UnicornAttractor Issue Tracker

What is required?

An issue tracker to log and track bugs and new features in the UnicornAttractor app.

The app will require secure user authentication and payment functionality.

The app must contain graphs showing statistics on bug fixes and new features.

Who will use the application?

It will be used by both users and developers of the UnicornAttractor app. The users will create tickets for bugs or suggested features and will be able to add comments to the tickets. The developers will also be able to comment and change the status of the ticket (pending, in progress, closed etc).

What technologies will be used?

The app will be written in Python using the Django framework. HTML, CSS and JavaScript will be used to create a good UX.

How will users interact with the application?

The application will be web based and accessible through all browsers on both desktop and mobile devices.

User stories

- As a user of the UnicornAttractor app, I would like to be able to suggest new features for the app, in order to have it fulfil my requirements more fully.
- As a developer of the UnicornAttractor app, I would like to have a central location for all work remaining on the app, as well as a method for users to pay for the work, in order to keep the app up to date and generate income from it.

Planning

Visual Layout

As the issue tracker's primary purpose is logging bugs and features, these should be presented on the main page of the app in a paginated list or panel display. It should be possible to click on any ticket to open it into a full page view. Any ticket that has changed since the user last logged in should be highlighted.

Bugs and features need to be clearly separate lists but both accessible using tabs.

The navigation bar should include links to Home, Account (submenu, initially containing Login/Register, then Profile/Logout/New Ticket), About (submenu containing Latest News/Statistics/What is UnicornAttractor?/Contact/Archives.

The user's profile page should contain all the details they entered when registering as well as the option to change some details, including the password. It should list tickets raised by that user,

donations towards new features made by that user, and if the user is a developer, tickets assigned to them.

The Latest News page will be in a blog format, showing a list of items that can be clicked to navigate to their full content. Commenting will be available. Articles will be related to the app: bug fixes, new features and other projects by the developers.

The Statistics page will be where the graphs and charts are displayed, showing the developer activity on the bugs and features.

There will be an archives page showing all resolved/implemented tickets.

Features to Implement

- User accounts
    - Registration (Developers should be able to register developer accounts which will require admin approval.)
    - Login/Logout functionality
    - Password reset
    - Profile page to change details
- Ticket logging
    - Ticket types (bugs/features)
    - Add ticket
    - Add comment
    - Add screenshot
    - Change status
    - Change priority
    - Vote
- Statistics
    - Bugs logged (last 24 hours, last 7 days, last 30 days)
    - Bugs fixed (last 24 hours, last 7 days, last 30 days)
    - Features logged (last 24 hours, last 7 days, last 30 days)
    - Features implemented (last 24 hours, last 7 days, last 30 days)
    - Top 5 new features pending/in progress.
    - Top 5 bugs pending/in progress
- Latest News
    - Add article (Developer accounts only)
- Contact
    - Contact form offering quotes on new projects
- Checkout
    - Order form
    - Payment form
    - Stripe integration
- About
    - Render About page
- Contact
    - Contact Form
    - Request for project

Database

The application will use Postgres and tables will be created dynamically using Django; models for these can be seen in the Models.py file in each section of the app.

Deployment

The application will be deployed on Heroku and use Heroku's Postgres functionality. Amazon Web Services S3 will be used for image storage.

Testing

The deployed app will make use of the Travis CI Testing platform for the automated tests.

Jasmine scripts will be used to test the JavaScript if necessary.

The app will also be manually tested by multiple users to ensure good UX and to check for bad links etc.

Questions

- At what point should user authentication be required?
    - Any page where ticket information is viewed?
    - Would require customisation of the login page to include links to about pages. May put users off signing up as they can't see what they are getting.
    -
    - Just to view tickets themselves?
    - This is certainly a minimum requirement as code already requires information about the current user to know what data to display. However this could be modified to include if not user etc. It means that users' interest is piqued by the minimum information given on the dashboard page and they can be prompted to login or sign up to view more. If users are able to view everything they may be less likely to sign up.
    -
    - Profiles of other users?
    - Dependent on what information will be shown on profiles, but as a user would need the direct url anyway this is perhaps not an issue.
    -

    Conclusion:

    Unauthenticated users can see:

    - All tickets dashboard (home page)
    - Latest News and articles
    - About page
    - Contact page

Must be authenticated for:

- Profiles
- Ticket view
- Submit a ticket
- Checkout view

- What packages does Python offer for the charting and which one would be best to use?

  After extensive research on the subject, I have opted to set up the Django Rest Framework to create an API, then utilise the JavaScript framework ChartsJS to do the charting.