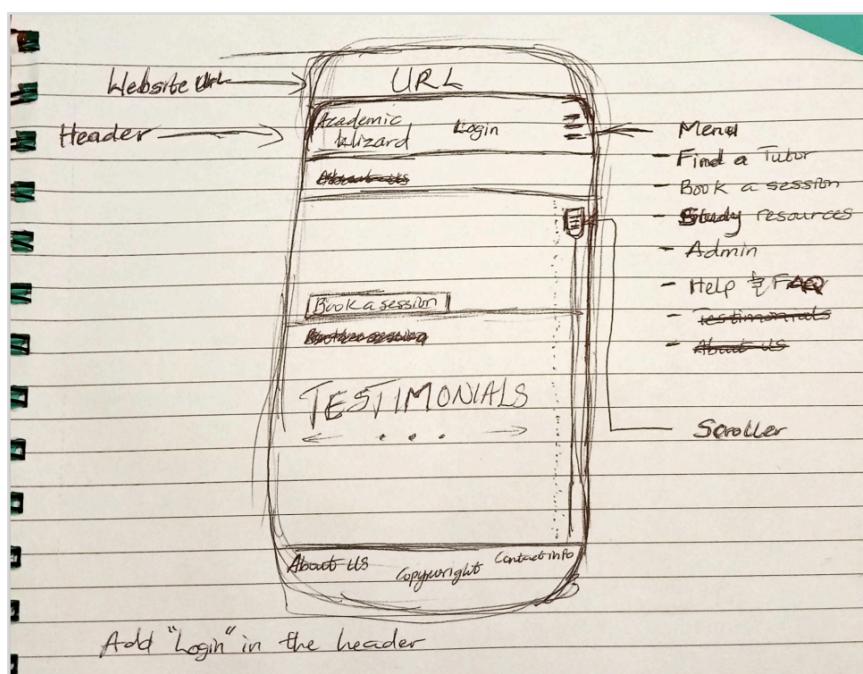
Design Documentation

- Wireframes and UI mockups :-



○

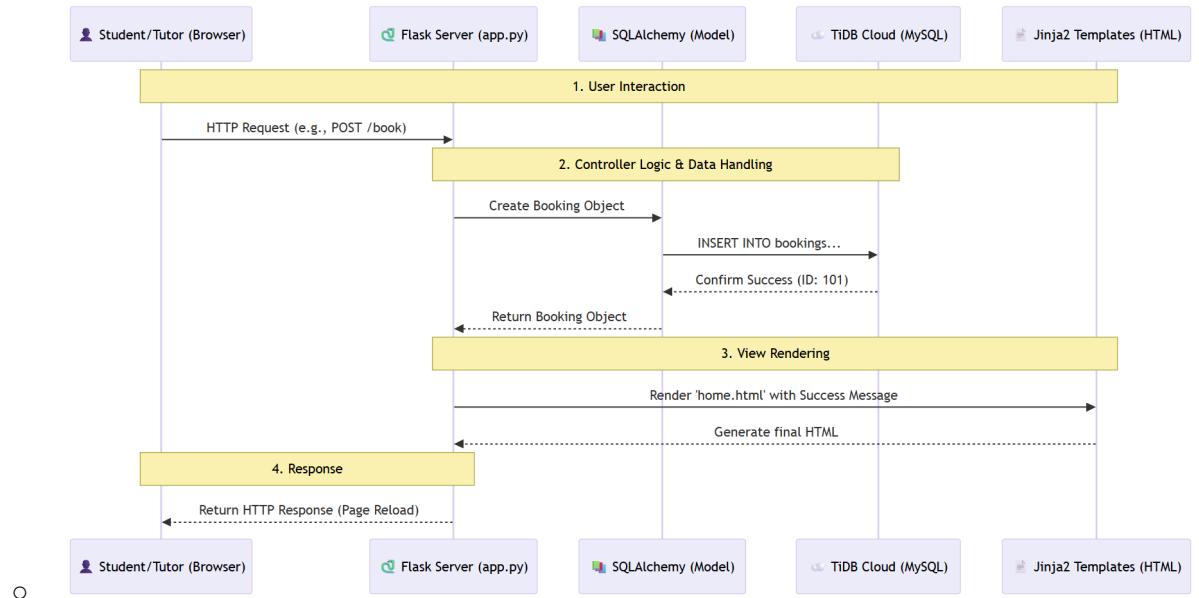
- Sketch of Homepage



○

Regarding accessibility, I designed the menu button to be at least 44x44 pixels to meet the touch target accessibility standards, ensuring it is easily accessible to all users regardless of motor skills.

- **System architecture** :- The application follows the Model-View-Controller (MVC) architectural pattern. This separates the internal representation of information from the way information is presented to and accepted from the user.



- **Diagram**

- **Database** :- TiDB Cloud (MySQL): A serverless, distributed SQL database hosted in the cloud.
- **Server** :- Python Flask: A lightweight WSGI web application framework.
- **Front-end** :- Jinja2 Templates: HTML files with dynamic content rendering, styled with standard CSS3.

- **Database schema** :-

- Table: Users (Student/Parent)
 - **`id`** (INT, Primary Key, Auto-increment)
 - **`name`** (VARCHAR, Required)
 - **`email`** (VARCHAR, Unique, Required)
 - **`role`** (ENUM: 'Student', 'Parent', 'Admin')
 - **`password_hash`** (VARCHAR, Secure storage)
- Table: Tutors
 - **`id`** (INT, Primary Key, Auto-increment)
 - **`name`** (VARCHAR, Required)
 - **`subject`** (VARCHAR, Required)
 - **`bio`** (TEXT)
 - **`is_verified`** (BOOLEAN)
- Table: Bookings
 - **`id`** (INT, Primary Key, Auto-increment)
 - **`student_id`** (INT, Foreign Key linked to Users)
 - **`tutor_id`** (INT, Foreign Key linked to Tutors)
 - **`date`** (DATE)

- `time` (TIME)
 - `status` (ENUM: 'Pending', 'Confirmed')
- Used Technology justification :-
 - Python & Flask: Flask was chosen for its lightweight nature, allowing for rapid prototyping without the heavy boilerplate of larger frameworks. It integrates seamlessly with SQLAlchemy for robust database management.
 - MySQL (TiDB Cloud): A relational database was selected over NoSQL because the data is highly structured (e.g., a Booking *must* link to a valid Tutor and Student). TiDB Cloud was chosen to ensure database accessibility from restricted college networks where local database ports were blocked.

Development Process

- Steps taken :-
- Requirement Analysis: Defined the core features (Booking, Tutor selection) and database structure.
 - Environment Setup: Attempted Node.js setup (blocked), then successfully established a Python virtual environment.
 - Database Connection: Configured TiDB Cloud with whitelist rules to allow connection from the college network.
 - Backend Logic: Implemented Flask routes to handle form submissions (Booking/Registration).
 - Frontend Integration: Created Jinja2 templates to display database content dynamically.

- Relevant code

- Snippets :-

```

# Model Definition (SQLAlchemy)
class Booking(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    student_name = db.Column(db.String(100),
                           nullable=False)
    tutor_id = db.Column(db.Integer,
                        db.ForeignKey('tutor.id'))
    date = db.Column(db.String(20), nullable=False)

# Controller Route (Flask)
@app.route('/book', methods=['POST'])
def book_session():
    tutor_id = request.form.get('tutor_id')
    new_booking = Booking(tutor_id=tutor_id, ...)
    db.session.add(new_booking)
  
```

```
        db.session.commit()
        return redirect(url_for('home'))
```

- Explanation :-
 - The Model snippet defines the structure of the `Booking` table using Python classes, which SQLAlchemy translates into SQL commands.
 - The Controller snippet listens for a POST request from the browser (when a user clicks "Book"), captures the form data, saves it to the database, and refreshes the page.
- Challenges
 - Encountered :-

Restricted Execution Environment: The college's Group Policy prevented the installation of Node.js packages and blocked the execution of runtime scripts (PowerShell restrictions).

Additionally, the college firewall blocked standard local database ports (3306), preventing a local database instance from running.
 - Solution :-

Architecture Pivot: I pivoted the technology stack to Python, which was whitelisted on the college system. To solve the database connection issue, I utilized TiDB Cloud (Port 4000) and configured a "Public Endpoint" with a wildcard (`0.0.0.0/255.255.255.255`) whitelist.

This effectively bypassed the local restrictions by offloading the database to the cloud.

Evaluation

- Effectiveness in usage of
 - API's :- The application effectively communicates with the TiDB Cloud API via the SQLAlchemy connector. This proved highly effective for maintaining data persistence across different locations (College and Home) without manual data transfer.
 - Frameworks :- Flask proved highly effective for this scale of application. Its modular design allowed for easy separation of the Routing logic (`app.py`) and the Presentation logic (`templates`), fulfilling the MVC requirement efficiently.
- Evaluation of website's
 - Strengths :-

- Portability: Due to the cloud database integration, the application is "device agnostic" and retains data state regardless of where it is run.
- Architecture: The strict MVC structure ensures the code is maintainable and scalable.
- Weaknesses :-
 - Styling: The current user interface relies on basic CSS and lacks responsive design for smaller mobile screens.
 - Authentication: The current "Login" is simulated (entering a name) rather than a secure hashed-password system.
- Suggestions of
 - Potential improvements :-
 - Secure Authentication: Implementing "Flask-Login" to handle secure user sessions and password hashing.
 - Input Validation: Adding server-side validation to ensure dates selected are in the future and do not clash with existing bookings.
 - Additional features :-
 - Calendar Integration: Visualizing bookings on a calendar view rather than a list.
 - Payment Gateway: integrating Stripe API to allow for paid tutor sessions.

Reflection

I learned a lot during this project, especially mistakes and suggestions for future projects such as:

- Creating the entire website pages in one single HTML file divided by lines and spaces to start when working solo, then separating them into different files later on.
- Taking notes of every thought and idea and correction to keep a clear mind while I work, and taking this note with me everywhere I go.
- At certain times take a step back to get an overview of where I am at, and make plans accordingly moving forward on the next best course of action to keep to deadlines.

From this experience, I am more eager to practice these outside personal work to improve and be better prepared for forthcoming projects and future job projects, being open to fine tuning and/or correcting these reflection steps as I go.