

# Developing an Open-Source Budget Management System



Callahan Hirrel  
Hendrix College

A capstone project submitted for the degree of  
*Bachelor of Computer Science*

Hendrix College 2019

## **Abstract**

In colleges and universities, budget funding is allocated in a number of manners, across many channels. The project described in this paper aims to provide a way to maintain and view budgets for academic departments in real time. The administrative assistant for the Biology and Psychology Departments at Hendrix currently keeps track of and maintains the budget for two departments and over 20 professors. By using the application developed in this project, the faculty of these departments can view their departmental budget in real time as well as past budget data, allowing them to view precise fundings and plan for cyclic expenses. This capstone project is the beginning of a Web based application written in Python using the Flask framework. Budget data, including expenses and remaining allocations, is stored on a school server. It is open source, free to use, and implemented in Python as a means for simple contribution. At the current point in time, the application has a solid foundation for continued work. Extensive code comments have been added as a source for future students who wish to contribute to this open source project.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>4</b>
2.1	Problem Statement and User Needs . . . . .	5
2.2	Overview of Framework and Tools . . . . .	7
<b>3</b>	<b>Budgets in Real Time (BiRT)</b>	<b>10</b>
3.1	Development Approach . . . . .	10
3.2	Usage Overview . . . . .	14
<b>4</b>	<b>Future Work</b>	<b>16</b>
<b>5</b>	<b>Conclusion</b>	<b>18</b>
<b>A</b>	<b>Code Reference</b>	<b>19</b>
<b>B</b>	<b>GUI Reference</b>	<b>22</b>
	<b>Bibliography</b>	<b>26</b>

# Chapter 1

## Introduction

A great draw of computer science is that it uniquely has opportunity for a vast amount of interdisciplinary work. The power of computing and automation is utilized in some form in every discipline. Academics regularly use computers and programs to serve focused purposes, and regular use with programs gives these people great familiarity with the user side of applications, but little with the development side. Thus, when someone has an idea of a program that would help their field, or a way to amend a program they frequently use, they may have to spend a great deal of time learning how solve this problem themselves, or they may simply abandon this goal. Thus, computer scientists have many opportunities to explore areas of interest through a familiar development environment.

In colleges and universities, each academic department typically has a budget allocated over a regular time interval, and these allocations can be further divided into specific areas in which certain amounts of money may be spent. Typically, it is the job of a single person to keep track of this budget for a department and handle funding requests for faculty within the department.

At Hendrix, the administrative assistant for the Biology and Psychology Depart-

ments, Nora Simmons, currently keeps track of and maintains the budget for two departments and over 20 professors. In lieu of this task, she came up with the idea of a Web application to help, in which faculty of these departments can view budget information in real time. Thus, they can more accurately keep track of where money can be spent. She then extended a request to the Hendrix Computer Science Department for a student to bring her vision to life. This paper describes the startup and implementation of an early prototype of this application.

The overall goal of this project is to provide a way to maintain and view budgets for academic departments in real time. By using the application developed for this project, the faculty of these departments can view the departmental budget in real time as well as past budget data, allowing them to view precise funding distributions and plan for cyclic expenses.

The technical design aspects are as such: The application will be Web based and written in Python, using the Flask framework. Data will be stored on a school server. The final result will be open source and free to use. Additionally, the choice of Python as the primary language is due to it opening the project up to contributions from any student having taken the introductory Computer Science course at Hendrix.

The scope of the semester-long senior capstone project is to establish solid groundwork for the overall application development to continue with open source contributions, past the 2018 Fall Semester. This includes a prototype of the application, with some core features including user login, viewing data, and memory persistence amongst multiple users. These prototype Web pages are pragmatic, to illustrate back-end functionality. Thus, they are lacking in terms of visual design and styling. This choice to only build a prototype is to ensure that the final product is not rushed,

is polished, and can be easily deployed by any IT department. All startup design decisions are made with these end goals in mind, in addition to making sure the inner workings of the application are easy to navigate and understand for future contributors, students or otherwise.

# Chapter 2

## Background

As previously mentioned in the introduction section, Nora Simmons maintains the budget for two distinct departments and over 20 professors. Through interviews with her and other departmental faculty, I learned that budget reports are released monthly, as PDFs delivered via email. However, these “monthly” reports are occasionally late, sometimes up to six months. Furthermore, these PDFs simply display a table of text summarizing the current budget, giving information such as percent/amount of the money that is spent/available. These numbers are also broken down by category, such as money for supplies or travel. So, a member of the faculty in these departments will always have some copy of their respective budget, but it may not be current.

There are some advantages to their current system, mainly that it only involves knowledge of an email program, which is required for essentially any office-type job. Furthermore, `.pdf` is widely recognized as one of the most versatile file types, so anyone can open and view a file of this type on any modern computer. However, professors then independently manage personal budgets for their classes, using different combinations of paper, digital spreadsheets, and text files. They communicate

requests to the administrative assistant primarily over email. All this in different formats, both paper and digital, lends to disorganized and tedious management of funds. The administrative assistant recognized the multitude of flaws in the budget management of these departments. Hence, she reached out to the Department of Computer Science with a vision of a Web application to aid in the organization and management of allocated funds.

## 2.1 Problem Statement and User Needs

The overall goal of this application is to offer a means to maintain and view budget information for academic departments in real time. Thus, by developing the framework and more technical aspects, we enable those faculty to manage their own budget in a more organized and effective manner. By using this application, they will be able to view real time as well as past budget data, serving as a means to understand more precise funding distributions and plan for cyclic expenses. As users, they will be able to access this application from their computer's Web browser. In addition to viewing a real time funding distribution, they may use keywords to search a database of budget data, sort, show, and hide on screen data by specific fields, and view an archive of spending information. This feature was requested specifically so professors may plan for cyclic expenses, like an expenditure required for a class offered in alternating years (See Figure 2.1). These user stories were detailed in specific use cases, some of which are shown below. Note that **any user** is defined as the administrative assistant, the department chairs, and the department professors (of the Hendrix College Psychology and Biology departments). **System** refers to the server-side program,



and **application** refers to aspects visible from the client side.

**Use Case 11:** View Class-Specific Expenses

**Actor:** professor

**Description:** View classes' expenses, including class name/code, purchase, and amount, to plan and prepare

**Preconditions:** Actor logged into application

**Postcondition:** Application displays table of expenses for single class

**Main Success Scenario:**

1. Actor clicks "sort by my classes" dropdown menu
2. Actor selects a class
3. Actor clicks "sort by my classes" button
4. System generates sort query based on selected class
5. System queries database
6. Application loads query result as a table

Extensions:

- 3a. No class selected
  - 3a1. Application notifies actor
  - 3a2. Terminate use case

Figure 2.1: A use case detailing a professor viewing expenses specific to their own classes

Each user logs in to an account (see Figure 2.2), and different accounts will have different permissions within the application. For example, the administrative assistant will have administrator privileges, thus enabling her to more directly work with saved data. Administrator access includes the options to add (see Figure 2.3), edit, or remove specific entries of data, add new fields or categories for data, and overall maintain the budget database in a simple and intuitive manner.

Bigger goals were discussed that are not in the scope of this project, but still worth mentioning. These include a feature of the system which professors specifically may use to manage grants, outside of the realm of the college. Additionally, the faculty would like to at some point access this data on their phones, via a mobile application

**Use Case 1:** Log in to application

**Actor:** Any user

**Description:** Actor successfully logs into application

**Preconditions:** Actor has navigated to application in browser.

**Postcondition:** System loads actor's application homepage

**Main Success Scenario:**

1. Actor inputs username/password into system via login screen text fields, clicks login button
2. System authenticates actor's login information
3. System returns application homepage

**Extensions:**

- 2a. Actor login information not authenticated
  - 2a1. System returns login screen with error message
  - 2a2. Terminate use case

Figure 2.2: A simple use case for logging into the application

communicating over a wireless network.

## 2.2 Overview of Framework and Tools

This application will be open source, thus to ensure a more polished project, as opposed to one rushed to completion. With that in mind, the primary language for development is Python, thus enabling any student having taken Foundations of Computer Science course at Hendrix may contribute. The framework used to implement this application is Flask, as it is a Python framework. Furthermore, it is designed to be small, but easily scalable. Thus, it has less core functionality than other Web frameworks like ASP.NET, but has plenty of built in support for external libraries. This makes incremental development more intuitive, which is a plus when looking to pass this project on to future students or open source contributors. One unforeseen disadvantage is that using multiple external libraries necessitates an up to date de-

**Use Case 8:** Add Data Entry to Database

**Actor:** administrative assistant

**Description:** Add a data entry into the current budget database

**Preconditions:** Actor logged into application

**Postcondition:** New entry added to database and displayed to actor

**Main Success Scenario:**

1. Actor clicks "Add Data" button
2. System loads adding data page
3. Actor inputs data into fields
4. Actor clicks "Add Entry" button
5. System validates input
6. System adds entry to database
7. System notifies actor of successful entry
8. System adds data entry to main budget table

**Extensions:**

- 3a. Actor leaves 1+ fields blank
  - 3a1. System notifies actor
  - 3a2. System confirms the fields should remain blank
  - 3a3. Resume Step 5 of Main Success Scenario
- 5a. System finds invalid input
  - 5a1. System returns relevant error
  - 5a2. Terminate use case

Figure 2.3: A use case detailing the process of adding a data entry to the budget database

dependencies file. However, this makes for good experience in understanding the role a framework plays in overall Web development. HTML and CSS are used for the Web page design, hard coded within the framework, and Javascript adds dynamic elements to some pages (for example, sorting a table of budget data upon a mouse click). The technical details are further explained in Chapter 3, Section 3.1.

All of this technical planning serves the purpose of organizing the system of budget management currently employed by the Biology and Psychology Departments at

Hendrix. In the end, the final product should reduce some of the tedious nature of the job of administrative assistant. And, since it is open source, it will be free to use by any other organization that may find it practical.

## Chapter 3

# Budgets in Real Time (BiRT)

The project began with in-person interviews with Nora Simmons, Dr. Leslie Zorwick, Psychology Chair, and Dr. George Harper, Biology Chair. The information discussed at these meetings was then organized into a preliminary series of user stories and use cases. These documents motivated the development of the prototype application, which was given a name of “BiRT: Budgets in Real Time.”

### 3.1 Development Approach

As described previously, the application is implemented in Python using the Flask framework, a decision made with future student contributors in mind. Python is the first language taught in the Computer Science Department at Hendrix, and Flask has extensive external library support, providing a means for intuitive incremental development.

As with almost all Web based programs, the pages of this application are structured using HTML and styled using CSS. The HTML is both hard coded and generated using the Jinja template engine. Jinja is the default template engine for Flask,

and allows non-HTML code blocks within HTML code, which contain instructions for generating HTML [3]. Examples of these code blocks can be found in lines 8, 18, 25, and 30 of Appendix A.1, the application’s layout template, which defines the general layout of each page. The code in lines 8, 18, and 25 set up a conditional that determines what links should display in the navigation bar, depending on whether or not a user is signed in. The code in line 30 indicates the location in the HTML tree of all code for pages inheriting from this “super” template.

Almost all CSS for this project is courtesy of Bootstrap, a free and open source front-end framework for designing Web pages [1]. The classes used for styling and organizing page documents came either as built-in Bootstrap classes, or from Bootstrap templates within its own documentation [1].

A big aspect of this application is the use of forms, as forms are a tool to register users, log in users, to make requests as a user, and to edit the budget data as an administrator. Hence, all form work is handled by the WTForms library, which contains its own definitions of various input fields and its own validators [6]. Input fields are the different interface elements of user input, such as text boxes, check boxes, and drop down boxes. Validators are essentially functions that check user input, like to confirm an email address is correctly formatted or that the “password” and “confirm password” fields are equivalent. To further simplify usage, this library is integrated into the framework with the Flask-WTF library, which allows easy access to the WTForms API. With Flask-WTF, forms are represented as Python classes, as seen in Appendix A.3. This example is of a form users may use to update their account information. The fields of the form are defined as fields of the corresponding Python class, and the `validate_email` function serves as a custom validator, to ensure the

user does not input an email already in use. Furthermore, Appendix A.4 illustrates usage of these form tools within Jinja code blocks in HTML files. For instance, in line 11, `form.first_name.label` refers to the `label` variable, assigned as a string in the `first_name` field of the `UpdateAccountForm` class (see line 2 in Appendix A.3). Thus, the forms can be accessed from within the HTML files in which they will be rendered.

Another big aspect of this application is data storage and access, per its main purpose of offering real time budget data. Thus, an Object Relational Mapper (ORM) is employed to create a mapping between our database model and the development language's traditional class and object structure. Basically, the ORM allows us to represent database tables as classes in our language, and the rows are represented as fields of these classes. This makes changes to the database model very easy, as we can change our class definitions and the ORM changes the database model for us. The ORM used for this project is SQLAlchemy, as it is built to operate specifically with Python [4]. An example of a user data model represented as a Python class can be found in Appendix A.2. In that example, the variable `db` serves as an instance of SQLAlchemy, which allows the definition of columns, relationships, and primary keys, amongst other features. Furthermore, SQLAlchemy is a good choice of ORM because it does not generate database schemas, which can often cause problems when the overall database schema needs to be changed [4]. Thus, incremental development may occur without the anxiety of rehauling and generating a new database every time a change is made to the schema.

Now, SQLAlchemy is the top level of interaction with our database, so we must specify which database management system (DBMS) and query language we wish to

use. This can vary depending on the situation. Since this is just a prototype implementation of the overall application, we must consider which aspects would be best for a development DBMS. It must be simple to undertake the process of converting user interviews to entity-relationship diagrams (ERDs) to a logical database schema [7]. It should also be relatively simple to “facilitate database testing and data refresh ...for example, the ability to compare the contents of two tables” [7]. With these criteria, along with future student contributors in mind, we will select SQLite as our development DBMS. Further research must be done in order to determine the best production DBMS for this application’s contextualized purpose. SQLite fits our criteria, in that there exists websites like Vertabelo which will convert an ERD into a SQLite database schema. Furthermore, it has a very simple and intuitive command line interface, as well as numerous freely available graphical user interfaces such as DB Browser for SQLite.

Figure 3.1 is an ERD which illustrates the working data model. This ERD was created using Vertabelo, so “PK” denotes a primary key, “FK” denotes a foreign key in a relationship, and “N” defines a field as nullable. Relationships are defined such that each budget report has multiple entries, one for each object code in the budget. Additionally, each user can have zero or more budget requests. One design aspect that remains is defining a relationship between the two halves of this model, as to connect the budget requests of users to each budget report.

Much of development was driven by tutorials by Corey Shafer, with his code [2] drawn upon for source files and the overall package structure.



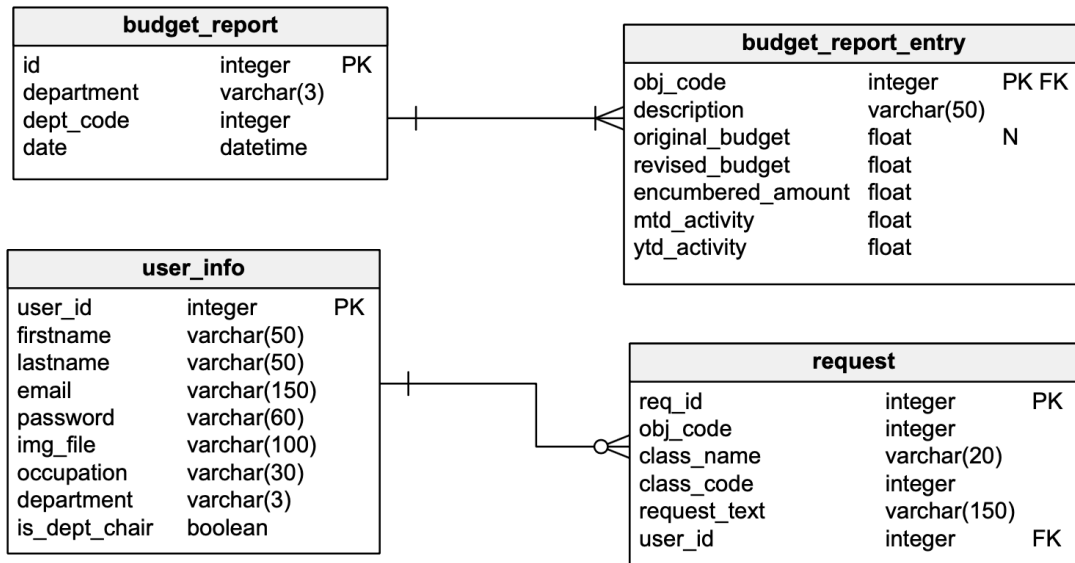


Figure 3.1: An entity-relationship diagram illustrating the working data model.

## 3.2 Usage Overview

This section will detail the functional usage of the application as is, referencing the GUI screenshots found in Appendix B. The prototype currently functions as an application where users can register, log in (refer back to Figure 2.2), and edit their basic user information. The goal for registration functionality is to eventually restrict registration to the administrator, who then creates accounts for each faculty member who would be using the budget management system. This is meant to mimic usage of other applications used in education, such as class management software like Moodle or Edline. Furthermore, the system keeps track of user sessions within their browser using Flask's login manager, so a user can do something like close out a tab and return to the application without logging in again. This also allows the restriction of certain pages to only logged in users.

The application's login page (Figure B.1) serves as the “home” page in that it is the immediate page to be loaded. This is because there is really no information accessible through the application that is relevant to anyone who enters the website via URL. Thus, all pages (except login and registration) cannot be viewed until a user is logged in and authenticated. Clicking navigation bar reveal button reveals the link to the registration page (Figure B.2). Then, clicking the registration link then loads the registration page (Figure B.3). As previously stated, the end goal is for all user registration to be handled by an administrator, but this page was implemented as it is useful for development. Entering valid registration information and clicking the submit button then redirects the user back to the login page, with a message flashing indicating a successful registration (Figure B.4). The user can then log in with the email and password they just created. After a successful login, the user will be authenticated and in session. Thus, they have access to extra pages, namely the “Home” and “Account” pages, as well as a link to logout. This is confirmed by the change in navigation bar links after the user has been authenticated (Figure B.5).

The user's session may be terminated at any point by clicking the logout link from the navigation bar. In addition to ending the session, it will redirect the user to the application login page, as shown in Figure B.1.

# Chapter 4

## Future Work

Now, from the previous chapter, it is clear that this application still needs much work before it is complete and ready for deployment. This reasoning is a big motivator for the decision to make it be an open source project. Furthermore, Hendrix students may continue its development, as they can remain in contact with Nora Simmons as well as gain experience (and potential credit!) on a relatively low-level Web development project.

Being an open source application, it is currently distributed as a Github repository with a README file containing instructions on downloading and installing the application package and its dependencies, as well as running the application for testing and debugging. One of the more immediate future goals is to rebuild the application package using Setuptools, a Python library which allows “developers to more easily build and distribute Python packages, especially ones that have dependencies on other packages” [5]. In a similar vein, a system for unit testing needs to be implemented, which will then offer the opportunity for future students contributing by writing unit tests and attempting to find bugs.

Another general goal from future contributors is more custom styles. Currently,

the page styles come from default Bootstrap classes, so implementing more aesthetically pleasing page design and styles needed. This is also a good student task, as it provides an introduction to the front-end aspect of Web development, and how HTML and CSS work together to respectively provide structure and style to a Web page. In addition to new styles, some pages need to be complete, such as the main home page, with the table of budget data, and a brief “about” page.

Finally, the data model in regards to budget information needs to be implemented within the framework, along with front-end functionality. This involves the creation of a special “administrator” user, who has unique access to the database of budget information. Furthermore, the budget requests made by other users should be sent to the administrator, and budget requests which have been confirmed should be reflected within the database, and thus up to date from any browser.

Currently, I am planning to spend the spring semester of 2018 working on some of the more technical and overarching of the aforementioned future goals. I will additionally work with Nora Simmons and the Computer Science Department in ensuring this project is continued, for the benefit of its future users.

# Chapter 5

## Conclusion

In this paper, we discuss the problem statement posed by Nora Simmons, the administrative assistant for the Biology and Psychology Departments at Hendrix College, of a lack of organization of budget information and the plethora of hindrances in viewing this information in its current state. We then discuss her solution idea, a Web application in which department faculty can view that budget information relevant, as well as keeping this budget information updated in real-time.

The original goal of this project was to build a prototype application to fix the problem of access to up to date budget information in academic departments. It instead turned into the beginnings of an open source project, meant to fully encapsulate the idea brought forth by Ms. Simmons. It will be developed and contributed to until ready for deployment by the college and used by Ms. Simmons and her departments for keeping track of real time budget information. Steps will be taken to ensure contributions continue through future students, and enough care and time is taken to deliver a well polished final product.

# Appendix A

## Code Reference

This appendix contains listings of code referenced. Unnecessary code has been removed and replaced with a line of dots.

Listing A.1: Jinja2 Application Layout Template

```
1 <!DOCTYPE html>
2 <html lang="en" dir="ltr">
3   <head>
4     .....
5   </head>
6   <body>
7     .....
8     {% if current_user.is_authenticated %}
9     <li class="nav-item">
10      <a class="nav-item _nav-link" href="{{ _url_for('home') }}">Home</a>
11    </li>
12    <li class="nav-item">
13      <a class="nav-item _nav-link" href="{{ _url_for('account') }}">My Account</a>
14    </li>
15    <li class="nav-item">
16      <a class="nav-item _nav-link" href="{{ _url_for('logout') }}">Logout</a>
17    </li>
18    {% else %}
19    <li class="nav-item">
20      <a class="nav-item _nav-link" href="{{ _url_for('login') }}">Login</a>
21    </li>
22    <li class="nav-item">
23      <a class="nav-item _nav-link" href="{{ _url_for('register') }}">Register</a>
24    </li>
25    {% endif %}
26    .....
27    <main role="main" class="container">
28      .....
29      {% block content %}{% endblock %}
30    </div>
31  </div>
32 </main>
33 .....
34 </body>
35 </html>
```

## Listing A.2: SQLAlchemy User Data Model

```

1  # User ~one -> zero or more~ Request
2  class User(db.Model, UserMixin):
3      id = db.Column(db.Integer, primary_key=True)
4      first_name = db.Column(db.String(50), nullable=False)
5      last_name = db.Column(db.String(50), nullable=False)
6      email = db.Column(db.String(150), unique=True, nullable=False)
7      password = db.Column(db.String(60), nullable=False)
8      img_file = db.Column(db.String(100), nullable=False, default='default.jpg')
9      occupation = db.Column(db.String(30), nullable=False)
10     department = db.Column(db.String(3), nullable=False)
11     is_dept_chair = db.Column(db.Boolean, nullable=False, default=False)
12     requests = db.relationship('Request', backref='requester', lazy=True)
13
14     def __repr__(self):
15         return f"User('{self.last_name}', '{self.first_name}', '{self.email}')"

```

## Listing A.3: Account Update Form implemented from a Flask-WTF Form

```

1  class UpdateAccountForm(FlaskForm):
2      first_name = StringField('First name', validators=[DataRequired(),
3                                                         Length(min=1, max=50)])
4      last_name = StringField('Last name', validators=[DataRequired(),
5                                                         Length(min=1, max=50)])
6      email = StringField('Email', validators=[DataRequired(), Email()])
7      img = FileField('Update profile picture',
8                      validators=[FileAllowed(['jpg', 'jpeg', 'png'])])
9      occupation = StringField('Occupation', validators=[DataRequired()])
10     department = SelectField('Department',
11                              choices=[('psy', 'PSYCH'), ('bio', 'BIO'), ('adm', 'ADMIN')])
12     is_dept_chair = BooleanField('Check if you are the department chair')
13     submit = SubmitField('Update')
14
15     def validate_email(self, email):
16         if email.data != current_user.email:
17             user = User.query.filter_by(email=email.data).first()
18             if user: # i.e. if user != None
19                 raise ValidationError(
20                     'Email already in use. Please choose a different email.')

```

## Listing A.4: Jinja2 Account Template and Update Form

```

1  <!-- This says that we are using layout.html as our basic layout -->
2  {% extends "layout.html" %}
3  <!-- This is where the content will go that is defined also in layout.html -->
4  {% block content %}
5      .....
6  <div class="content-section">
7      <form method="POST" action="" enctype="multipart/form-data">
8          .....
9          <!-- First Name -->
10         <div class="form-group">
11             {{ form.first_name.label(class="form-control-label") }}
12
13             <!-- This block handles invalid user input -->
14             {% if form.first_name.errors %}
15                 {{ form.first_name(class="form-control form-control-lg is-invalid") }}
16                 <div class="invalid-feedback">
17                     {% for error in form.first_name.errors %}

```

```

18         <span>{{ error }}</span>
19     {% endfor %}
20 </div>
21 {% else %}
22     {{ form.first_name(class="form-control form-control-lg") }}
23     {% endif %}
24 </div>
25 .....
26 </fieldset>
27 <div class="form-group">
28     {{ form.submit(class="btn btn-lg btn-primary btn-block") }}
29 </div>
30 </form>
31 </div>
32 {% endblock content %}

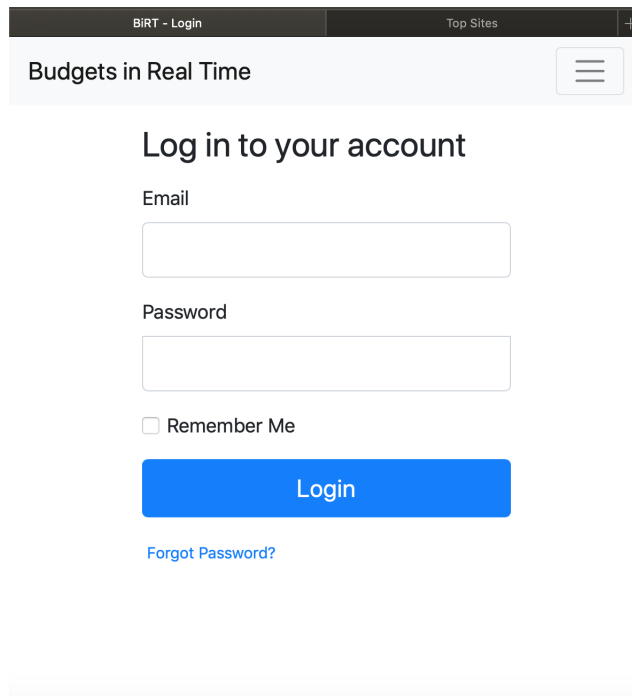
```



# Appendix B

## GUI Reference

This appendix contains screenshots of the application GUI (graphical user interface) referenced.



The screenshot shows a web application interface. At the top, there is a dark header bar with "BIRT - Login" on the left and "Top Sites" on the right, followed by a plus sign icon. Below the header, there is a light gray bar with the text "Budgets in Real Time" and a hamburger menu icon on the right. The main content area is white and contains the heading "Log in to your account". Below this heading are two input fields: "Email" and "Password". Below the "Password" field is a checkbox labeled "Remember Me". Below the checkbox is a blue button with the text "Login". Below the button is a link that says "Forgot Password?".

Figure B.1: Application login page.

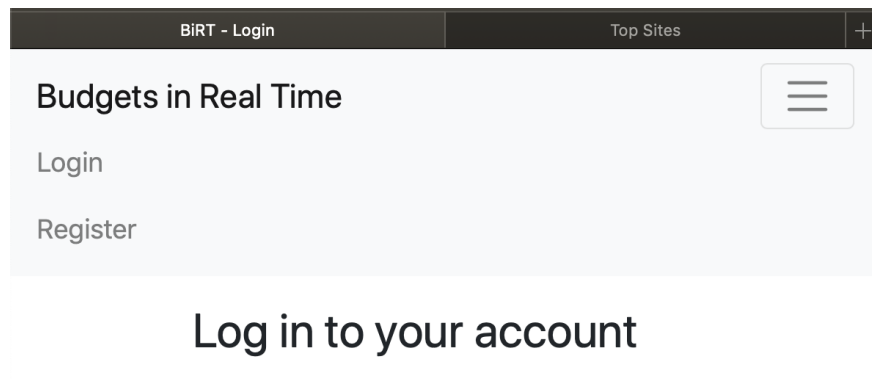


Figure B.2: Navigation bar if there is no authenticated user.

The image shows the registration page of the web application. The header is dark with 'BiRT - Register' on the left, 'Top Sites' in the center, and a '+' icon on the right. Below the header, the main content area has a light gray background. On the left, there are links for 'Login' and 'Register'. The main heading 'Log in to your account' is centered below the links. The registration form is on the right, with fields for 'First name', 'Last name', 'Email', 'Occupation', 'Department' (a dropdown menu with 'PSYCH' selected), 'Check if registree is department chair' (a checkbox), 'Password', and 'Confirm Password'. A blue 'Sign Up' button is at the bottom of the form. Below the button, there is a link: 'Already have an account? [Sign in.](#)'

Figure B.3: Registration page.

The screenshot shows a web application interface for 'BIRT - Login'. At the top, there is a dark header bar with 'BIRT - Login' on the left, 'Top Sites' in the center, and a '+' icon on the right. Below the header, the main content area has a light gray background. A green notification box at the top of the main area displays the message 'Account created for Callahan Hirrel.'. Below this, the heading 'Log in to your account' is centered. Under the heading, there are two input fields: 'Email' and 'Password'. Below the 'Password' field is a checkbox labeled 'Remember Me'. A blue 'Login' button is positioned below the checkbox. At the bottom of the login section, there is a blue link that says 'Forgot Password?'. On the right side of the main content area, there is a hamburger menu icon.

Figure B.4: Redirect after successful registration.

The screenshot shows a web application interface for 'BIRT - Budgets in Real Time Home'. At the top, there is a dark header bar with 'BIRT - Budgets in Real Time Home' on the left, 'Top Sites' in the center, and a '+' icon on the right. Below the header, the main content area has a light gray background. A navigation bar is visible, containing the following links: 'Home', 'My Account', and 'Logout'. On the right side of the navigation bar, there is a hamburger menu icon.


Figure B.5: Navigation bar for authenticated users.

Budgets in Real Time

[Home](#) [My Account](#) [Logout](#)

Account successfully updated.

### Account details for Callahan Hirrel



Email: callahan.hirrel@testing.com

#### Update Account

First name

Callahan

Last name

Hirrel

Email

callahan.hirrel@testing.com

Occupation

student

Department

ADMIN

Check if you are the department chair

☐

Update profile picture

Choose File

no file selected

Update

Figure B.6: Account page after updating user image.

# References

- [1] Bootstrap documentation. <https://getbootstrap.com/docs/4.1/getting-started/introduction/>. Accessed: 13 November 2018.
- [2] code.snippets/python/flask\_blog. [https://github.com/CoreyMSchafer/code\\_snippets/tree/master/Python/Flask\\_Blog](https://github.com/CoreyMSchafer/code_snippets/tree/master/Python/Flask_Blog). Accessed: 13 November 2018.
- [3] Jinja documentation. <http://jinja.pocoo.org/>. Accessed: 19 November 2018.
- [4] Key features of sqlalchemy. <https://www.sqlalchemy.org/features.html>. Accessed: 13 November 2018.
- [5] Setuptools documentation. <https://setuptools.readthedocs.io/en/latest/setuptools.html>. Accessed: 04 December 2018.
- [6] Wtforms documentation. <https://wtforms.readthedocs.io/en/stable/>, 2010. Accessed: 13 November 2018.
- [7] Craig S. Mullins. Dbal: Development versus production. <https://datatechnologytoday.wordpress.com/2013/10/15/dba-development-versus-production/>, 2013. Accessed: 13 November 2018.