17-423/723 Course Project

Spring 2025

Overview

The goal of this project is to help you gain hands-on experience applying design principles and techniques from this class by designing, implementing, and iteratively improving a complex software system. In particular, you will work in teams to design, implement, and deploy a scheduling system for medical appointments, like those used by the public to schedule testing and vaccine appointments during a pandemic.

Although scheduling might seem like an easy task, it has multiple layers of complexity that makes it a challenging design problem. There are a number of different stakeholders (e.g., users/patients, pharmacies, hospitals, medical personnel, test/vaccine suppliers, policy makers) with competing requirements and constraints. The requirements may evolve as new types of testing/vaccine requirements arise or medical supplies fluctuate over time. During the pandemic, within the US, there were ambitious plans for building a unified, nation-wide or state-wide scheduling app, many of which ended up being far less than successful (ex1, ex2, ex3). These systems were found to be difficult to use, unreliable, and slow to keep up with rapid changes in vaccine supply/demands and policies. One of the main objectives of this project is to learn to design systems to be ready for these types of changes and to be able to satisfy different types of quality attributes that are critical for the success of the application.

Project Mechanics

Teamwork: You will work on this project in your assigned teams. As a team, you will use a shared GitHub repository and a virtual machine to coordinate your work. Please establish a way of communication and collaboration that works for your team -- for example, a Slack channel or a Trello board. Please agree on how you take clear notes at meetings that include agreed tasks and responsibilities. We do not expect that all team members contribute equally to each part of the project, but we expect that all team members make every effort to be a good team citizen (attend meetings, prepared and cooperative, respect for other team members, work on assigned and agreed tasks by agreed deadlines, reaching out to team members when delays are expected, etc).

Milestones: There will be in total six milestones throughout this project. In the first two milestones, you will design, test, and implement an initial prototype with a basic level of functionality involving appointment scheduling. Over the subsequent milestones, you will be asked to re-design the system to handle additional layers of complexity through the introduction of new features and quality attributes. In addition, some of the milestones will involve cross-team interactions: You will be asked to build a service that will be used by other teams; through this exercise, you will learn to develop services that are reusable and interoperable. Finally, one of the milestones will be devoted to design review and critique, where you will be

asked to review and provide construction criticism of other teams' designs.

Infrastructure: Each team will be provided with a virtual machine, where they will deploy their system as a web app. For evaluating and testing your system throughout the milestones, the course staff will attempt to interact with the deployed application by playing the role of various types of users (e.g., a patient, an administrator, a doctor or healthcare personnel). In a later milestone where you are asked to improve the robustness of the application, the course staff may evaluate your application through stress testing or other form of systematic testing.

Languages, tools, frameworks: Your team is free to choose any programming language (e.g., Python, Javascript, C#,...) or technology stack (e.g., Django, Flask, Express,...) for any part of this project. Each team will be assigned a virtual machine to deploy your system as a web application. You will have root access to the virtual machine and are free to install any software you deem suitable; but be responsible and careful as you also have the power to mis-configure the machine to make it inaccessible. You also may use external data and services (e.g. cloud services). For example, you can use free cloud credits that companies like Microsoft, Google, and AWS provide to students. Whenever you set up tools or services, pay some attention to configuring them with reasonable security measures; they may be vulnerable to indiscriminate attacks on the web, which could result in loss of data or Internet access for your virtual machine.

Documents and reports: Throughout most of the milestones, your team will submit reports that describe design alternatives that you considered, justification behind the final design decisions, and any lessons learned (what worked well and what did not). Please note that there's no one "correct" or "best" way to design a system; our evaluation of your reports will be based on the clarity and quality of your discussion of design alternatives, justification, and self-reflections.

Milestone 1: Domain Modeling and Initial System Design

Released: Wednesday, January 29, 2025
Due: Friday, Feb 7, 2025 11:59 pm (on Gradescope)

Learning Objectives

- Identify and specify the key entities in the problem domain and assumptions about them.
- Define a set of quality attribute scenarios that are relevant to the system.
- Apply design notations to specify and document a high-level design of the system.
- Consider alternatives for a set of design decisions and document justifications for final decisions.

Milestone Tasks

In this milestone, your team will design an initial version of the scheduling application.

Task 1: Domain Modeling

Begin by understanding the problem domain and identifying the key entities and assumptions (**ASM**) about their behaviors and properties. For this initial version of the application, focus on requirements (**REQ**) that are related to the basic functionality of scheduling, viewing, and modifying appointments. In particular, the user (i.e., a patient) should be able to use the application to view a list of available appointments, select an appointment at their preference, and be able to view an appointment(s) that they've previously made. In addition, the user should be able to modify an existing appointment (e.g., change it to a different, available time slot) or simply delete it.

Document the set of relevant domain entities and assumptions that you've identified using a context model.

Tips: In a typical design process, understanding the problem domain and identifying assumptions will involve talking to various stakeholders and domain experts. For this project, we've deliberately selected a domain that most people in the class are likely to be familiar with from their own personal experience using a similar type of application. There is no one "correct" context model that we are expecting to see for this task, and any model with a reasonable set of entities and assumptions will be acceptable. However, if you need further clarifications about the problem domain, please feel free to contact the course staff.

Task 2: Quality Attributes

Describe the most important quality attribute scenarios (between 5 to 8 would be a reasonable number) that you need to consider when designing the system and explain why they are

important. Furthermore, assign rough categories of priorities to each quality attribute scenario (e.g., “high priority”, “medium priority”, or “low priority”) to indicate which quality attributes are more important than others, and justify your prioritization.

Tips: Remember that quality attribute specifications should be measurable and associated with a scenario. Also note that for the same type of quality attribute (e.g., performance), you can specify multiple quality attribute scenarios (e.g., response time for one type of request and response time for another type of request)

Scope: The list of quality attributes and their specification might change throughout the project when you are learning more about the domain or new requirements. The quality attributes in this milestone should demonstrate a good understanding of the domain and be relevant for the basic functionality of appointment scheduling, but do not necessarily have to be “complete” or “final”.

Task 3. Component and Data Model Design

Once the domain model has been developed, develop a high-level design of the system, including (1) the set of major components in your system and interfaces among those components and with the domain entities and (2) a data model that captures different types of information that your system will store.

As you discuss possible design solutions within your team, consider the following questions, including:

- What are the major components in the system, and what are their responsibilities?
- How do the components communicate with each other? (e.g., HTTP, method calls)
- What does the interface for each component look like? What information does each interface function take as input, and what does it return?
- Where is each component deployed? (e.g., on the user’s web browser, a server)
- What information about the domain entities does the system need to store?
- What are appropriate multiplicity constraints over those data types?
- What programming languages and web frameworks will be used?
- What is the expected sequence of interactions between the components and domain entities in typical scenarios?
- How is the system designed to achieve the quality attributes identified in Task 2?

Document your design using (1) a component diagram and (2) a data model. In addition, for **two** of the design questions listed above, (i) describe alternatives that you considered, (ii) your final decision, and (iii) justification for your decision.

Scope. Since this is your team’s first design assignment, you will not be evaluated based on the quality of your design (i.e., how well it achieves different types of quality attributes that we will discuss throughout this class, such as modularity, reusability, interoperability, scalability, robustness). For this milestone, a design that achieves the basic functional requirements of

appointment scheduling will be sufficient; you will be asked to iterate on and improve this design in the future milestones.

Deliverables

Submit a report as a single PDF file to Gradescope that covers the following items in clearly labeled sections (ideally, each section should start on a new page). **Please correctly map the pages in the PDF to the corresponding sections.**

1. **Domain model (2 pg max)**: A context model that includes the domain entities, along with a list of system requirements (REQ) and assumptions about the behaviors or properties of the entities (ASM).
2. **Quality attribute specifications (2 pg max)**: A list of quality attribute scenarios, their categories of prioritization, and justification for their prioritization.
3. **Component design (2 pg max):** (i) A component diagram that describes the set of major components in the system, (ii) a description of the interfaces among those components, and (iii) a description of the responsibilities of each component.
4. **Data design (1 pg max):** A data model that describes different types of information stored in the system.
5. **Design discussion (1 pg max)**: A discussion of two design questions, alternatives considered, and justification for the final decisions.

Grading

This assignment is out of **120** points. For full points, we expect:

- A valid context model that includes a set of relevant domain entities and interactions between them (**10 pts**), a list of requirements (**10 pts**), and a list of assumptions about the entities (**10 pts**).
- At least 5 relevant quality attribute scenarios, specified in a unambiguous and measurable way, with justified priorities (**20 pts**).
- A valid component model that contains a set of components and connections between them (**10 pts**), a description of the interfaces between the components **(10 pts**), and a description of the responsibilities of those components (**10 pts**).
- A valid data model that (1) contains information that is necessary for fulfilling system requirements (**10 pts**) and (2) includes relations between data types and valid multiplicity constraints over those relations (**10 pts**).
- A discussion of two design decisions. For each decision, (1) a description of at least two alternatives considered (**5 + 5 pts**) and (2) a justification for the final decision (**5 + 5 pts**).
- Bonus social points **(5 pts)**: See below.

Bonus social points (5 pts): Participate in an in-person social activity with your team that is not related to any coursework. This could just be an informal happy hour, playing a board game or computer game together, doing a puzzle or trivia quiz, watching a movie, or whatever you like, as long as it is not course related. After you do, post a selfie to the public Slack channel #social

(including a photo of the team members) and tag all members who participated. To receive the bonus points, please post the Slack message **within 3 days** after the milestone deadline.

Milestone 2: First Prototype Development

Released: Monday, Feb 10, 2025
Due: Monday, Feb 28, 2025 11:59 pm (on Gradescope)

Learning Objectives

- Develop an implementation that conforms to a design specification
- Refine and adjust earlier design decisions
- Design the system for testability and changeability

Milestone Tasks

In this milestone, your team will develop an initial version of the scheduling application based on the design that you have devised in Milestone 1.

Task 0: Team Contract

Meet with your team and discuss how you plan to divide the tasks among the team members. Do not wait until the last few days of the M2 deadline to start working on the tasks. Set intermediate milestones (e.g., every 4~5 days) and check in with the other team members frequently; this will allow you to identify and debug technical problems and issues early on. Reach out to the course staff if you face team challenges that are difficult to address internally.

A part of your deliverables will be a team contract that states the responsibility division and the intermediate milestones.

Task 1: Prototype Implementation

Implement a web-based application that supports the following basic set of features: Scheduling, viewing, and modifying appointments. The user (i.e., a patient) should be able to use the application to view a list of available appointments, select an appointment at their preference, and be able to view an appointment(s) that they've previously made. In addition, the user should be able to come back to your site at a later time and modify an existing appointment (e.g., change it to a different, available time slot) or simply delete it. Your system must avoid scheduling conflicting appointments (i.e., each appointment slot should be assigned to at most one user).

Deploy your application on the VM that we have provided you with. Your application should provide a basic user interface that allows the user to perform the above tasks. Your system must support persistence of data (e.g., appointments made by users should persist in a database). The course staff will be accessing the URL for the frontend of your application and testing various scenarios that involve the scheduling features listed above.

Your team will use a private Github repository to develop and version source code artifacts for your application (instructions to follow separately). The course staff will have access to this repository and read your code as part of the evaluation.

Scope: Your app does not need to provide user authentication or other security mechanisms, although the user should be able to retrieve their appointments using an identifier such as their name and/or email. In addition, your app does not need to handle issues that may arise due to concurrency (i.e., multiple users simultaneously trying to select the same appointment slot, causing race conditions).

Testing: As part of the codebase that you will submit for this milestone, you must provide a set of test cases that you develop and use to test the key functionality of the application. We do not impose a strict requirement on the number of test cases, but your test suite should be diverse enough to cover important scenarios that may arise in production. We also encourage you to test scenarios that involve unusual corner cases.

Tips: Focus on building a first *minimal viable product (MVP)*, not a perfectly polished application. For example, you do not need to spend a lot of effort on the visual aesthetics or usability of the frontend; a bare minimum UI that supports the required functionality will be sufficient. Similarly, performance is not a quality attribute that we will be evaluating in this milestone (although your application should still be functional; i.e., it should not hang forever!).

Sample code (optional): To help you get started, we have provided barebone code (with a React frontend, a Flask backend, and a MongoDB database), available at the following link:

https://github.com/cmu-swdesign/Team_Project_Boilerplate_Code

You are free to fork/clone the repository and use the code as part of your project. Alternatively, you are also welcome to build your own application from scratch, using any programming language, web framework, or libraries of your choice.

Task 2: Design for Changeability

Design your system to be ready for change. First, consider different types of changes that may occur in your application, due to changes in requirements, domain assumptions, new features, and/or qualitiy improvement. For each change, consider the impact of the change on the rest system and identify other components that may also need to be modified.

Apply design principles (i.e., information hiding, single responsibility, interface segregation, and dependency inversion) and techniques (data abstraction, interface abstraction, and encapsulation), discussed in class, to minimize the impact of likely changes in your system. As a part of the deliverables, you will be asked to describe the design decisions that you've made to address **at least two** of the possible changes.

Task 3: Design for Testability

Design your system to be testable. Identify **at least one controllability challenge** and **at least one observability challenge** for testing your application. For each of these challenges, use a concrete use case scenario that you'd like to test, to describe how the challenge makes testing difficult or require more effort.

Apply design principles discussed in class to address these two challenges. As a part of the deliverables, you will be asked to (1) describe the design decisions that you've made to improve testability and (2) include a reference to the test cases for the above scenarios in your codebase.

Task 4: Design Reflection

As you develop the application, you will gain new insights and knowledge about the domain or solution space that were not obvious during the design stage. As a result, you will likely refine abstract design decisions into more concrete ones, make additional decisions, or change the decisions that you made during M1. For this task, prepare a report that reflects on how the implementation process has influenced the design decisions that you made earlier in M1. In particular, the report should:

1. Discuss **at least two** design decisions that you have changed or additionally made since your initial M1 design. For each, provide a justification for the change or the need for an additional design decision.
2. Include an updated component diagram for your implementation. If there are any changes from the component diagram in M1, describe the design decisions that resulted in those changes.

Deliverables

Submit a report as a single PDF file to Gradescope that covers the following items in clearly labeled sections (ideally, each section should start on a new page). **Please correctly map the pages in the PDF to the corresponding sections.**

1. **Deployment (1 paragraph)**: Include the URL for accessing the main frontend of your application.
2. **Test suite (1 paragraph)**: Include the URL to the part of your team Github repository that contains the set of test cases used to validate the functionality of the system.
3. **Changeability discussion (2 pg. max)**: A document discussing possible changes to your system and the design decisions that were made to improve the changeability of the system (Task 2).
4. **Testability discussion (2 pg. max**): A document discussing (1) a controllability challenge and an observability challenge, and (2) the design decisions that were made

to improve the testability of the system, along with an URL reference to the test cases in the Github codebase (Task 3).

5. **Design reflection** (**3 pg. max**): A document reflecting on how the implementation process has influenced earlier design decisions (Task 4).
6. **Team contract (1 pg max)**: A documentation of (i) the division of development tasks among team members and (ii) intermediate milestones, each with a target date and goals achieved.

Grading

This assignment is out of **120** points. For full points, we expect:

- **(40 pt)** A functional web application that provides the features for (1) viewing a list of available appointments, (2) making an appointment, and (3) modifying or deleting an appointment.
- **(10 pt)** A set of test cases that cover the above functionality
- **(20 pt)** A discussion of **at least two possible changes** in your system (**10 pt)** and the design decisions made for minimizing the impact of those changes (**10 pt**)
- **(20 pt)** A discussion of at least one controllability challenge (**5 pt**) and one observability challenge (**5 pt**) in testing your application, and the design decisions made to address these challenges (**10 pt**). The solution also includes an URL to the test cases that demonstrate the effect of testability improvement.
- **(20 pt)** A design reflection with (i) a discussion of **at least two** new or changed design decisions along with their justifications (**10 pt**), (ii) an updated component diagram with the design decisions that resulted in changes (if any) **(10 pt)**.
- **(10 pt)** A team contract that describes (i) the division of tasks among team members and (ii) a set of intermediate milestones, their target date, and expected goals.
- **(5 pt)** Bonus social points (same as M1).

Milestone 3: Design for Interoperability

Released: Wednesday, Mar 5, 2025
Due: 11:59 pm Friday, Mar 21, 2025

Learning Objectives

- Extend an existing system to support additional features.
- Design interfaces for interoperability with other systems.
- Design a large-scale software system in multi-team settings.

Milestone Tasks

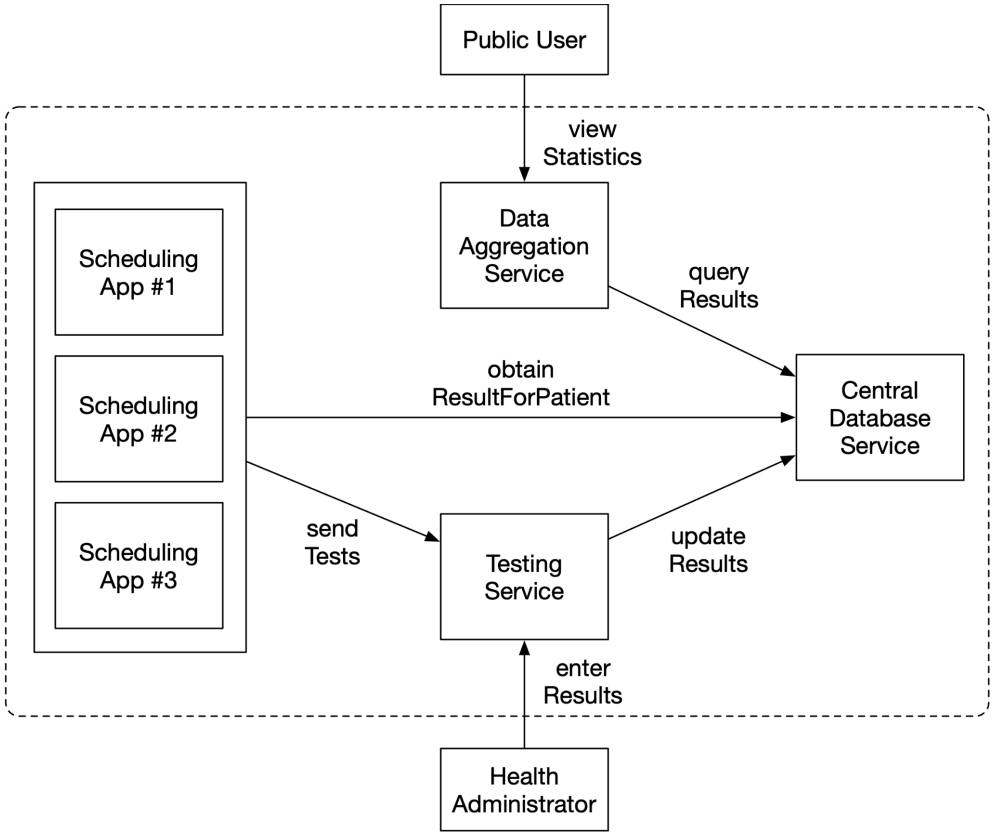
In this milestone, you will build on the prototype that you have developed in M2 to support additional features. In addition, you will start working with other teams in the class to design interfaces for a new service that will eventually be implemented and used by everyone else.

Task 1: Prototype Extension

You will extend the existing implementation from M2 to accommodate the following feature:

Feature: Test result reporting: Once a testing appointment has been fulfilled, your system should provide the user (i.e., a patient) with an ability to view the result of their test (Positive, Negative, or Pending). In reality, the test samples collected from the patients would be sent to a medical laboratory for processing and the results would be sent back after a few days. For the purpose of this milestone, you should build **test double components** that stand in for an external Testing Service and Central Database Service (which will eventually be built by the teams, as discussed below). The test double components should **randomly** produce, for each patient who has completed a testing appointment, an output that represents the hypothetical test result. Note that once a test result has been generated for a particular patient and an appointment, the test result should remain constant (i.e., if Alice tests Positive for a test done on 03/05/2025, then it should permanently be recorded in your system as Positive). Adding this feature is likely to involve storing additional data about patients, including test results and their corresponding dates. In your system, patients should be able to view test results for all their previous tests that have been reported back already.

Staff testing: Like in M2, the course staff will directly interact with your application to test out the new feature. For the purpose of staff testing only, please configure the test double components so that test results are generated immediately when the user makes a testing appointment.



Task 2: API Design for Interoperability

This part of the milestone will involve collaboration across multiple teams.

Starting this milestone, each team will be assigned the responsibility of designing (and eventually implementing) a service that will be deployed and shared by all teams. Each service will provide functionality to support a particular set of stakeholders and tasks. There will be three shared services (one for each team), as illustrated in the diagram above:

- **Testing Service**: Provides the service for scheduling apps to request test results. Each scheduling app sends information about patients and their testing appointments, which are first stored into the Central Database. Once a health administrator enters the test results, the Testing Service updates the corresponding entries with the test results.
- **Central Database Service:** Provides storage and retrieval of information about patients across multiple scheduling apps (including their up-to-date testing history). Other apps and services will be able to access information about patients through an API that is provided by this service.
- **Data Aggregation Service**: Allows the members of the public to view various statistics related to an on-going pandemic, such as the number of known positive cases and the overall trend over a user-specified period of time. To compute the statistics, this service relies on the information provided by the Central Database Service.

API design activity (Mar 12, 2025): In this milestone, your team will design (but not yet implement) an API for the assigned service. Since these services will be accessed by multiple applications, **interoperability** is a key quality attribute to be achieved. To facilitate the process of designing APIs together, we will allocate a lecture slot on **Wednesday, Mar 12, 2024** as a design session. **Every member of your team is expected to be present at this session**.

API documentation: Your team will document the outcome of this activity as an API documentation that describes (1) the list of interface functions, (2) input and output parameters that are associated with each function, and (3) a contract that describes the pre- and post-condition of each function. In addition, your documentation should include an ontology that describes the meaning of data elements that are passed as parameters, as well as the relationships between those elements.

Tips for Cross-team Collaboration

To ensure seamless integration of your service API with the services and scheduling apps developed by others, **cross-team communication will be crucial.** You can use Slack for cross-team communication and maintain a shared Google Docs document outlining all APIs and assumptions. We also recommend that each team picks one representative who is responsible for being an "**interface person**" of the team and responsible for answering questions from other teams. Tell the other teams who your interface person is so that they know who to contact. In your team contract, allocate a member to be responsible for cross-team communication tasks.

Remember that other teams might use your API and might need to write test double components for your API in this milestone. Please be courteous to them and **finalize your API design & documentation early** to give them time to design their system around your API.

Task 3: Design Reflections

In the process of extending your prototype with the new feature, it is likely that the design of your application will undergo some changes. Depending on how you designed your system in M1 & M2, these changes may be straightforward to make (e.g., by adding new components with no or minor changes to the existing ones) or disruptive (i.e., involves significantly changing the existing components). For this task, prepare a report that reflects on the design of your system has evolved to accommodate new features and possible future changes. The report should:

1. Describe the changes to the design of your system from M2. Include an updated component diagram that highlights those changes.
2. Describe design decisions that you have made for the API of the shared service in Task 2 and justify how those decisions support service interoperability.

Deliverables

Submit a report as a single PDF file to Gradescope that covers the following items in clearly labeled sections (ideally, each section should start on a new page). **Please correctly map the pages in the PDF to the corresponding sections.**

1. **Deployment (1 paragraph)**: Include the URL for accessing the main frontend of your application.
2. **Testing report (1 pg max)**: A description of how you used test doubles to test the new feature, including links to the test artifacts (e.g., parts of the source code that contains the test doubles and the test cases that use these doubles to test the new feature).
3. **Shared service API documentation (2 pg max**): A documentation of the API for the shared service that has been assigned to your team, based on Task 2.
4. **Design reflection report** (**2 pg max**): A document reflecting on the (re)design decisions that you have made, as described above in Task 3.
5. **Team contract (1 pg max)**: A documentation of (i) the division of development tasks among team members (including the interface person) and (ii) intermediate milestones, each with a target date and goals achieved.

Grading

This assignment is out of **110** points. For full points, we expect:

- **(30 pt)** A functional web application that extends the prototype from M2 with the new feature.
- (**20 pt**) A documentation of your approach to implementing the test doubles for the new feature & corresponding automated tests.
- **(30 pt)** A documentation of an interface for the shared service that has been assigned to your team, including (1) a contract for each function describing its expected behavior and (**20 pt**) and (2) an ontology for data elements that are passed as input and output parameters (**10 pt**).
- **(20 pt)** A design reflection report with (i) a discussion of how the system design from M2 has been changed to support the new feature, with an updated component diagram (**10 pt**) and (ii) a discussion of design decisions for the shared service interface and justifications for those decisions with respect to interoperability (**10 pt**).
- **(10 pt)** A team contract that describes (i) the division of tasks among team members and (ii) a set of intermediate milestones, their target date, and expected goals.
- **(5 pt)** Bonus social points (same as the previous milestones).

Milestone 4: Service Development & Integration

Released: Saturday, Mar 22, 2025

Learning Objectives

- Design and implement a large-scale software system in multi-team settings.
- Design, implement and test an interoperable service to be used by other teams.
- Integrate and test services developed by other teams.

Milestone Tasks

In this milestone, your team will develop a working prototype of the service assigned to your team in Milestone 3, integrate the services developed by other teams, and help other teams integrate your service. This milestone has **two parts with separate deadlines,** where (A) the first 1.5 weeks will be allocated for implementing the shared service that has been assigned to your team and (B) the second week will involve integrating and testing all of the services. **Only part B includes a report submission.**

Part A: Service Development

Due: 11:59 pm Wednesday, **Apr 2**, 2025

Task 1: Service Implementation

Implement the web service assigned to your team. Please ensure that your service conforms to the API you shared and try to not change your API, as the other teams depend on it. In the exceptional case where it is absolutely necessary to modify the API, please notify the course staff and the other teams through Slack. When implementing your service, please carefully consider applying the design principles taught in the lectures, including Design for Change and and Design for Interoperability, to ensure that the implementation is modular, understandable, and compatible with other services.

You may use any available resources for your implementation, including code generation, generative AI, frameworks, libraries, and code snippets from the web. Please document when you use generative AI tools and code from other sources.

Task 2: Service Testing

Develop tests to validate the functionality of your service.

Recall the discussion of **contract testing** from Recitation 6 (March 14). First, develop a set of tests to validate the functionality of your own service. If your service (i.e., **consumer**) depends on another service(s) (i.e., **provider**), create test double components for those services.

Document the expected input-output behaviors of each test double as a **contract**. Publish those contracts to the respective section for the provider service on the shared interface document.

When another team publishes contracts for your service, use those contracts as a basis to create test cases for your own service. Please ensure that your service handles errors gracefully and does not crash on erroneous inputs, as this will make integration easier!

Note: If your team is developing the Central Database Service, you will not need to create any contracts, as your service does not depend on another service. If your team is developing the Data Aggregation Service, you may not receive any contracts, as no other team depends on your service.

Task 3: Service Deployment

After you have implemented and tested your service, deploy the service as an HTTP endpoint on the VM that has been assigned to your team. You will be asked to share the URL and port number for your service with the other teams, so that they can start accessing the service from their own services. To avoid delays in the following integration process, your service must be up and running by the above stated deadline for Part A.

Part B: Service Integration
Due: 11:59 pm Wednesday, **Apr 9**, 2025

Task 4: Service Integration

Replace the test doubles that you used in Part A with the actual services that have been deployed by the other teams. In theory, this step should be as simple as replacing the invocation of a service from a test double to the deployment endpoint for the service. However, there are likely to be unexpected incompatibility issues that arise (possibly due to the changes in the APIs or implicit assumptions about how the other services work). Keep track of these issues as they arise and coordinate with the teams to resolve them. At the end of this milestone, all of the appointment scheduling apps and shared services must be fully functional.

Task 5: End-to-end Integration Testing

The goal of this task is to develop and run additional tests to ensure that your team's implementation (i.e., the scheduling app and the assigned shared service) integrates correctly with the other services. Identify a set of use case scenarios that involve your scheduling app, your shared service, and other teams' services as different components in the overall system. Document these scenarios and develop tests to ensure that the integration works correctly. Sequence diagrams are a good way to document these scenarios, although you do not need to create them for every integration test. You may automate these tests if possible, but a precise sequence of instructions to follow how to run the tests manually is sufficient as well.

Important notes: Since some of the services are shared by multiple teams, please **take extra caution when testing stateful operations**; i.e., those that involve changes to the state of the system (e.g., modify an existing appointment that was created by another team's scheduling app). Doing so may interfere with and cause another team's tests to fail. If you are testing a shared service that involves state changes, make sure that those changes involve test data that was created by your team.

In addition, please avoid overloading the shared services with too many requests at once, as doing so may introduce delays and interfere with another team's usage of the service.

Finally, during the course of the integration process and the future milestones, your team's service may behave unexpectedly or even possibly fail (e.g., when given an unexpected input). Please try to monitor your service from time to time and ensure that it is available throughout the remainder of this project. If, at some point, your service experiences a failure, please try to re-deploy it and make it available to the other teams as soon as possible.

Task 6: Design Reflection

Reflect on your experience implementing, (possibly) re-designing, and integrating your service with the scheduling app and the other teams' services. In particular, discuss the following points:

1. **Design changes**: How did the design of your service or scheduling app change as a result of multi-team integration? Why were those changes necessary? Based on your experience in M4, how would you have designed your service differently in M3?
2. **Communication**: What difficulties did you face in communicating and discussing design decisions with the other teams?
3. **Integration testing:** What challenges did you face in testing the integration of the entire system? How did you overcome these challenges?

Deliverables

Submit a report as a single PDF file to Gradescope that covers the following items in clearly labeled sections (ideally, each section should start on a new page). **Please correctly map the pages in the PDF to the corresponding sections.**

1. **Deployment (1 paragraph)**: Include (1) the URL for accessing the main frontend of your application and (2) the URL for the shared service that your team has implemented.
2. **Service testing report (1 pg max)**: A description of how you tested your own service using test doubles and/or contracts (if applicable to your service). Include links to the test artifacts (e.g., parts of the source code that contains your test cases).
3. **End-to-end Integration testing report (2 pg max):** A description of at least **three use case scenarios** that you developed for testing the integration of your scheduling app, your service, and the other services. Include links to the test artifacts.

4. **Design reflection report** (**2 pg max**): A document reflecting on the process of implementing and integrating your service with the other parts of the system, as described above in Task 6.
5. **Team contract (1 pg max)**: A documentation of (i) the division of development tasks among team members (including the interface person) and (ii) intermediate milestones, each with a target date and goals achieved.

Grading

This assignment is out of **110** points. For full points, we expect:

- **(40 pt)** A functional web service that implements the functionality assigned to your team and a functional appointment scheduling system that integrates with the other services.
- **(10 pt**) A service testing report outlining how you tested your own service.
- **(20 pt)** An integration testing report outlining how you tested the overall system during the integration phase.
- **(30 pt)** A design reflection report.
- **(10 pt)** A team contract that describes (i) the division of tasks among team members and (ii) a set of intermediate milestones, their target date, and expected goals.
- **(5 pt)** Bonus social points (same as the previous milestones).

Milestone 5: Robustness Testing

Released: Wednesday, Apr 9, 2025
Due: **11:59 pm Friday, Apr 18, 2025**

Learning Objectives

- Test and evaluate a software component/service for robustness using an API specification.
- Redesign the component/service to improve its robustness.

Recommended Resources

- https://www.geeksforgeeks.org/robustness-testing/
- https://www.geeksforgeeks.org/security-testing/
- https://www.geeksforgeeks.org/software-testing-scalability-testing/
- Fuzz Testing:
  - https://www.freecodecamp.org/news/web-security-fuzz-web-applications-using-ffuf/
  - https://www.fuzzingbook.org/html/WebFuzzer.html
  - https://medium.com/@cuncis/fuzzing-made-easy-how-to-use-wfuzz-for-efficient-web-application-testing-d843e5b089bf
  - https://github.com/marmelab/gremlins.js
- Bug Report Writing: https://www.softwaretestinghelp.com/how-to-write-good-bug-report/

Milestone Tasks

In this milestone, your team will evaluate the robustness of the service developed by another team by trying to “break” it. Your team will be assigned the service to test in a separate message from the course staff.

Task 1: Designing Robustness Tests

Given the API specification of the service under test (SUT), identify faults that could break the SUT. These could be malformed requests, unexpected corner cases, malicious inputs, or creating an overload situation. Your goal is to identify multiple ways in which the SUT might break and describe potential ways for you to inject these faults into the system using the API provided to you.

Be **creative:** Think of many different ways to break the SUT. You can consider possible robustness issues, security attacks, or overload situations that expose scalability issues.

Try to think of **potential implementation shortcuts** the development team might have taken: Given the time constraints, the development team most likely identified certain inputs as not

likely to happen and did not consider them in their design. If you would have been on their team, which inputs would you have ignored? These are potential candidates for you to test!

Also, to increase your likelihood of finding issues, we recommend you to think of ways to **increase the coverage** of your robustness tests (i.e, efficiently covering many possible inputs - see the ressources on fuzz testing above). How can you generate a large range of inputs that are likely to break the SUT? We do not have requirements for minimum coverage. This note is just to help you find issues more efficiently.

Task 2: Executing Robustness Tests

Execute the tests that you described above. These can be automated, semi-automated, or manual. If possible, consider the use of a tool to automate the test execution. You may, for example, use a stress/robustness testing tool such as locust.io (quick start guide) or gremlins for this purpose.

Please be courteous to the team that has built the SUT. Before running your tests, please let them know ahead by sending them a message. If you actually manage to crash the SUT, request a restart of the service and try to avoid running the same test again to avoid forcing restarts more often than necessary.

You have successfully found an issue if:

- the SUT crashes
- the SUT returns factually incorrect results or results that are different from the API specification
- the response takes much longer than expected or the SUT does not respond at all
- the SUT returns an error message even though the inputs conformed to the API specification
- you gained unauthorized access to data within the service, commands within the service, or the VM inside which the service runs (please do not attempt to extract real personal information, such SSH keys, credentials, or personal files, from the VMs of other students)

If possible, please try to identify **at least two** issues in the SUT. The issues can be of the same general type (i.e., service crashes on incorrect input), but should be meaningfully different from each other (e.g., “failing on -1, -2, -3” and “failing on negative inputs” together are considered one issue instead of two, but “failing on negative inputs” is considered a different issue from “failing for extremely large inputs”). You will receive **bonus points** for discovering issues from more than **two** categories in the above list.

Task 3: Reporting Results

Send a report containing your tests and results to the team that built the SUT by **Tuesday, April 15**. Provide them with steps to reproduce the issues that you identified (e.g., sequence of API

requests or user inputs). If you used scripts to simulate requests, please also provide them with these scripts to make it easier for them to reproduce the results.

In the unlikely scenario that you were not able to find any issues with the SUT, send a report containing your tests only. In your report, provide an argument that your tests cover a diverse set of scenarios and inputs.

Task 4: Improving Robustness

For each of the two issues that the other team found in your service, describe a possible design modification for improving the robustness of your system. Provide a justification for how this modification improves robustness; this may include (although not necessary) a fault tree that shows how the modification increases the size of the minimum cut set for a particular failure. If applicable, refer to a design pattern or principle discussed in one of the lectures on robustness, scalability, or security.

Given time constraints, you are **NOT** required to actually implement the robustness improvements that you come up with. However, you will receive **bonus points** for any improvement that you implement and demonstrate by re-running the relevant test and showing that it no longer causes a failure.

Deliverables

Submit a report as a single PDF file to Gradescope that covers the following items in clearly labeled sections (ideally, each section should start on a new page). **Please correctly map the pages in the PDF to the corresponding sections.**

Test Report: Describe your test design, results that you found, and steps to reproduce the issues. Your test report should describe how you selected the inputs that you test, how you execute the tests, and how you assert that an issue is present. For creating bug reports and reproduction steps, please see these guidelines for writing effective bug reports: https://www.softwaretestinghelp.com/how-to-write-good-bug-report/. Please send this report to the development of the SUT by **Tuesday, April 15**, and submit a copy on Gradescope by the **Friday**, **April 18** deadline.

Robustness Improvement Report (**3 pg max**): Describe modifications to the design of your service that would improve its robustness against the issues found by the other team. For each of the two, justify how the modification improves robustness. Optionally, if the modification is implemented, provide a brief description of the implementation and a link to the part of your source code that contains the modification.

Grading

This assignment is out of **70** points. For full points, we expect:

- **(40 pt)** A testing report that clearly describes your approach to identify potential robustness issues with the service of another team, presents your findings (including at least two found issues), and clearly describes steps to reproduce the issue. **5 bonus points** for issues from more than two categories found.
- **(20 pt)** A report describing possible robustness improvements to your system and a discussion of alternative solutions (Task 4). **Bonus points** for actually implementing and demonstrating an improvement (5 bonus points per each improvement, up to 10 points in total).
- **(10 pt)** A team contract that describes (i) the division of tasks among team members and (ii) a set of intermediate milestones, their target date, and expected goals.
- **(5 pt)** Bonus social points (same as the previous milestones).

Final Presentations

Released: Wednesday, Apr 16, 2025
Due: 9:30 am Wednesday, Apr 23, 2025 (**in-class**)

Learning Objectives

- Reflect on your semester-long design process and generalize them towards actionable lessons learned.
- Describe concepts of cross-team collaboration, multi-team development, design evolution, and testing.

Recommended Resources

- Lecture slides throughout the semester
- Recommended reading from the lectures (see the lecture slides and course website)
- Example talk: https://canvas.cmu.edu/files/11086390/download?download_frd=1

Tasks

For continued growth as a software designer, it is important to continuously reflect on how effective the techniques & principles that you've used are, and how your approach to designing software can be improved. The final presentation is dedicated to practicing this process of retrospective reflection and identification of areas of improvement. To accomplish this, the major part of your presentation is to share what went wrong, what went right, and what you can learn from this experience for future projects.

Please prepare a presentation (**15 minute maximum**) that focuses on sharing your lessons learned from the project throughout the semester. There will be a **3~4** min Q&A session after your talk. For each lesson learned, please include and clearly separate your **observations** from your **interpretations** towards more generalizable guidelines that you or other students can apply for future projects.

The **observation** aspect should include a concrete description of the situation (i.e., available design options and the context that is necessary to understand these options), the decision you made (e.g., selection of process, method, or design decisions), and the consequences you observed. The **interpretation** aspect should include a reflection on whether you think the decision you made was appropriate and your lesson learned from this experience, which should be actionable and based on your observations. You can see an example of a talk that follows this structure at the following link:
https://canvas.cmu.edu/files/11086390/download?download_frd=1

We encourage **every team member** to participate in the presentation. You do not have to evenly split your time, and it is up to you if you want to divide the talk into sections or present each section as a "dialogue" between multiple speakers.

Please share your experience on each of the following four topics. Sample questions that you could discuss for each topic are also listed below (you do **NOT** need to answer every question listed). You are also free to deviate from these questions if there are more interesting things that you'd like to discuss for each topic.

Topic 1: Your Lessons Learned from Cross-Team Communication

Observations: How did you structure your communication? How did teamwork across multiple teams go? What challenges did you face when communicating with other teams?

Interpretations: What have you learned from communicating with other teams? What advice would you give to next year's students? How would you improve cross-team communication when working on your next cross-team project?

Topic 2: Your Lessons Learned from Multi-Team Service Development

Observations: Which design principles or design techniques did you apply to structure the high-level architecture of the system and to ensure compatibility between services? How well did integration go? Were there any issues caused by implicit assumptions, interface incompatibility, or other integration issues?

Interpretations: What have you learned from developing a service in a multi-service project? What advice would you give next year's students? Which design techniques would you use in your next large-scale project and how can you improve the architecture consistency in multi-team projects?

Topic 3: Your Lessons Learned from Design Evolution

Observations: Which design decisions have changed throughout the project? How well did the consequences of your design decisions align with your expectations before implementation?

Interpretations: If you could go back in time, which design decisions that you made would you change? What have you learned from designing a software system over multiple iterations under changing requirements? What advice would you give to next year's students? How would you improve your design process to be more effective for evolving architectures (e.g., which design techniques would you use in your next large-scale project)?

Topic 4: Your Lessons Learned from Evaluating Large-Scale Software Systems

Observations: Which design evaluation & testing techniques have you used throughout the project? Did your own tests find any bugs that surprised you? Did robustness testing by another team discover issues in your service that you didn't anticipate? Did the process of writing test cases change your perception of software design?

Interpretations: If you could go back in time, would you spend more or less time on design evaluation and/or testing? What have you learned from testing functionality and from testing the integration of multiple services? What advice would you give to next year's students? How would you improve your design evaluation and testing techniques for your next large-scale project?

Tips

You are also welcome to include other content in your presentation and you are free to choose a format and structure for your talk. However, since you only have 15 minutes to talk about a semester-long project, we recommend that you focus on the important insights & lessons learned, instead of trying to cover . Make sure that the observations you include are relevant to your interpretations, and the interpretations clearly follow from the described observations.

As you can see from the grading rubric, we do not evaluate slide design or verbal delivery, so please focus your time on the quality of the content rather than visuals.

Deliverables

Submit your slides as a PDF file to Gradescope and verbally deliver your presentation in class. We recommend that you use your own laptop to present your talk by connecting to the room AV system.

Grading

This assignment is out of **100** points. For full points, we expect:

- **(50 pt)** Insights clearly demonstrate a deep understanding of software design by referring to appropriate concepts, design principles, and challenges of designing software systems.
- **(20 pt)** All four discussion topics are included in the presentation.
- **(10 pt)** Interpretations are separated from observations.
- **(10 pt)** Interpretations are clearly based on observations.
- **(10 pt)** Presentation ended on time.

Milestone 4: Design Review and Critique

Learning objectives:

- Review, understand, and extract high-level design models from an existing application.
- Evaluate an existing design with respect to different quality attributes.
- Apply scenario-based walkthrough to evaluate a design.
- Develop and communicate a critique of a design.

Tasks:

- Read the implementation developed by another team from Milestone #2.
- Document the key aspects of the design using design models.
- Evaluate the design models with respect to different quality attributes (e.g., modularity, reusability, robustness, scalability).
- Prepare and present a constructive critique of the design.

Milestone 5: Redesign for Scalability

Milestone 6: Final Project Report

Part B: Design Improvement
Due: **11:59 pm Monday, Apr 22, 2024**

Task 4: Design Improvement

For the issues that the other team found in your service:

- Describe at least two design options that improve the robustness of your design. If plain text does not clearly communicate the differences, model them using appropriate models that communicate the essential aspects of each design option.
- Identify possible trade-offs between robustness and other relevant quality attributes for each design option.
- Justify which option you would pick if you had to implement the improvement.

Given time constraints, you are **NOT** required to implement the robustness improvements that you come up with. But if you fix them, you will receive **bonus points**.

In the unlikely scenario where the other team has not found any issues in your system, please discuss possible ways in which you could further improve the robustness and/or security of your service by following the same approach described above.

Shared Data / Interfaces: Appointment Requests, Appointments, Test Results, Policies

1. Health-Care Professionals View
- See appointment requests
- Approve appointment
- Enter Test Results (positive / negative)

2. Patients View
- See Test Results
- See Appointment approval
- Apply Quarantine Policy

3. Policy Maker View:
- Change Quarantine Policy
- View Aggregated Test Results
    -

4. Backend team: GDS-like system, maybe for PhD student team?!
- Persistent database storing the data

- Each system only needs to interface with this one system

OR

4. The Public
- See aggregated Test Results
    - Maybe Google Maps Overlay

OR

4. Hospital Administrator
- Assign health care professionals to appointment requests based on their work schedule
-

Map
- Display Test results on a Google map overlay
- Can be a separate web page that receives data from the other system and doesn't store data itself

Doctors
- Maintain patient documents (history)
- Add test results
- Research symptoms
- Automatically Recommend treatment
- Message Patient
- (Order tests, vaccines)

Policy Makers
- Update Testing Requirements
- Update Vaccination Requirements
- Update Quarantine Policies
- Analyze Situation (Aggregation of Test Results)

Patients
- Request appointment (for testing or vaccination)
- View Test Results
- View Health Advice
- Receive Quarantine Request
- Message Doctor

Optimisation System

- Reads appointment requests and schedules appointments
- Distributes Resources for tests and vaccines
-

(Quality Team)

- Design Stress Tests
- Design Design System Tests
-