Interim Project Plan
Phase 1
CS 4376.0U1
Team 1
Team URL: https://cs-4376-cyberminer.herokuapp.com/
Rotating Leader: Darrien Kramer

Name	Student ID	Email	% of Contribution	Signature
Areebah Fatima	AXF190025	AXF190025@utdallas.edu	16.67%	fatima
Tyler Hargreaves	TTH150630	tth150630@utdallas.edu	16.67%	
Darrien Kramer	dlk210000	dlk210000@utdallas.edu	16.67%	
Ilhaam Syed	IXS180013	ixs180013@utdallas.edu	16.67%	Ilhaam
Nathan Heindl	NJH180002	NJH180002@utdallas.edu	16.67%	nathan
Matthew Bedford	MDB190007	mdb190007@utdallas.edu	16.67%	Matthew Bedford

Meetings:

1. Saturday June 10th @1:00 PM, ECSW Lobby & Remote (**All team members** created diagrams and discussed implementation details) (**Agenda:** Go over implementation details to clear up confusion among members, discuss major components of the system, draw rough drafts of diagrams) (**Summary:** Ilhaam responsible for use case diagrams, Tyler & Areebah & Nathan responsible for class diagrams & sequence diagrams)

1. Introduction

1.1 Project overview

The following document will describe the planning, scheduling, and team organization involved in implementing the Cyberminer web search engine. The project's end goal is to create a system that will provide users with relevant information using the keywords entered by the user. The resulting data provided to the end user will be a sorted and filtered list of web page URLs.

The major components of the search engine will include a search interface, indexing system, result filtering, query processing, etc. The search interface will allow the end user to interact with the system, enter search queries, and retrieve information. The indexing system will organize, store, and rank information to allow for a reasonably fast search. Result filtering will ensure that our system provides users with relevant, up-to-date data in sorted order. Finally, the query processing component of the project will be responsible for interpreting user inputs, identifying keywords in said input, performing index lookup, and retrieving results.

1.2 Project deliverables

A)	Preliminary Project Plan	06/01/2023
B)	Interim Project	06/15/2023
C)	Final Project I Submission	06/29/2023
D)	Interim Project II	07/13/2023
E)	Final Project II Submission	08/01/2023

1.3 Evolution of this document

Revision History

Who	**When**	**Changes**
Tyler Hargreaves	May 29th	Started document
Areebah Fatima	May 30th	Began Preliminary Documentation; Wrote project description, deliverables, etc.
Nathan Heindl	May 30th	Wrote project responsibilities and management priorities
Tyler Hargreaves	June 11th	Updated to reflect recent decisions and architecture changes

1.4 References

I.	Team Source Code Website	https://github.com/tyharg/CS-4376
II.	Team Demo Website	https://cs-4376-cyberminer.herokuapp.com/
III.	Course Homepage	https://personal.utdallas.edu/~chung/OOD/syllabus.htm
IV.	Getting Started with Rails	https://guides.rubyonrails.org/getting_started.html

Cited References

[1] Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The Unified Modeling Language User Guide*. Addison-Wesley.

1.5 Definitions, acronyms, and abbreviations

UML: Unified Modeling Language
CI/CD: Continuous Integration / Continuous Delivery

2. Project organization

2.1 Process model

2.2 Organizational structure

Team Members: Tyler Hargreaves, Darrien Kramer, Ilhaam Syed, Nathan Heindl, Areebah Fatima, Matthew Bedford

Deliverable	**Team Leader**
Preliminary Project Plan	Tyler Hargreaves
Interim Project	Darrien Kramer

Final Project I Submission	Ilhaam Syed
Interim Project II	Nathan Heindl
Final Project II Submission	Matthew Bedford
Project Phase I and II Presentation	Areebah Fatima

2.3 Organizational boundaries and interfaces

2.4 Project responsibilities

Every member will be involved in both of the projects main two life cycles. Team leaders are specifically to turn in work, keep workflow on track, and organize meetings. For more specific responsibilities they will be discussed at a later date.

3. Managerial process

3.1 Management objectives and priorities

The team leaders are to help manage meetings, turn in deliverables, and keep everyone up to date on the project's progression. If sub groups are used then it's the team leader's responsibility to make sure both teams have what they need to complete their work.

3.2 Assumptions, dependencies, and constraints

Because we are implementing a long-term-support version of Ruby on Rails, we will be operating under the presumption that HTTP dispatching, database access, and other core functionality is handled appropriately. This design philosophy allows us to concern ourselves with engineering the search algorithm and other important application-specific features.

The application is dependent on Rails and its own dependencies. A full list of requirements can be found at: https://github.com/tyharg/CS-4376/blob/main/Gemfile

3.3 Risk management

The project will be hosted on Heroku and will implement a basic CI/CD pipeline to ensure that deployed code will have a high probability of accuracy.

3.4 Monitoring and controlling mechanism

4. Technical process

4.1 Methods, tools, and techniques

The Creately workspace will be the modeling tool our team will use to create our Use Case, Class, and Sequence Diagrams. The programming language our team has agreed to for the project is Ruby, and we will be utilizing the Rails framework to handle HTTP requests and database access. We will additionally be using various packages such as Pry and AppMap to analyze our program and create diagrams.

Our team will use the following tools to communicate: the Discord social platform, Google Docs, and Microsoft Teams. In addition to these communication tools, our team will host in-person meetings when needed.

4.2 Software documentation

The application aims to utilize automatically generated documentation wherever possible. This is done to avoid the achilles heel of outdated documentation.

- Main readme: https://github.com/tyharg/CS-4376/tree/main
- User manual: To be created

4.3 Project support functions

- Rails testing infrastructure: https://guides.rubyonrails.org/testing.html

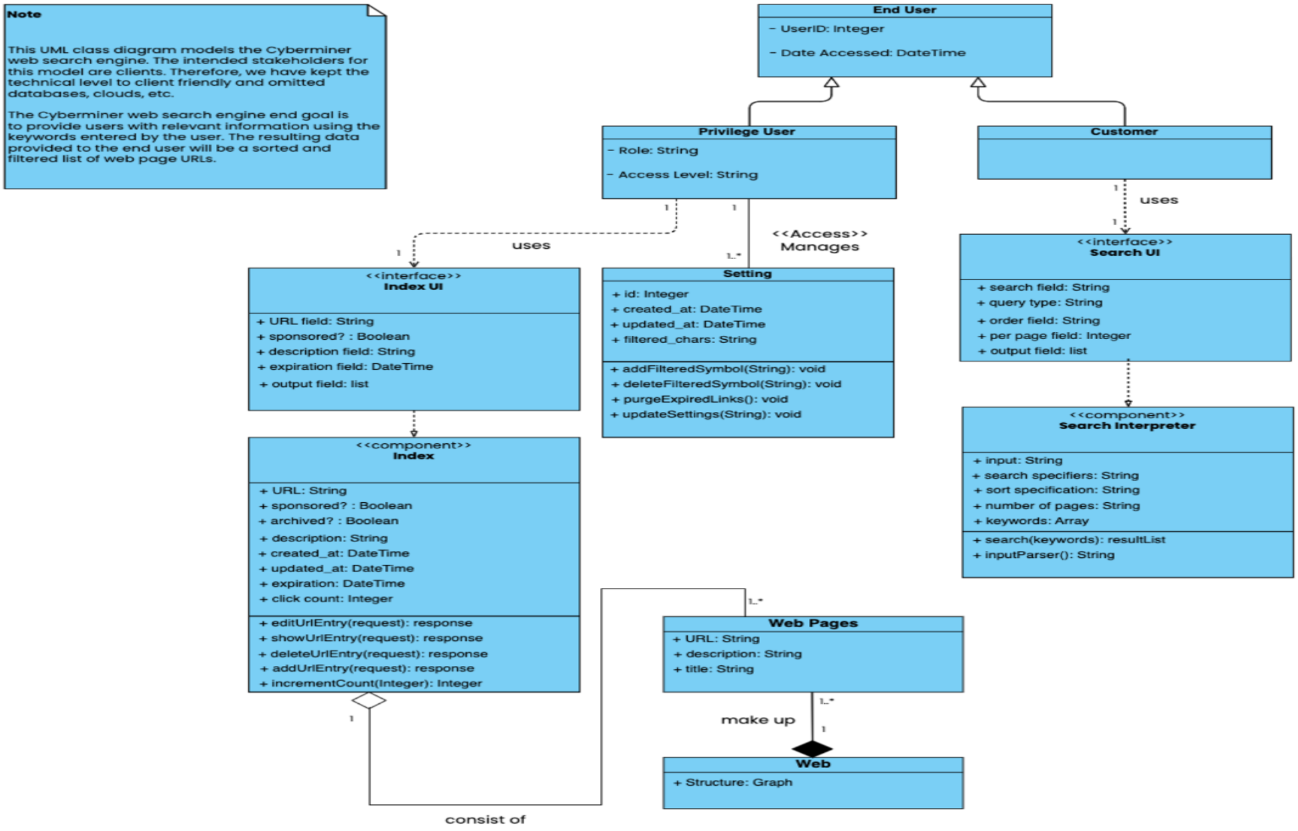
5. Work elements, schedule, and budget

This project is scheduled to be completed by August 1st, 2023. Listed below is the project deliverable due date

Deliverable	**Due By**
Preliminary Project	June 1, 2023
Interim Project I	June 15, 2023
Final Project I Submission	June 29, 2023
Interim project II	July 13, 2023
Final Project II Submission	August 1, 2023

6. Diagram

Stakeholder Class Diagram: Client

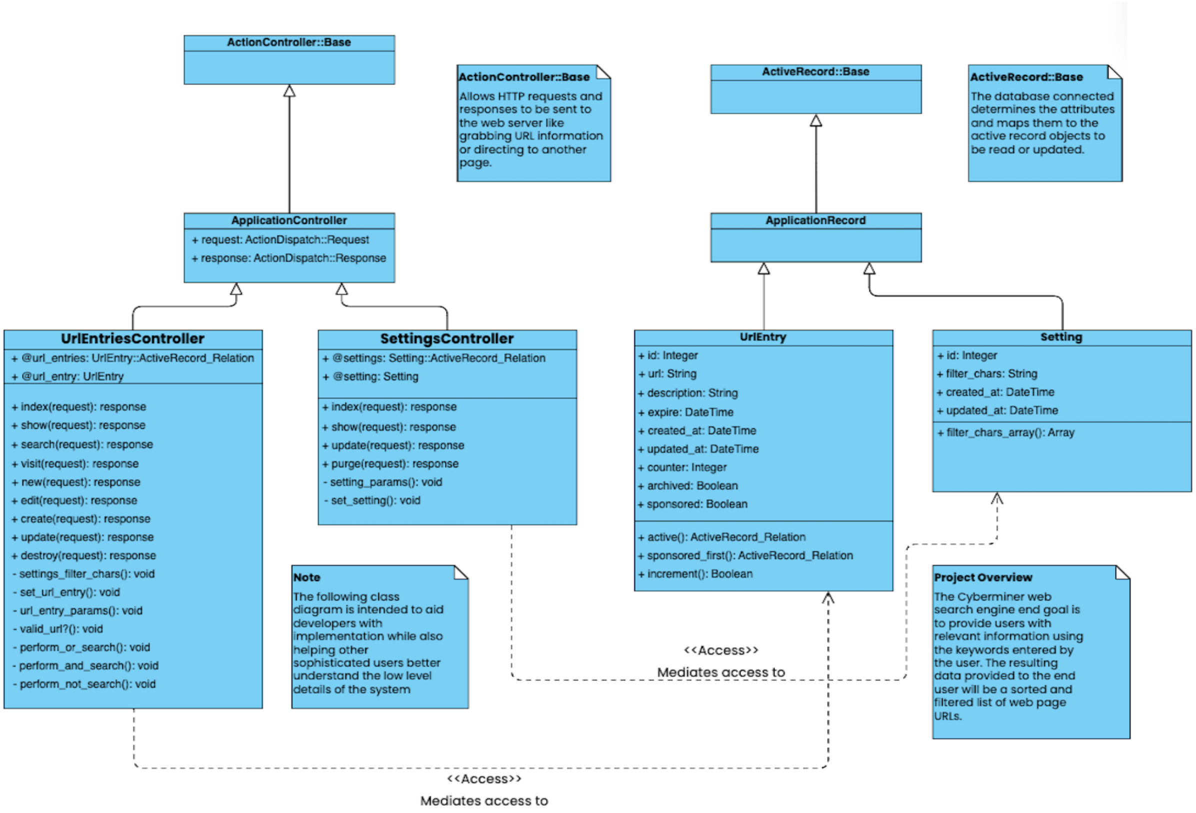


Note

This UML class diagram models the Cyberminer web search engine. The intended stakeholders for this model are clients. Therefore, we have kept the technical level to client friendly and omitted databases, clouds, etc.

The Cyberminer web search engine end goal is to provide users with relevant information using the keywords entered by the user. The resulting data provided to the end user will be a sorted and filtered list of web page URLs.

Stakeholder Class Diagram: Developers



ActionController::Base
Allows HTTP requests and responses to be sent to the web server like grabbing URL information or directing to another page.

ActiveRecord::Base
The database connected determines the attributes and maps them to the active record objects to be read or updated.

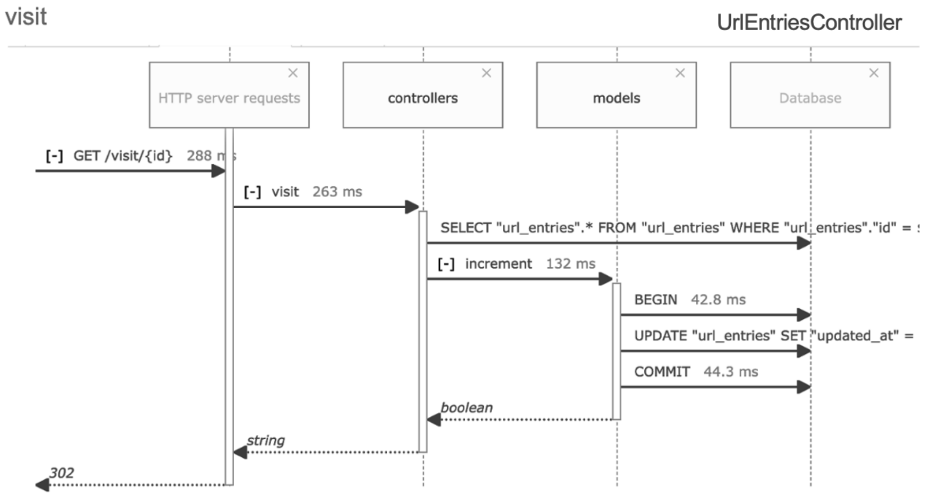
Note
The following class diagram is intended to aid developers with implementation while also helping other sophisticated users better understand the low level details of the system

Project Overview
The Cyberminer web search engine end goal is to provide users with relevant information using the keywords entered by the user. The resulting data provided to the end user will be a sorted and filtered list of web page URLs.

ActionController::Base
ActiveRecord::Base
ApplicationController
+ request: ActionDispatch::Request
+ response: ActionDispatch::Response
ApplicationRecord
UrlEntriesController
+ @url_entries: UrlEntry::ActiveRecord_Relation
+ @url_entry: UrlEntry
+ index(request): response
+ show(request): response
+ search(request): response
+ visit(request): response
+ new(request): response
+ edit(request): response
+ create(request): response
+ update(request): response
+ destroy(request): response
- settings_filter_chars(): void
- set_url_entry(): void
- url_entry_params(): void
- valid_url?(): void
- perform_or_search(): void
- perform_and_search(): void
- perform_not_search(): void
SettingsController
+ @settings: Setting::ActiveRecord_Relation
+ @setting: Setting
+ index(request): response
+ show(request): response
+ update(request): response
+ purge(request): response
- setting_params(): void
- set_setting(): void
UrlEntry
+ id: Integer
+ url: String
+ description: String
+ expire: DateTime
+ created_at: DateTime
+ updated_at: DateTime
+ counter: Integer
+ archived: Boolean
+ sponsored: Boolean
+ active(): ActiveRecord_Relation
+ sponsored_first(): ActiveRecord_Relation
+ increment(): Boolean
Setting
+ id: Integer
+ filter_chars: String
+ created_at: DateTime
+ updated_at: DateTime
+ filter_chars_array(): Array
<<Access>>
Mediates access to
<<Access>>
Mediates access to

Sequence Diagrams:

URLEntriesController Diagrams

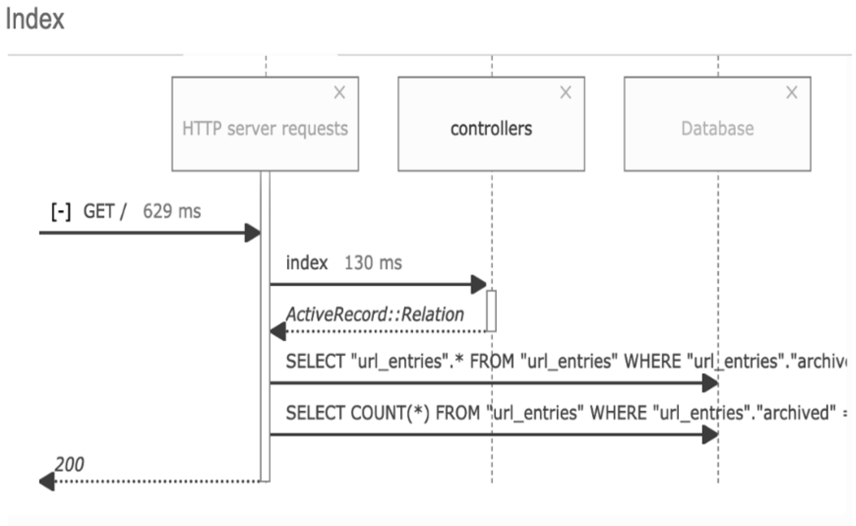


```
SELECT "url_entries".* FROM "url_entries" WHERE "url_entries"."id" = $1 LIMIT $2

UPDATE "url_entries" SET "updated_at" = $1, "counter" = $2 WHERE "url_entries"."id" = $3
```

NOTE

The following digram depicts the sequence of events that occur when the visit method in the UrlEntriesController class is invoked. The job of the visit method is to redirect users to their desired web page


Index
HTTP server requests
controllers
Database
[-] GET / 629 ms
index 130 ms
ActiveRecord::Relation
SELECT "url_entries".* FROM "url_entries" WHERE "url_entries"."archiv
SELECT COUNT(*) FROM "url_entries" WHERE "url_entries"."archived" =
200

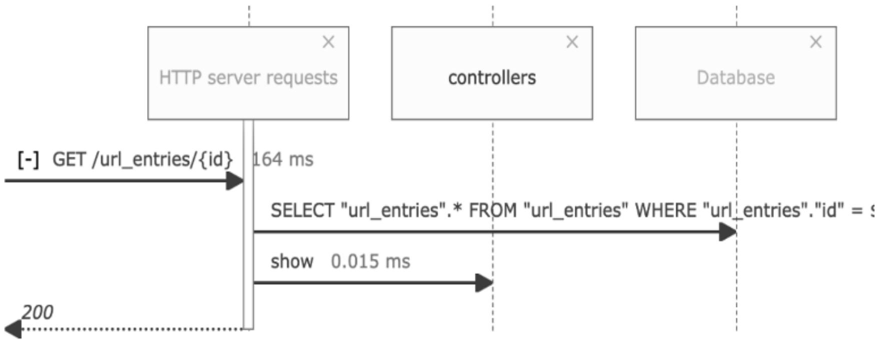
```
SELECT "url_entries".* FROM "url_entries" WHERE "url_entries"."archived" = $1
AND ("url_entries"."sponsored" = $2 OR "url_entries"."sponsored" = $3) ORDER
BY "url_entries"."sponsored" DESC LIMIT $4 OFFSET $5

SELECT COUNT(*) FROM "url_entries" WHERE "url_entries"."archived" = $1 AND
("url_entries"."sponsored" = $2 OR "url_entries"."sponsored" = $3)
```

NOTE

The following digram depicts the sequence of events that occur when the Index method in the UrlEntriesController class is invoked.

show

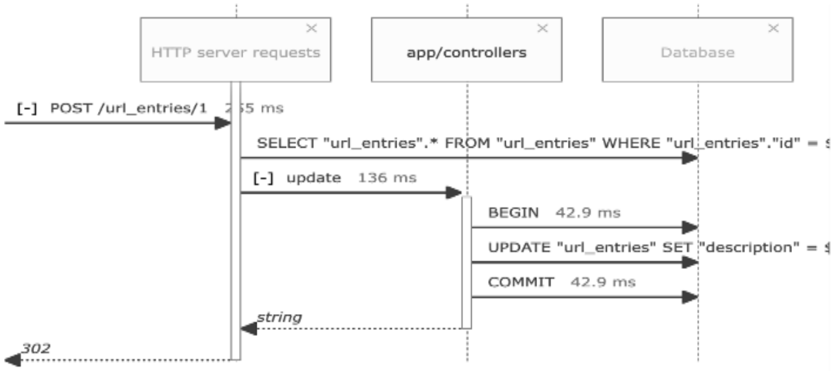

HTTP server requests
controllers
Database
[-] GET /url_entries/{id} 164 ms
SELECT "url_entries".* FROM "url_entries" WHERE "url_entries"."id" = $
show 0.015 ms
200

```
SELECT "url_entries".* FROM "url_entries" WHERE "url_entries"."id" = $1 LIMIT $2
```

NOTE

The following digram depicts the sequence of events that occur when the Show method in the UrlEntriesController class is invoked.

update


HTTP server requests
app/controllers
Database
[-] POST /url_entries/1 2 5 ms
SELECT "url_entries".* FROM "url_entries" WHERE "url_entries"."id" = $
[-] update 136 ms
BEGIN 42.9 ms
UPDATE "url_entries" SET "description" = $
COMMIT 42.9 ms
string
302

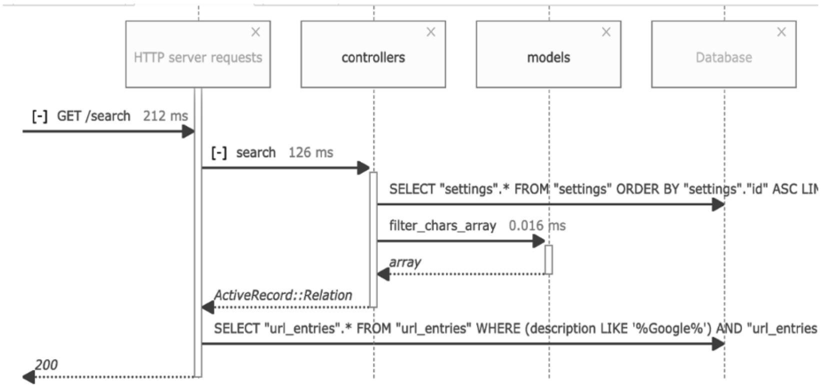
```
SELECT "url_entries".* FROM "url_entries" WHERE "url_entries"."id" = $1 LIMIT $2

UPDATE "url_entries" SET "description" = $1, "updated_at" = $2 WHERE "url_entries"."id"
= $3
```

NOTE

The following digram depicts the sequence of events that occur when the Update method in the UrlEntriesController class is invoked.

search


HTTP server requests
controllers
models
Database
[-] GET /search 212 ms
[-] search 126 ms
SELECT "settings".* FROM "settings" ORDER BY "settings"."id" ASC LIM
filter_chars_array 0.016 ms
array
ActiveRecord::Relation
SELECT "url_entries".* FROM "url_entries" WHERE (description LIKE '%Google%') AND "url_entries
200

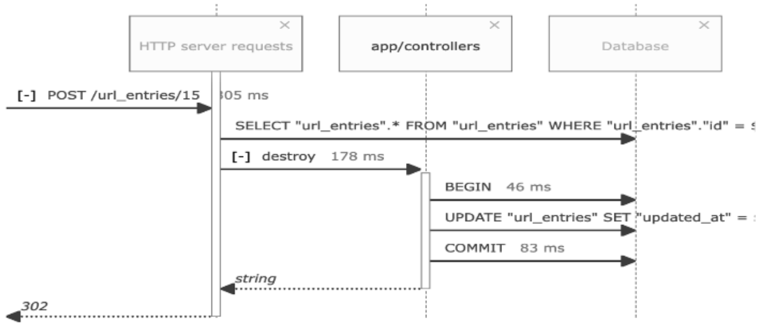
```
SELECT "settings".* FROM "settings" ORDER BY "settings"."id" ASC LIMIT $1

SELECT "url_entries".* FROM "url_entries" WHERE (description LIKE '%%') AND
"url_entries"."archived" = $1 AND ("url_entries"."sponsored" = $2 OR
"url_entries"."sponsored" = $3) ORDER BY "url_entries"."counter" DESC,
"url_entries"."sponsored" DESC LIMIT $4 OFFSET $5
```

NOTE

The following digram depicts the sequence of events that occur when the search method in the UrlEntriesController class is invoked.

destroy

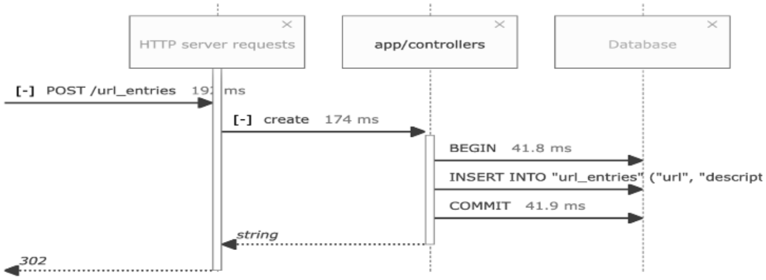

HTTP server requests
app/controllers
Database
[-] POST /url_entries/15 05 ms
SELECT "url_entries".* FROM "url_entries" WHERE "url_entries"."id" = $
[-] destroy 178 ms
BEGIN 46 ms
UPDATE "url_entries" SET "updated_at" = :
COMMIT 83 ms
string
302

```
SELECT "url_entries".* FROM "url_entries" WHERE "url_entries"."id" = $1 LIMIT $2
```

NOTE

The following digram depicts the sequence of events that occur when the destroy method in the UrlEntriesController class is invoked.

create



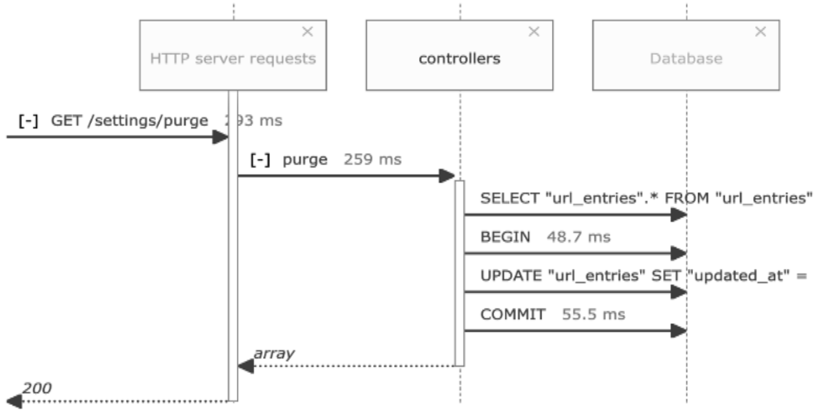
```
INSERT INTO "url_entries" ("url", "description", "expire", "created_at",
"updated_at", "counter", "archived", "sponsored") VALUES ($1, $2, $3, $4, $5, $6,
$7, $8) RETURNING "id"
```

NOTE

The following digram depicts the sequence of events that occur when the create method in the UrlEntriesController class is invoked.

SettingController Diagrams

purge


HTTP server requests
controllers
Database
[-] GET /settings/purge 93 ms
[-] purge 259 ms
SELECT "url_entries".* FROM "url_entries"
BEGIN 48.7 ms
UPDATE "url_entries" SET "updated_at" =
COMMIT 55.5 ms
array
200

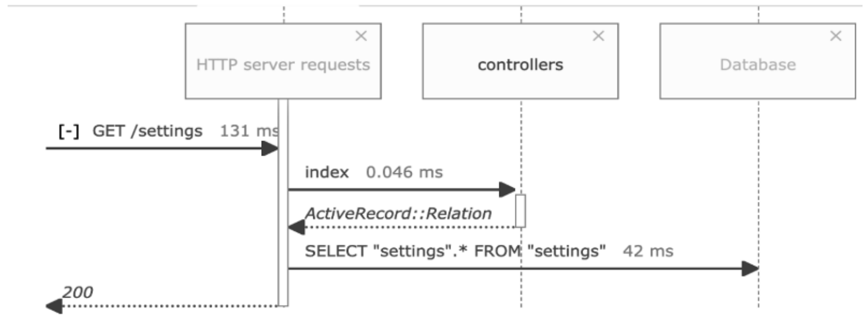
```
SELECT "url_entries".* FROM "url_entries" WHERE "url_entries"."archived" = $1 AND (expire <
'2023-06-07 18:46:55.172212')

UPDATE "url_entries" SET "updated_at" = $1, "archived" = $2 WHERE "url_entries"."id" = $3
```

NOTE

The following digram depicts the sequence of events that occur when the purge method in the SettingController class is invoked.

index

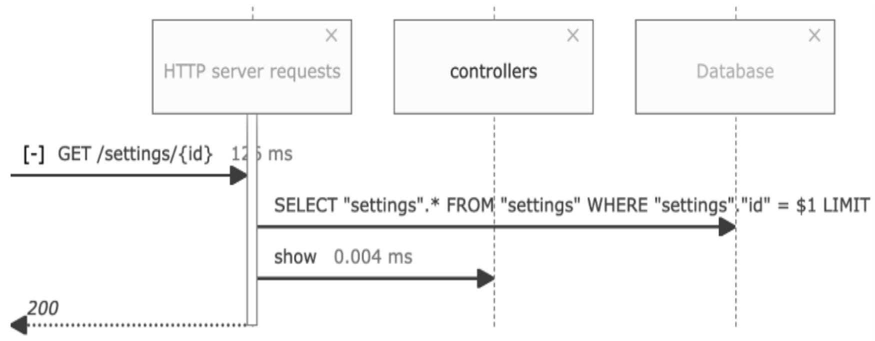


```
SELECT "settings".* FROM "settings"
```

NOTE

The following digram depicts the sequence of events that occur when the index method in the SettingController class is invoked.

show

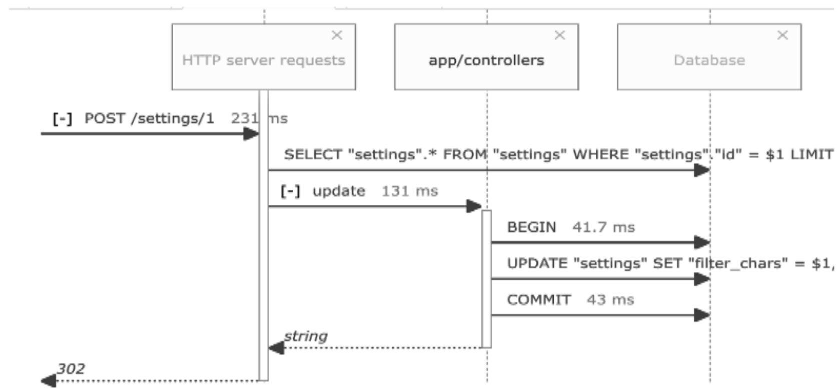

HTTP server requests
controllers
Database
[-] GET /settings/{id} 125 ms
SELECT "settings".* FROM "settings" WHERE "settings"."id" = $1 LIMIT
show 0.004 ms
200

```
SELECT "settings".* FROM "settings" WHERE "settings"."id" = $1 LIMIT $2
```

NOTE

The following digram depicts the sequence of events that occur when the show method in the SettingController class is invoked.

update


HTTP server requests
app/controllers
Database
[-] POST /settings/1 231 ms
SELECT "settings".* FROM "settings" WHERE "settings"."id" = $1 LIMIT
[-] update 131 ms
BEGIN 41.7 ms
UPDATE "settings" SET "filter_chars" = $1,
COMMIT 43 ms
string
302

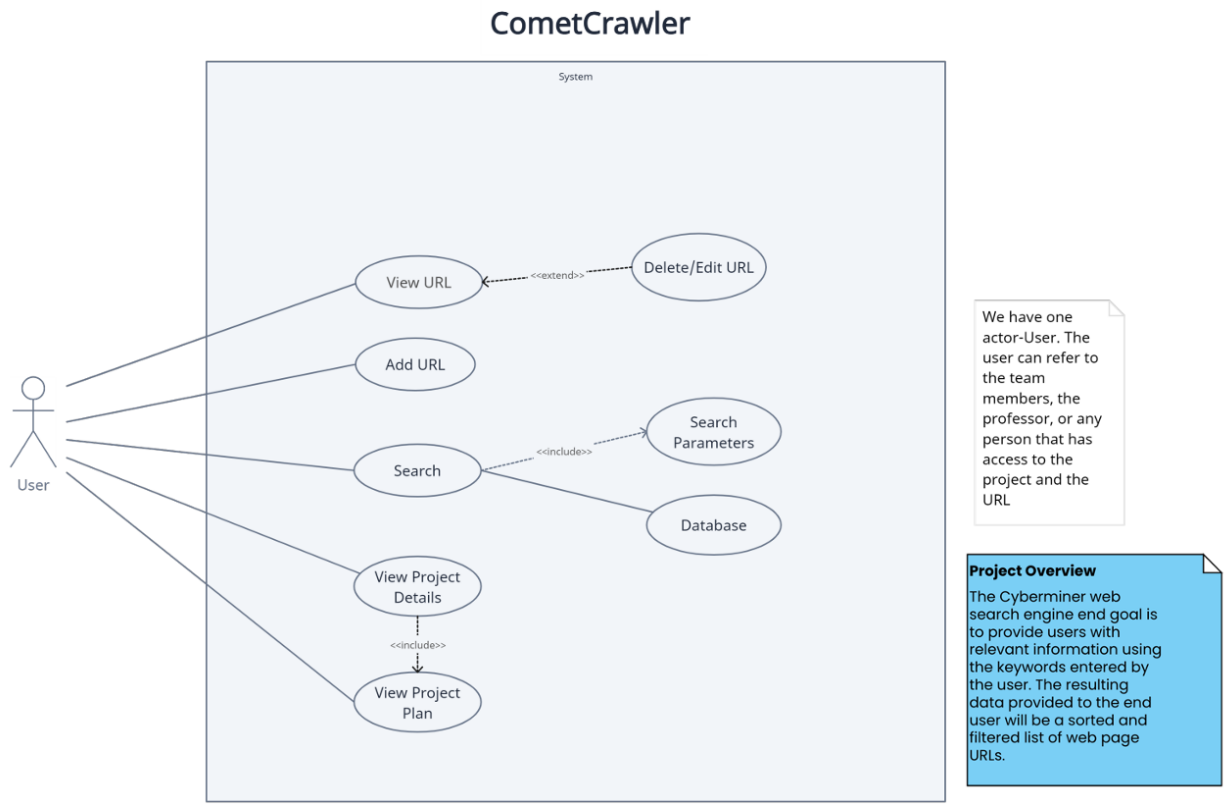
```
SELECT "settings".* FROM "settings" WHERE "settings"."id" = $1 LIMIT $2

UPDATE "settings" SET "filter_chars" = $1, "updated_at" = $2 WHERE
"settings"."id" = $3
```

NOTE

The following digram depicts the sequence of events that occur when the update method in the SettingController class is invoked.

Use Case Diagram:

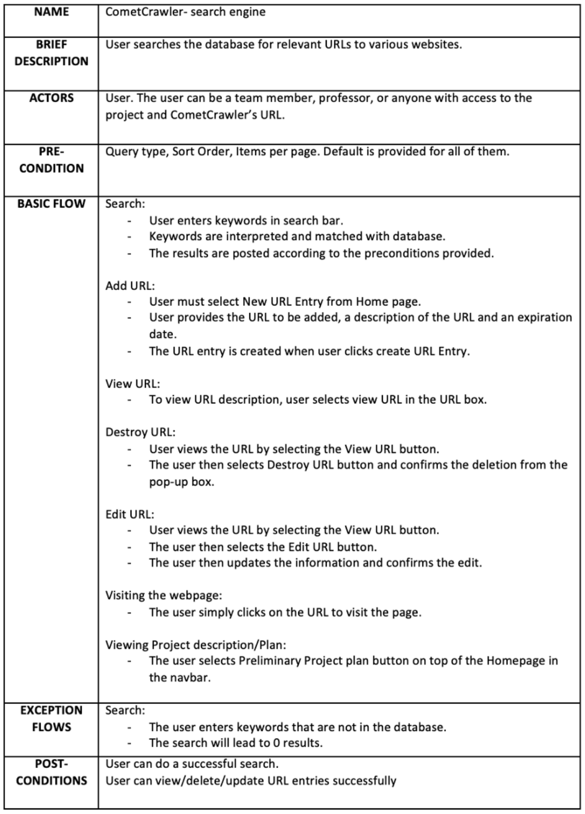


We have one actor-User. The user can refer to the team members, the professor, or any person that has access to the project and the URL

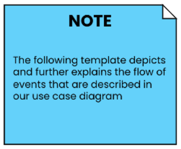
Project Overview

The Cyberminer web search engine end goal is to provide users with relevant information using the keywords entered by the user. The resulting data provided to the end user will be a sorted and filtered list of web page URLs.

Use Case Template:

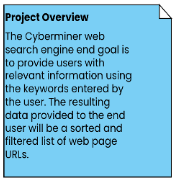


NAME	CometCrawler- search engine
BRIEF DESCRIPTION	User searches the database for relevant URLs to various websites.
ACTORS	User. The user can be a team member, professor, or anyone with access to the project and CometCrawler's URL.
PRE-CONDITION	Query type, Sort Order, Items per page. Default is provided for all of them.
BASIC FLOW	Search: - User enters keywords in search bar. - Keywords are interpreted and matched with database. - The results are posted according to the preconditions provided. Add URL: - User must select New URL Entry from Home page. - User provides the URL to be added, a description of the URL and an expiration date. - The URL entry is created when user clicks create URL Entry. View URL: - To view URL description, user selects view URL in the URL box. Destroy URL: - User views the URL by selecting the View URL button. - The user then selects Destroy URL button and confirms the deletion from the pop-up box. Edit URL: - User views the URL by selecting the View URL button. - The user then selects the Edit URL button. - The user then updates the information and confirms the edit. Visiting the webpage: - The user simply clicks on the URL to visit the page. Viewing Project description/Plan: - The user selects Preliminary Project plan button on top of the Homepage in the navbar.
EXCEPTION FLOWS	Search: - The user enters keywords that are not in the database. - The search will lead to 0 results.
POST-CONDITIONS	User can do a successful search. User can view/delete/update URL entries successfully



NOTE

The following template depicts and further explains the flow of events that are described in our use case diagram



Project Overview

The Cyberminer web search engine end goal is to provide users with relevant information using the keywords entered by the user. The resulting data provided to the end user will be a sorted and filtered list of web page URLs.