SE/CS/CE 3354

Software Engineering

Course Team Project Specifications (Part IV)

BiteScout

Deliverable #1 [All group tasks]

- Diagrams (Youngjun Ryoo, Shayaan Zari, Ridham Gabani, Rami Kamel)
- Architectural design (Ridham Gabani, Shayaan Zari, Khaja Yazdaan)
- Setup Github Repository (Shayaan Zari)
- Make the first commit (Mark Armstrong)
- Project Scope (Sammy Bustos, Youngjun Ryoo, Mark Armstrong, Rami Kamel)
- Project deliverable report #1 (Everyone)

GitHub Repository: https://github.com/ShayaanZari/3354-BiteScout

Feedback on Proposal

Our Course Team Project Proposal met all the requirements for our client, and the client was satisfied with it.

Software Process Model

For the development of BiteScout, we adopted the incremental process model. This model was selected due to the evolving nature of the project's requirements, particularly the need to gradually introduce features such as dietary filter, allergy preferences, and restaurant recommendations based on user location. The incremental approach would allow us to develop the system in functional segments, ensuring that each module could be designed, implemented, and tested independently before integration.

Functional Requirements

The following functional requirements define the main user and system interactions for BiteScout, a restaurant recommendation and management platform designed to connect users, restaurant owners, delivery partners, and administrators.

User Functions

- The system must allow users to create and manage accounts securely.
- The system shall allow users to set preferences (such as cuisine type, distance, dietary restrictions, and alcohol availability).
- The system must allow users to fetch nearby restaurants based on location and preferences.
- The system must allow users to get directions to selected restaurants.
- The system shall allow users to submit and view reviews for restaurants.

Non-functional Requirements

1. **Product Requirements**

- **Efficiency Requirements**: The website shall load the home page in under 2 seconds on a standard 4G connection. The search algorithm shall return restaurant recommendations within 3 seconds of a user submitting their filters.
- **Dependability Requirements**: The application shall have a monthly uptime of 90%, excluding planned maintenance. The system shall gracefully handle errors from third-party services without crashing.
- **Security Requirements**: All user data must be transmitted over HTTPS. User passwords shall be hashed and salted before storage. The system shall be protected against SQL Injection and XSS attacks.
- **Usability Requirements**: A new user shall be able to understand how to use the main search feature without instruction. The interface shall be intuitive with clear filters. The design shall be responsive across mobile, tablet, and desktop devices.

- **Performance Requirements**: The website shall support up to 1,000 concurrent users during peak hours (Friday and Saturday evenings) without significant performance degradation.

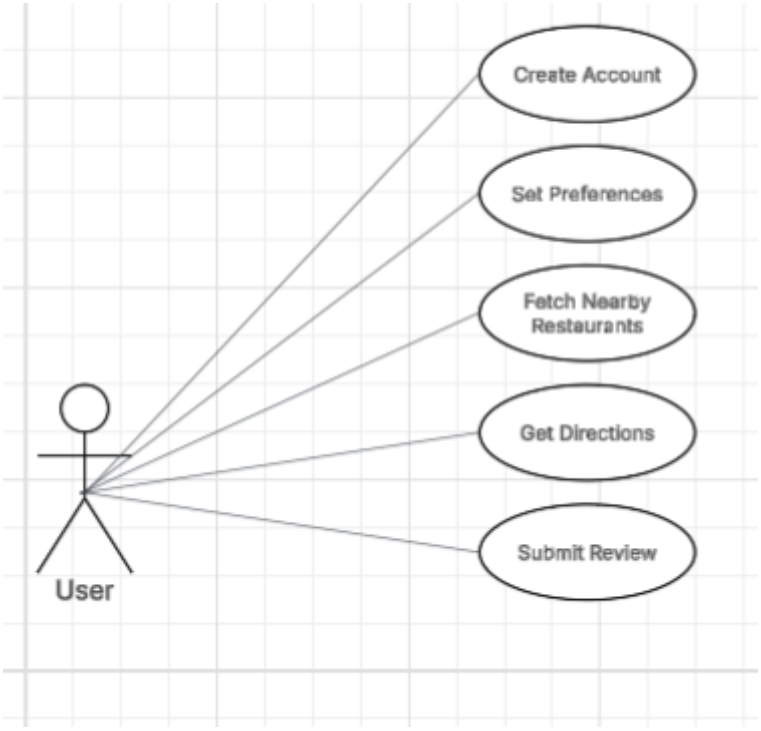
2. **Organizational Requirements**

- **Operational Requirements**: The application shall be deployed using a cloud-based containerization service. The system shall integrate with a logging and error-monitoring service to alert the development team of critical issues.
- **Development Requirements**: The backend shall be developed in Python using Flask, and the frontend shall use HTML and JavaScript. The source code shall be managed using Git and hosted on GitHub.

3. **External Requirements**

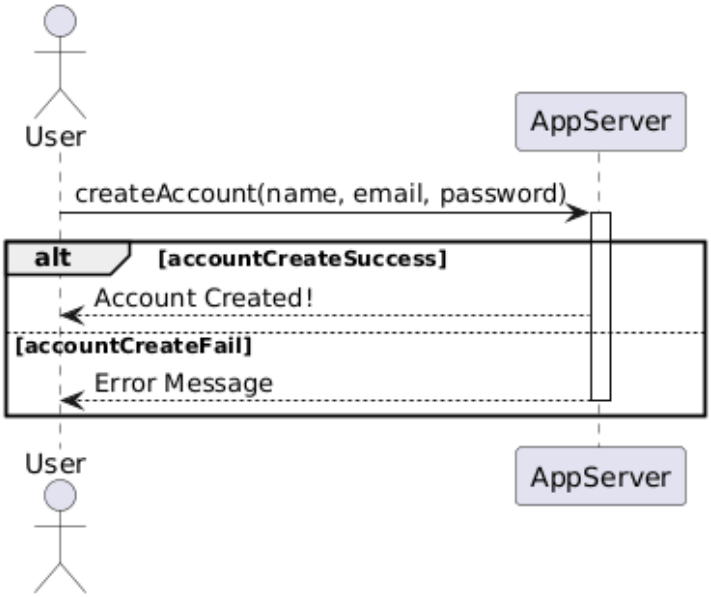
- **Regulatory Requirements**: Not applicable for the initial version as no payments are processed. The system shall be designed to allow for future integration of a PCI-DSS compliant payment gateway if needed.
- **Ethical Requirements**: The recommendation algorithm shall be designed to avoid unfair bias against local or new restaurants. User data shall not be sold to third-party advertisers without explicit, opt-in consent.

Use Case

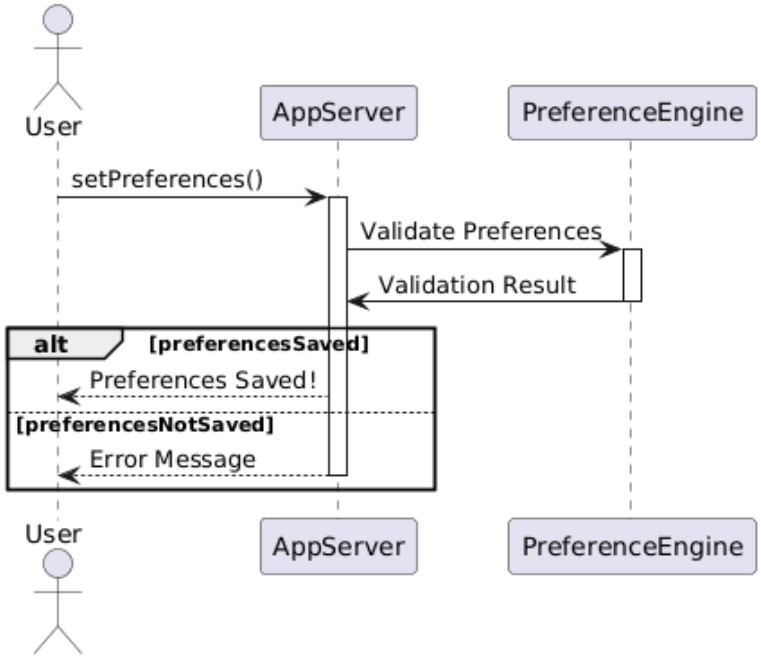


Sequence Diagram

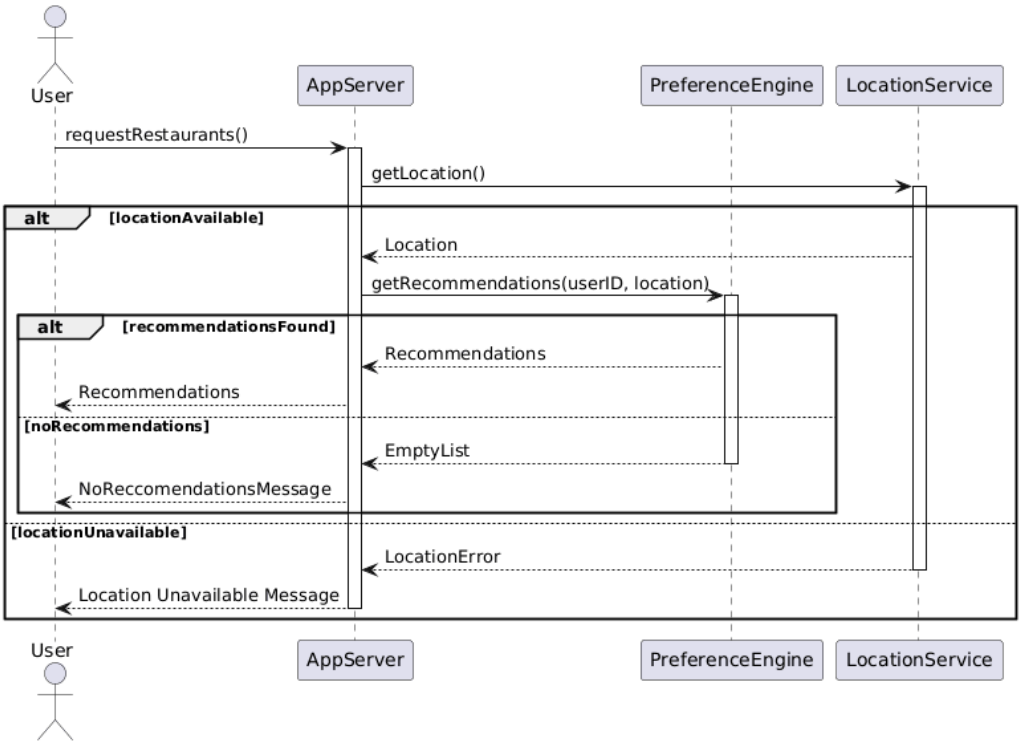
User creating an account



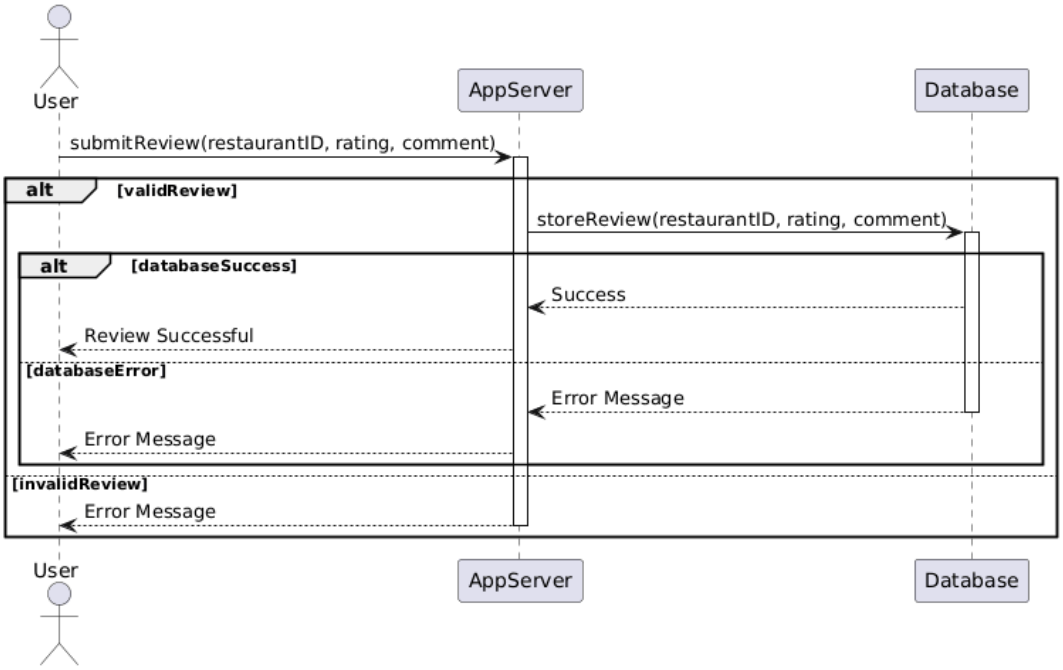
Setting Preferences



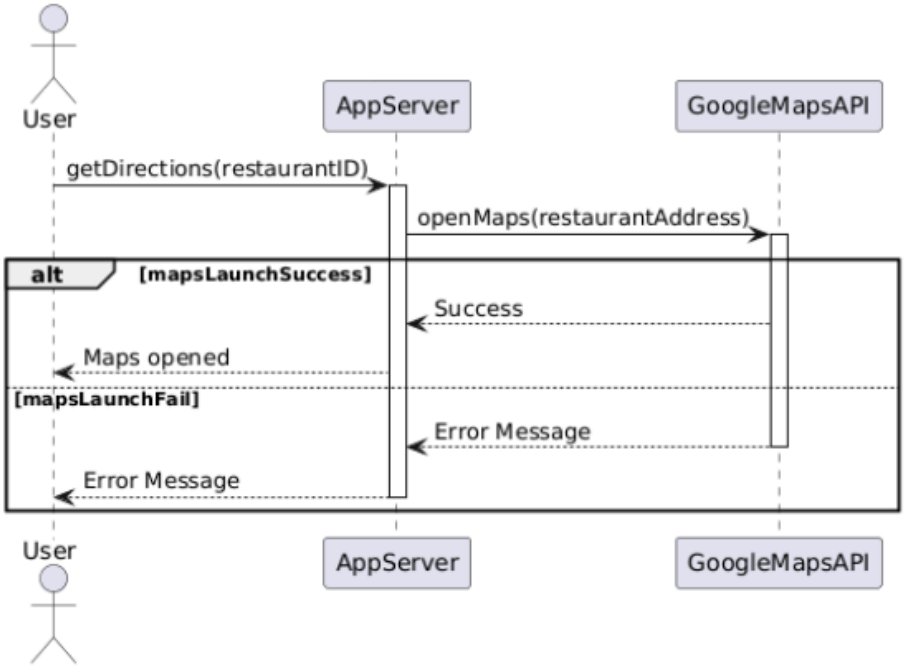
Fetch Recommended Restaurants



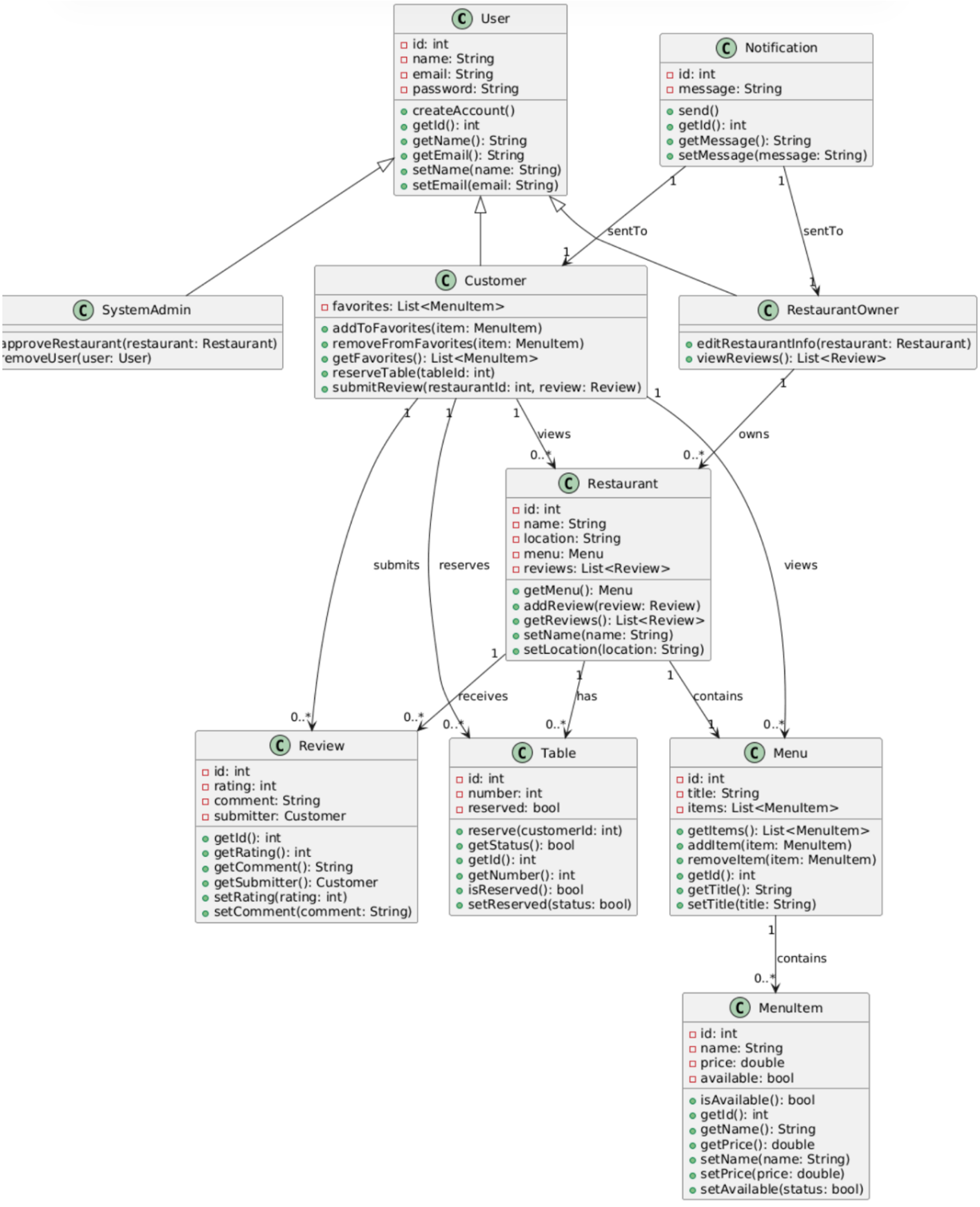
Post Review



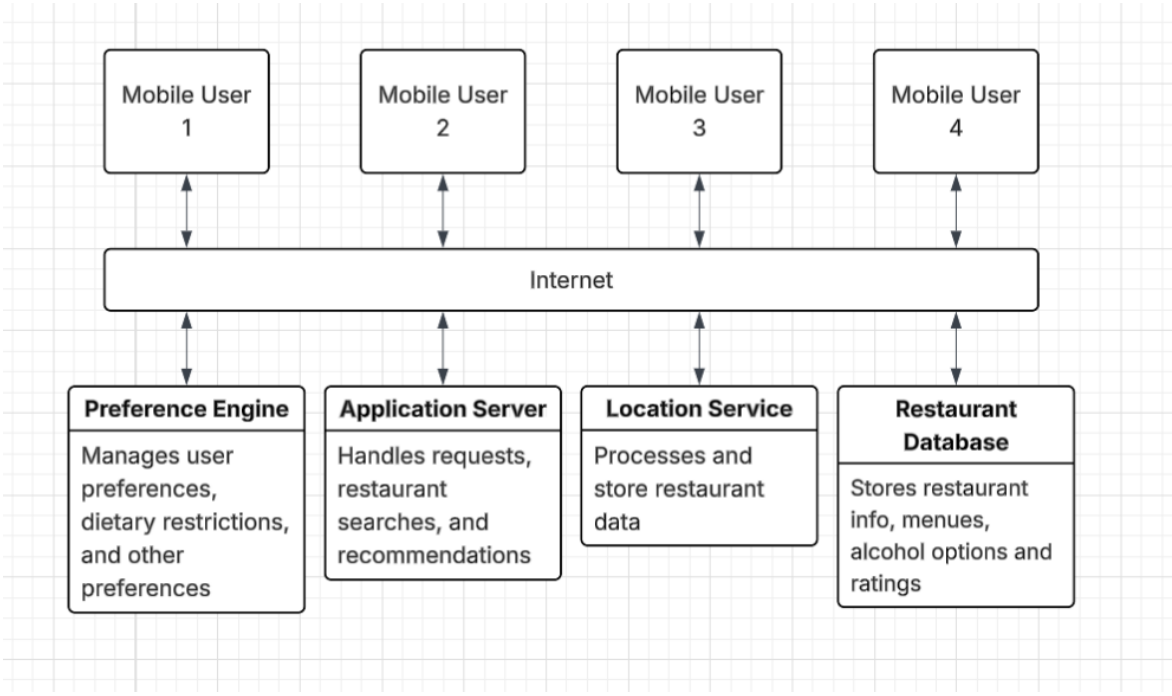
Get Directions



Class Diagram



Architectural Design



Deliverable # 2 [All Group tasks]

- Slides (Everyone)
- Rehearse and record the presentation (All members)
- Project Scheduling (Shayaan Zari)
- Cost, Effort and Pricing estimation (Ridham Gabani, Youngjun Ryoo, Mark Armstrong)
- Comparison / Conclusion / References (Khaja Yazdaan, Sammy Bustos)
- Test plan (Rami Kamel)
- Deliverables #2 report (Everyone)

Project Scheduling, Cost, Effort, and Pricing Estimations

4a. Project Scheduling

Question	**Description**	**Rating (0-5)**
PC1	Does the system require reliable backup and recovery?	5
PC2	Are data communications required?	5
PC3	Are there distributed processing functions?	4
PC4	Is performance critical?	4
PC5	Will the system run in an existing, heavily utilized operational environment?	4
PC6	Does the system require online data entry?	5
PC7	Does the online data entry require the input transaction to be built over multiple screens or operation?	3
PC8	Are the master files updated inline?	4
PC9	Are the inputs, outputs, files, or inquires complex?	3

PC10	Is the internal processing complex?	3
PC11	Is the codes designed to be reusable?	3
PC12	Are the conversion and installation included in the design?	4
PC13	Is the system designed for multiple installations in different organizations?	4
PC14	Is the application designed to facilitate change and ease of use by the user?	5

Summing up the ratings, we have a processing complexity (PC) of 56. Using the formula PCA = 0.65 + 0.01(PC) we get a processing complexity adjustment (PCA) of 1.21.

The function Point (FP) is FP = GFP x PCA = 125 * 1.21 = 151.25 ≈ 152

Based on past projects, the team productivity is 10 FP per person-week.

The estimated effort E is then E = FP / Productivity = 152/10 = 15.2 ≈ 16 person-weeks.

The team size is 7 members, so the Project Duration D is D = E / team size = 16 person-weeks / 7 people = 2.29 weeks ≈ 3 weeks.

Project Timeline

- Proposed Start Date: Monday, January 12, 2026
- End Date: Friday, January 30, 2026
- Duration: 3 weeks (15 working days)
- Working Hours: 40 hours per person per week (8 hours/day)
- Weekends: Excluded from schedule

Team Composition

- DevOps Team: 2 Developers
- Frontend Team: 2 Developers
- Backend Team: 3 Developers

Task Identification and Distribution

Week 1

DevOps Team:

- Day 1–2 (T1): Provision AWS EC2 instances for the Application Server and AWS RDS (PostgreSQL) for the database. Configure Security Groups, VPN/Firewall.
- Day 3–4 (T2): Implement database schema based on class diagram for User, Restaurant, Menu, and Review tables.
- Day 5 (T3): Set up Firebase for push notifications and configure Cloudflare WAF for security.

Backend Team:

- Day 1–3 (T4): Integrate Auth0 for user registration with password hashing.
- Day 4–5 (T5): Set up Flask structure and API stubs (Google Maps API, Restaurant API).

Frontend Team:

- Day 1–5 (T6): Create Figma designs, translate into HTML/CSS prototypes for Create Account, Login, Home screens.

Week 2

DevOps Team:

- Day 1–3 (T7): Configure Jira for task tracking, set up CI/CD pipeline to automate Flask updates to EC2.

Backend Team:

- Day 1–3 (T8): Implement Preference Engine logic (filters restaurants based on user preferences).
- Day 4–5 (T9): Implement getAddress() (Google Maps API integration) and Review APIs (submitReview, viewReviews).

Frontend Team:

- Day 1–3 (T10): Connect HTML forms to Flask APIs using JavaScript. Implement "Set Preferences" multi-screen flow.
- Day 4–5 (T11): Build "Your Restaurant Recommendations" page with a "Get Directions" button.

Week 3

All Teams:

- Day 1–2 (T12): White-box unit tests using JUnit and cross-browser compatibility tests using BrowserStack, logging and tracking defects in TestRail.
- Day 3 (T13): Load testing (1,000 concurrent user non-functional requirement) and SQL optimization (for restaurant search).
- Day 4 (T14): Finalize API documentation.
- Day 5 (T15): Final deployment to AWS Production environment.

Task Schedule and Dependencies:

Task	Description	Effort (person-days)	Duration (days)	Dependencies
T1	Provision AWS infrastructure	4	2	-
T2	Implement database schema	4	2	T1
T3	Set up Firebase and Cloudflare	2	1	T1
T4	Integrate Auth0 authentication	9	3	-
T5	Set up Flask and API stubs	6	2	T2
T6	Create UI designs and prototypes	10	5	-
T7	Configure Jira and CI/CD pipeline	6	3	T1
T8	Implement Preference Engine	9	3	T5
T9	Implement Maps and Review APIs	6	2	T8
T10	Connect frontend to backend APIs	9	3	T4, T5, T6
T11	Build recommendations page	6	2	T10
T12	Unit and compatibility testing	14	2	T7, T9, T11
T13	Load testing and optimization	7	1	T12
T14	Finalize API documentation	7	1	T13
T15	Final production deployment	7	1	T14

Total Effort: 106 person-days = 15.1 person-weeks ≈ 16 person-weeks

The critical path through the project is: T1 → T2 → T5 → T8 → T9 → T12 → T13 → T14 → T15

This path determines the minimum project duration of 3 weeks (15 working days).

4b. Cost, Effort and Pricing Estimation

We have chosen the Function Point (FP) method for estimating the effort required for the BiteScout project. This method is suitable for this project because it provides a technology-independent assessment of the project size and it can also be used to estimate the effort.

	Function Category	Count	Complexity	Count * Complexity
1	**Number of User Inputs**			
	User registration	1	6	6
	User login	1	3	3
	Set preferences	1	4	4
	Submit review	1	4	4
	Update profile	1	4	4
	Search/filter input	1	6	6
	Subtotal User Inputs	**6**		**27**
2	**Number of User Outputs**			
	Restaurant recommendation	1	7	7
	Restaurant details	1	5	5
	Reviews display	1	5	5
	Search results	1	7	7
	Error messages	1	4	4
	Subtotal User Outputs	**5**		**28**
3	**Number of User Queries**			
	View user profile	1	3	3
	Check availability	1	4	4
	View ratings	1	3	3
	Subtotal User Queries	**3**		**10**
4	**Number of Data Files and Relational Tables**			
	User database	1	15	15
	Restaurant database	1	15	15
	Preferences database	1	10	10
	Subtotal Data Files	**3**		**40**
5	**Number of External Interfaces**			
	Google Maps API	1	10	10
	Restaurant API	1	10	10
	Subtotal External Interfaces	**2**		**20**

Gross Function Point = User inputs + User outputs + User queries + Data files + External interfaces

GFP = 27 + 28 + 10 + 40 + 20 = 125

Question	Description	Rating (0-5)
PC1	Does the system require reliable backup and recovery?	5
PC2	Are data communications required?	5

PC3	Are there distributed processing functions?	4
PC4	Is performance critical?	4
PC5	Will the system run in an existing, heavily utilized operational environment?	4
PC6	Does the system require online data entry?	5
PC7	Does the online data entry require the input transaction to be built over multiple screens or operation?	3
PC8	Are the master files updated inline?	4
PC9	Are the inputs, outputs, files, or inquires complex?	3
PC10	Is the internal processing complex?	3
PC11	Is the codes designed to be reusable?	3
PC12	Are the conversion and installation included in the design?	4
PC13	Is the system designed for multiple installations in different organizations?	4
PC14	Is the application designed to facilitate change and ease of use by the user?	5

Total Processing Complexity (PC) = 56

Processing Complexity Adjustment = 0.65 + 0.01(56) = 1.21

Function Point = GFP * PCA = 125 * 1.21 = 151.25 = 152 FP

E = FP / Productivity = 152 / 10 = 15.2 = 16 person-weeks

D = E / team size = 16 / 7 = 2.28 = 3 weeks

4c. Estimated cost of Hardware Products

Category	**Item**	**Estimated Cost**	**Justification**
Development	Developer Workstations	$8,000	High-performance PCs for coding, testing, and development
	Mobile Testing Devices	$2,500	Android and iOS devices for application testing
Production (Cloud)	AWS EC2 Instances	$1,680	Hosts application servers with scalability
	AWS RDS Database	$780	Managed PostgreSQL database for production
Security & Backup	Firewall & VPN	$800	Network security and secure remote access
	Backup Storage (NAS)	$1,000	Local backups for disaster recovery

Total Hardware Cost = $14,760

4d. Estimated cost of Software products

Category	Item	Estimated Cost	Justification
Development Tools	IntelliJ IDEA	$1,000	Backend development IDE
	Postman	$600	API testing and collaboration
Design & Prototyping	Figma	$360	UI/UX design and prototyping
	Adobe Creative Cloud	$660	Graphics and asset creation
Third-Party APIs	Google Maps API, Restaurant API	$4,400	Location and directions services
	Firebase (Auth & Push)	$600	Authentication and push notifications
Security & Monitoring	Auth0	$2,880	Authentication and user security
	Cloudflare WAF	$1,200	Web application firewall and protection
Testing & QA	BrowserStack	$1,200	Cross-browser testing
	TestRail	$600	Test case management
Project Management	Jira	$420	Project tracking and agile management
	Slack	$480	Team communication and collaboration

Total Software Cost = $14,400

4e. Estimated cost of Personnel

To estimate the personnel cost for the BiteScout project, we used the Function Point method to calculate the total effort required to develop the software.

Total Function Points (FP): 152

Productivity: 10 FP per person-week

Estimated Effort: 152 / 10 ≈ 15.2 person-weeks → rounded to 16 person-weeks

Team Size: 7 developers

Project Duration: 3 weeks

Assuming an average labor rate of **$8,000 per person-month** (4 weeks), we can calculate the cost for the 4 developers

Total Personnel Cost=16 person-weeks×$2,000=$32,000

Additional Training Cost

After installation, a small training cost for developers and support staff is estimated at $1000, covering onboarding and operational procedures.

Total Personnel Cost = 32,000 + 1,000 = $33,000

Total Project Investment

Total Project Cost = Hardware + Software + Personnel

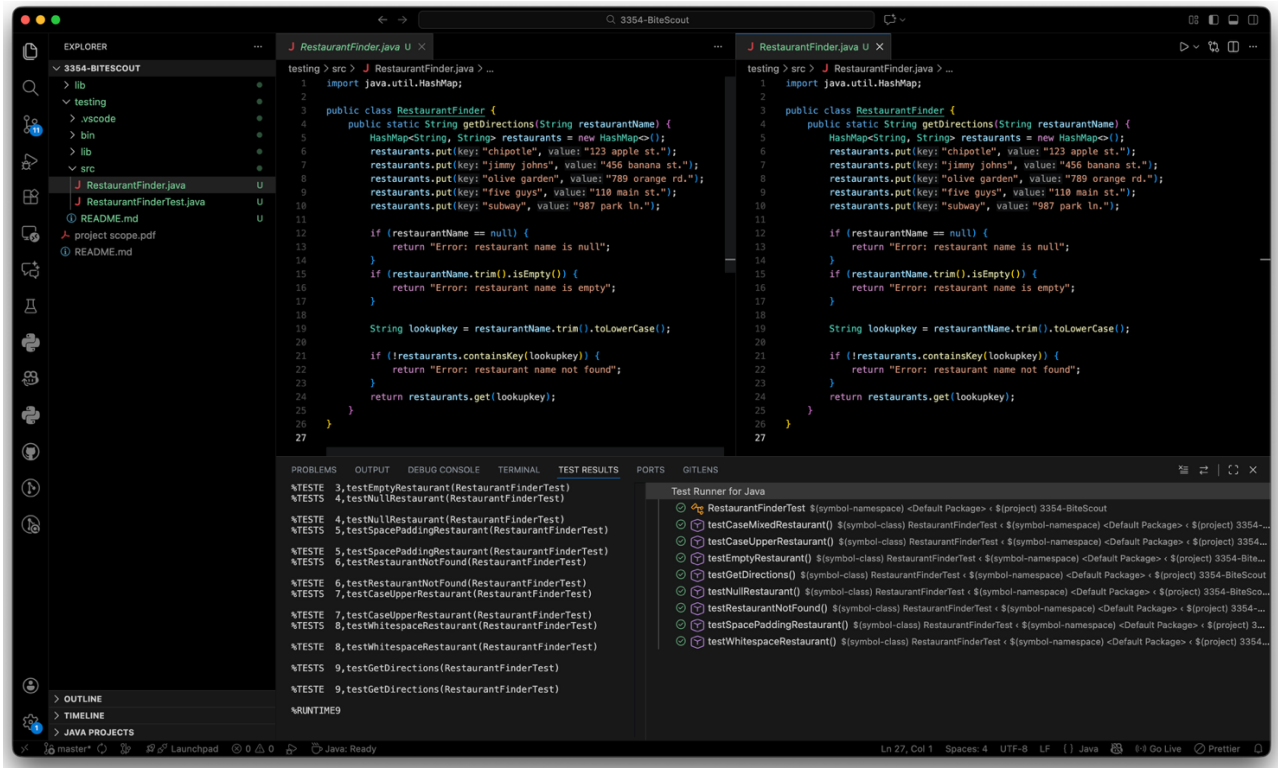
Total Project Cost = 14,760 + 14,400 + 33,000 = $62,160

Test Plan

a. Test plan

The testing will use Junit for a Java Unit as a form of Automated White-box Unit Testing. The software unit being tested is a function called getDirections(String restaurantName). This method is designed to receive a single String argument for a restaurant name. It makes a query to a sample hashmap with 5 restaurants. This method is case-insensitive, disregards whitespace for the input, and outputs a String for the address of the restaurant if found. If the restaurant name is not found, or the input is null or only whitespaces, the output should say what the error is that matches the input. By using a white-box strategy, we can write tests that will check all the logic. The method is considered successful when all unit tests pass the JUnit tests.

b. Evidence



Since all test cases passed (defined in the next section), the method is considered successful.

c. Test Cases and Results

Description	Input (restaurantName)	Expected Output
Success (Found)	“chipotle”	“123 apple st.”
Failure (not found)	“chick-fila”	“Error: restaurant name not found”
Edge case (null)	null	“Error: restaurant name is null”
Edge case (empty)	“”	“Error: restaurant name is empty”
Edge case (whitespace)	“ “	“Error: restaurant name is empty”
Case-insensitive (upper)	“JIMMY JOHNS”	“456 banana st.”
Case-insensitive (mixed)	“Olive Garden”	“789 orange rd.”
Whitespace padding	“ Five Guys “	“110 main st.”

Success is expected behavior and failure is for a restaurant name that is not found. Edge cases return Error: and why for null, empty, whitespace inputs. Case-insensitive tests for all caps, and a mixture of capitalization, to check it converts to lowercase and compare to the hashmap.

Whitespace padding ensures it trims leading and trailing whitespace in addition to converting to lowercase.

Comparison of Work with Similar Design

BiteScout is a restaurant recommendation system that helps people find nearby restaurants based on personal preferences. With this in mind, we can compare our project to other similar programs that are similar to an extent. There are two widely used platforms with similar functions interacting with food such as Yelp and Google Maps. These two platforms may have similar purposes as BiteScout, but the final goals of each one may differ in style and user experience.

Yelp first focuses on mostly user generated reviews and location search capabilities [1]. Its recommendation process is mostly just manual as the user will have to search and compare options themselves. BiteScout would differ with this motion, since it will be driven by preference recommendation engine where the system will automatically give you a list based on dietary needs, cravings and distance preferences, which make the experience more personal.

Google maps are also included in having search features, but it mainly focuses on navigation and general information on restaurants rather than having personalized recommendations [2]. When users are on google maps, it will show a list of sorted places by distance or rating which will require scrolling and comparing each one individually. It’s helpful for directions and basic details for each location, but it will not adjust towards your cravings or dietary needs. BiteScout on the other hand will help with suggestions that are more customized instead of just showing you the closest nearby restaurant.

Overall, while these two platforms are helpful with food in different ways, BiteScout is designed to combine preference-based filtering with a simple personal experience to match the user with what they want. The intention is to create a smooth, quick way to choose where to eat and make the process feel more tailored.

Conclusion

The app aims to make the food searching and delivery process easier for consumers, as it employs several features catered to dietary and personal restrictions. It provides directions, reviews, and account management for personalized use. It utilizes secure routes for accounts and connects these accounts to various parts of the restaurant system. If implemented, the app will rival Uber Eats and DoorDash as it uses features from both apps alongside its own custom properties. Through careful software development, this project can become reality.

References

[1] Yelp, “Find the best local restaurants,” Yelp.com. https://www.yelp.com (accessed: Nov. 15, 2025).

[2] Google Maps, “Nearby restaurant search and navigation,” Google Maps. https://maps.google.com (accessed: Nov. 15, 2025).