PV&J

PV&J Simplified Coding Turk Machine
Design Report
For Turk System

Version 1.0

Revision History

Date	Version	Description	Author
16/Nov/17	1.0	First Draft	PV&J

Table of Contents

1. Introduction

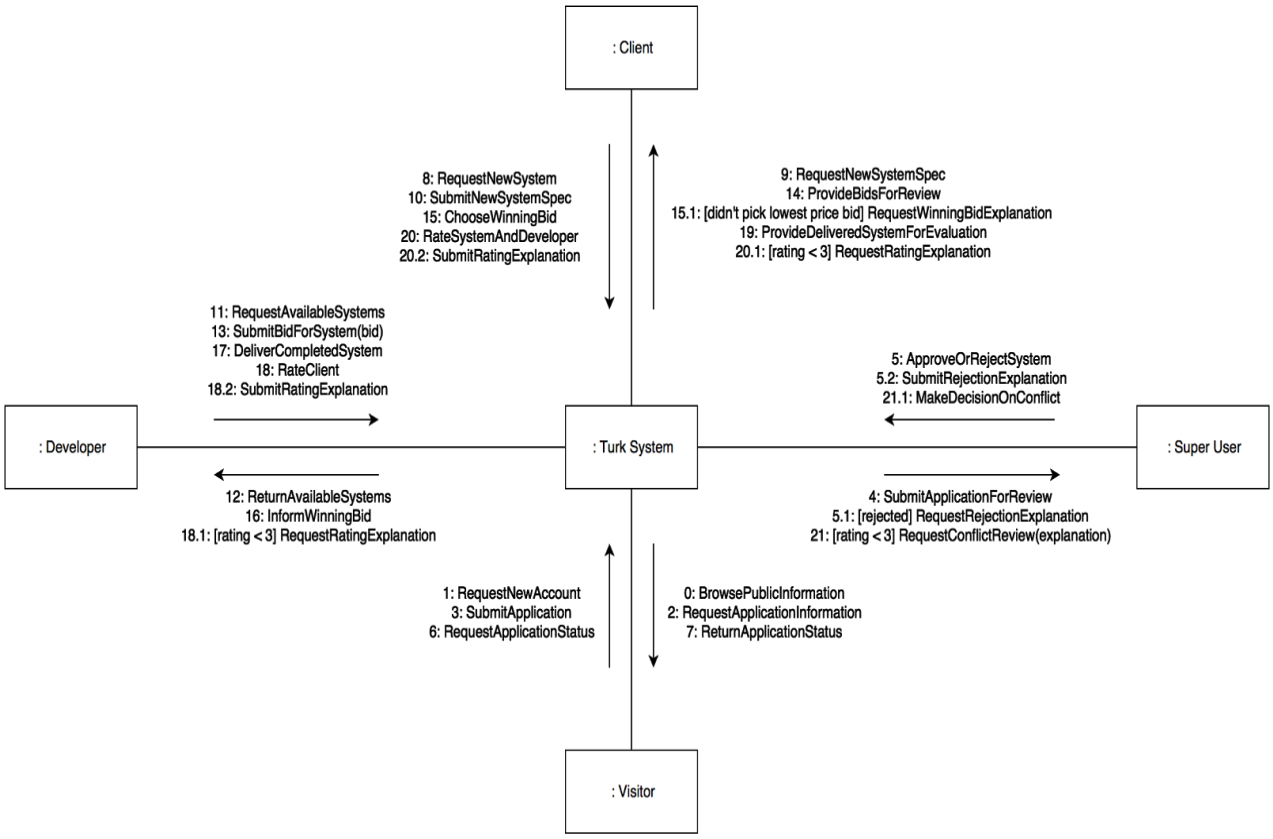
This design report will give an overview of the design for the entire PV&J Simplified Coding Turk System (SCTS). The purpose and scope will be herein clarified along with key definitions for understanding the rest of this document.

1.1 Purpose

This document is meant to detail the functionalities to be carried out by the PV&J Simplified Mechanical Turk System as specified in the earlier specification report. This section focuses on introducing how the system is meant to function.

1.2 Collaboration Class Diagram

The collaboration class diagram below provides an overview of the system. This overview outlines the interactions of the various users with the system and the overall functionality of the system under typical use. More detailed diagrams are provided in the following section outlining each of the use cases mentioned in the specification report.

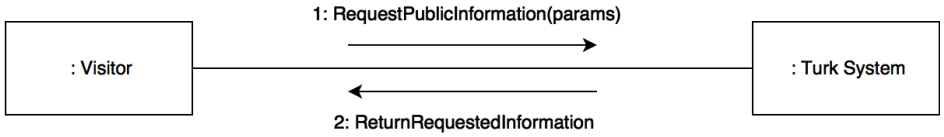


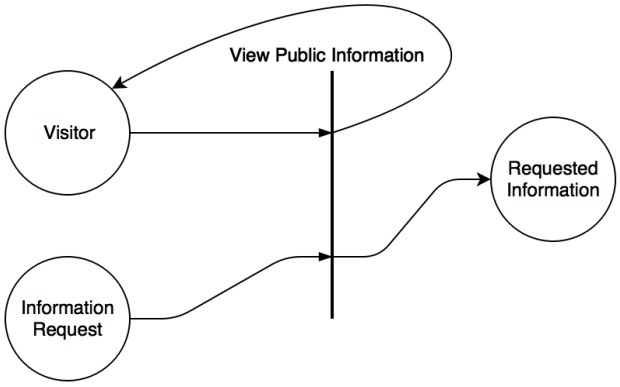
2. Use Case Analysis

In this section, we provide a more detailed overview of each of the use cases mentioned in the specification report. For each, we provide a collaboration class diagrams detailing the interaction between different classes in the system and a petri net detailing the various processes involved with each use case.

1. Browse Public Information

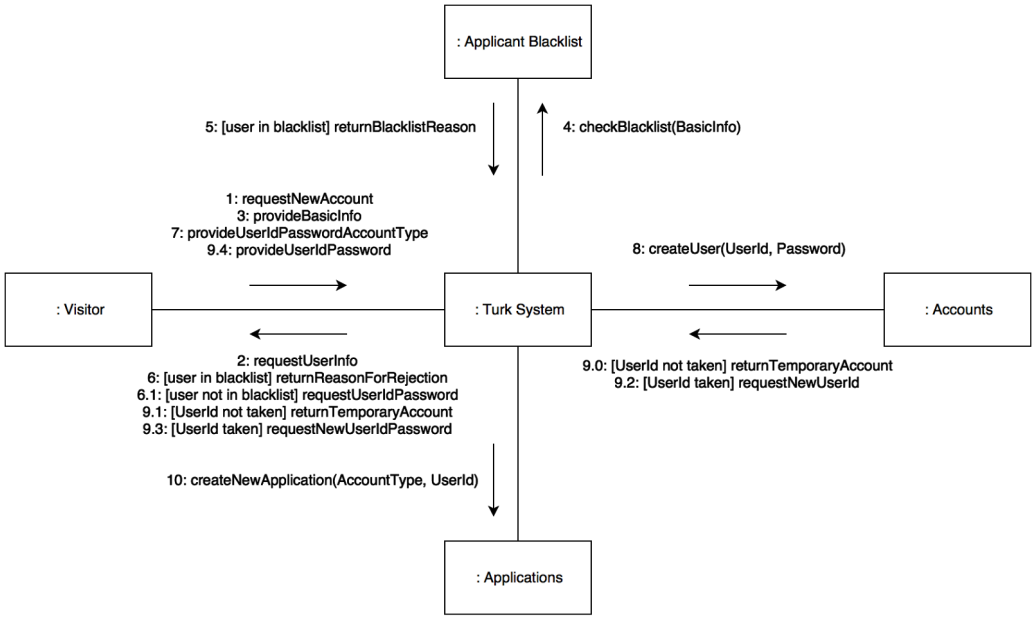
Visitors can browse and search public information without having to be registered into the Turk System. They can communicate with the Turk system and request such information through a request specifying the information desired.

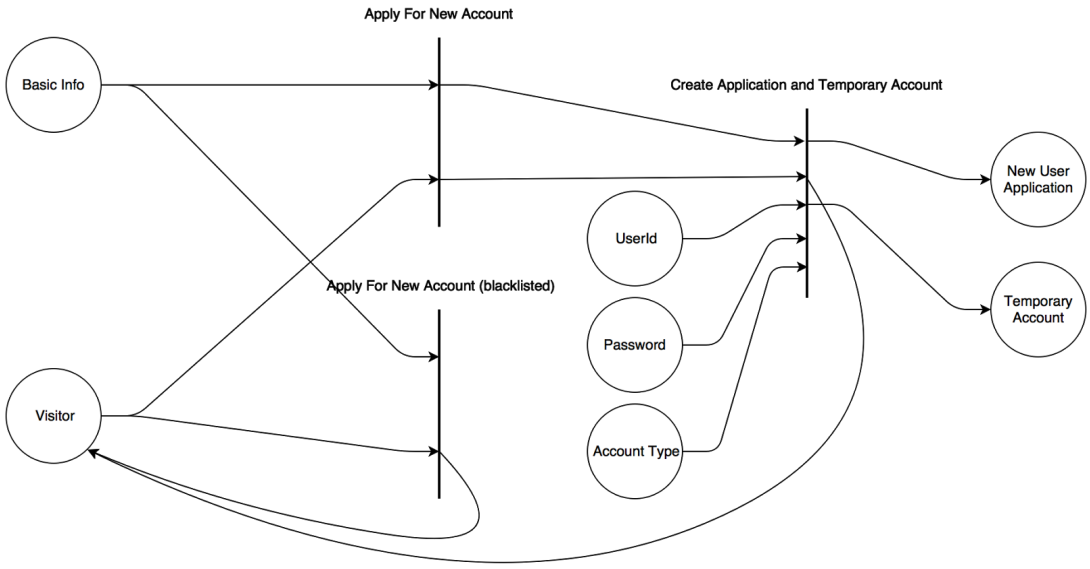




2. Request New Account

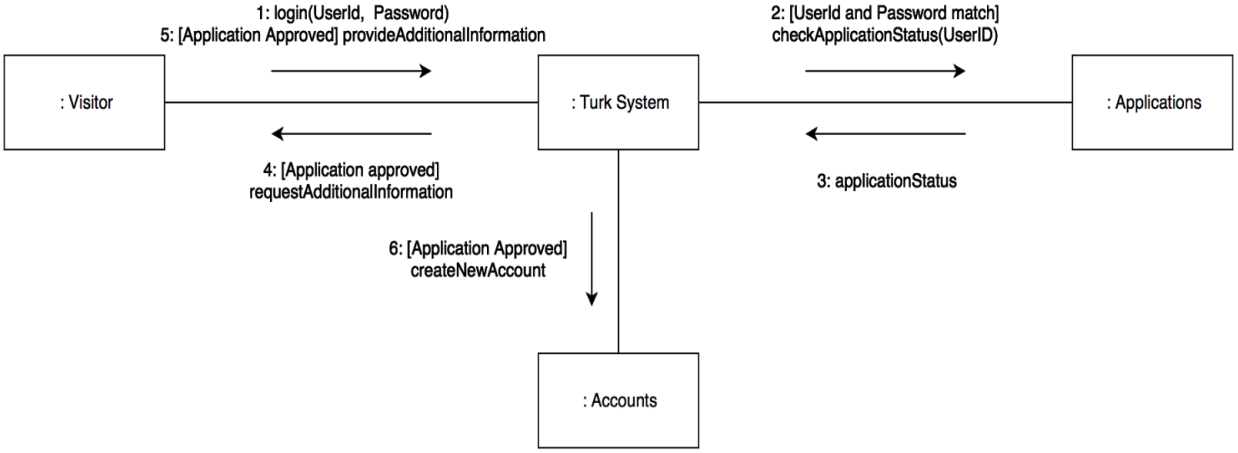
Visitors can register for an account on the Turk System by applying to be either a client or developer. The Turk System would verify that the visitor is not in a blacklist. If the user is in a blacklist, the system will automatically reject the application. Otherwise, the applicant is prompted to select a UserId, which is verified to be unique by the Turk System, and Password. This information is used to create a new temporary account and membership application.

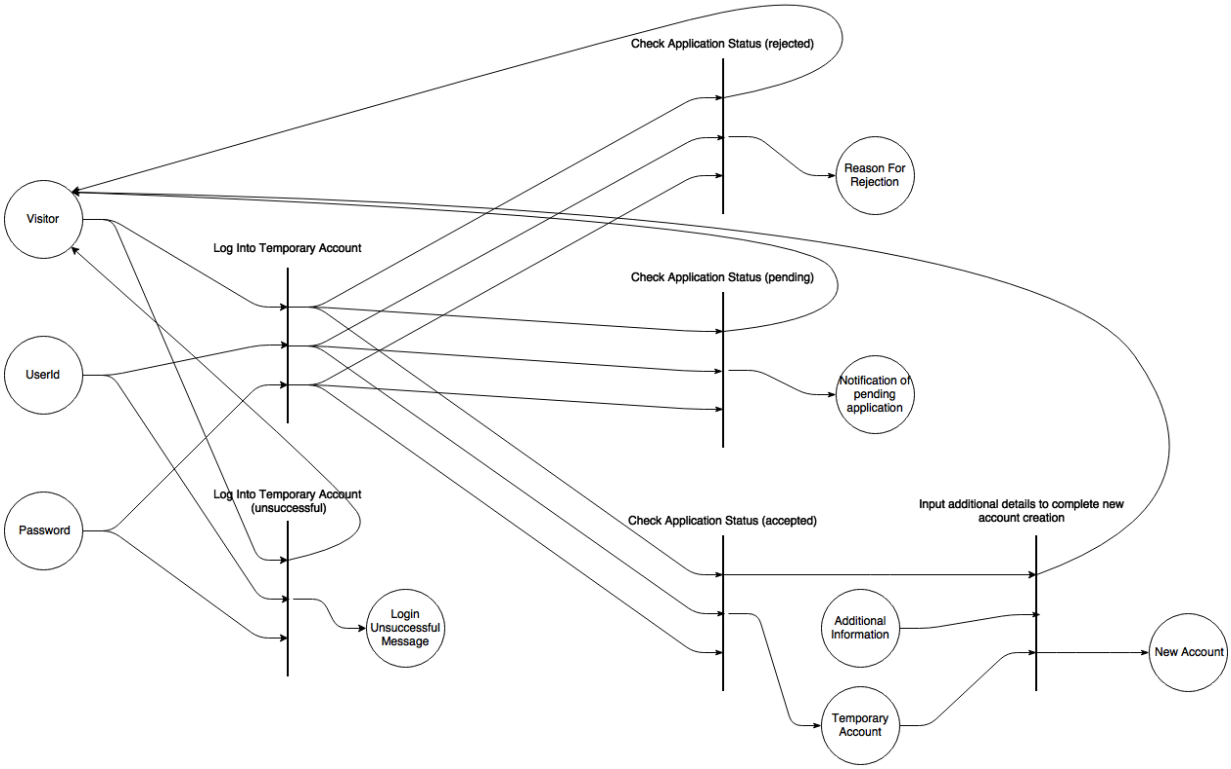




3. Check Application Status

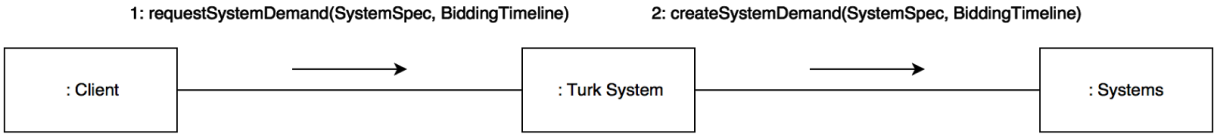
Applicants can log into their temporary account in order to check the status of their application. The Turk System will log the applicant into their temporary account. Based on the status of the application, it will either display a notification of their pending application, a reason for their rejection, or prompt the user for further information. In the latter case, the information will be used to create a permanent account of the type that the user requested in the application.

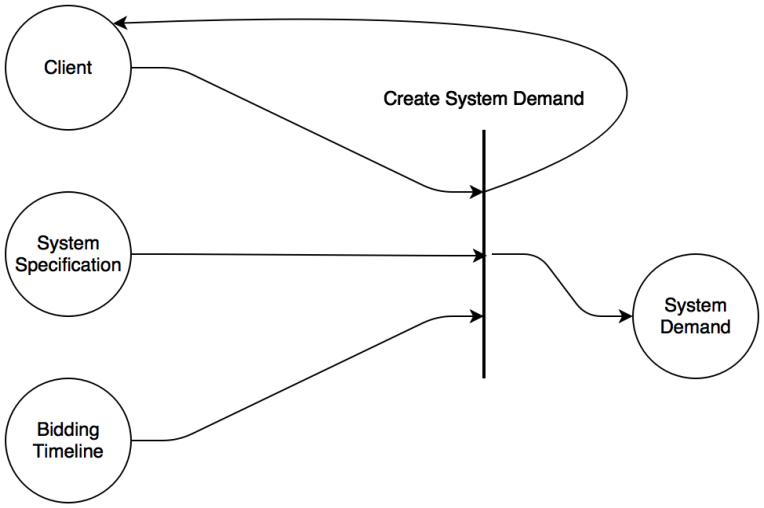




4. Create System Demand

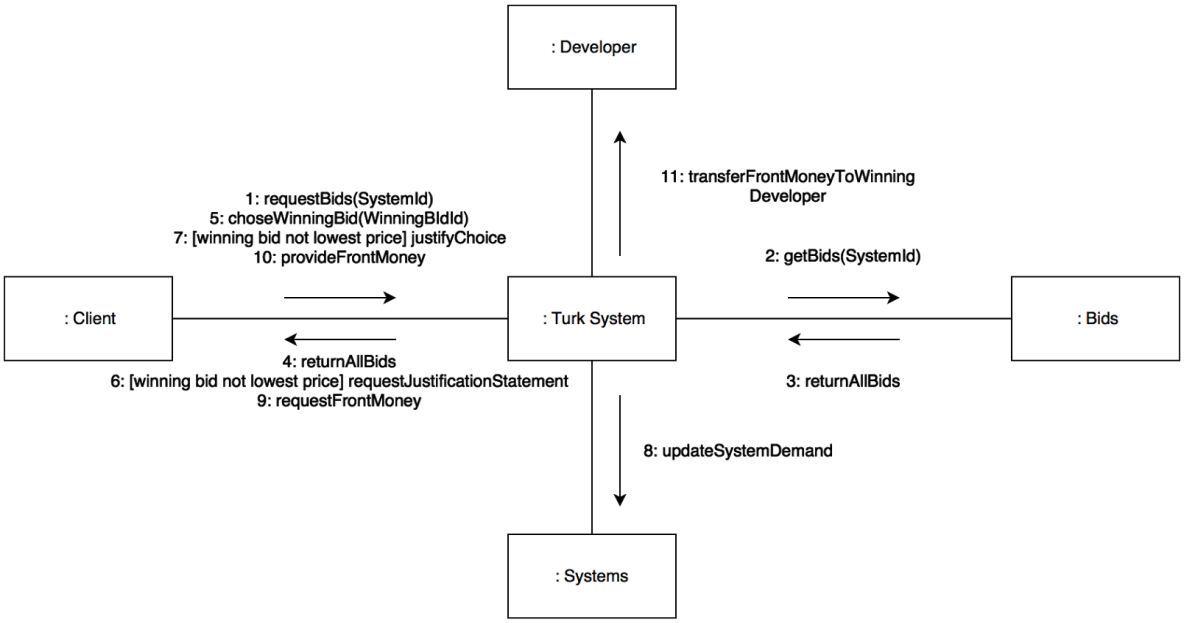
Clients are able to create system demands in the Turk System in order to find developers. In order to do so, the client must provide the Turk System with a specification for the desired system and a bidding timeline during which the system demand will be posted on the Turk System.

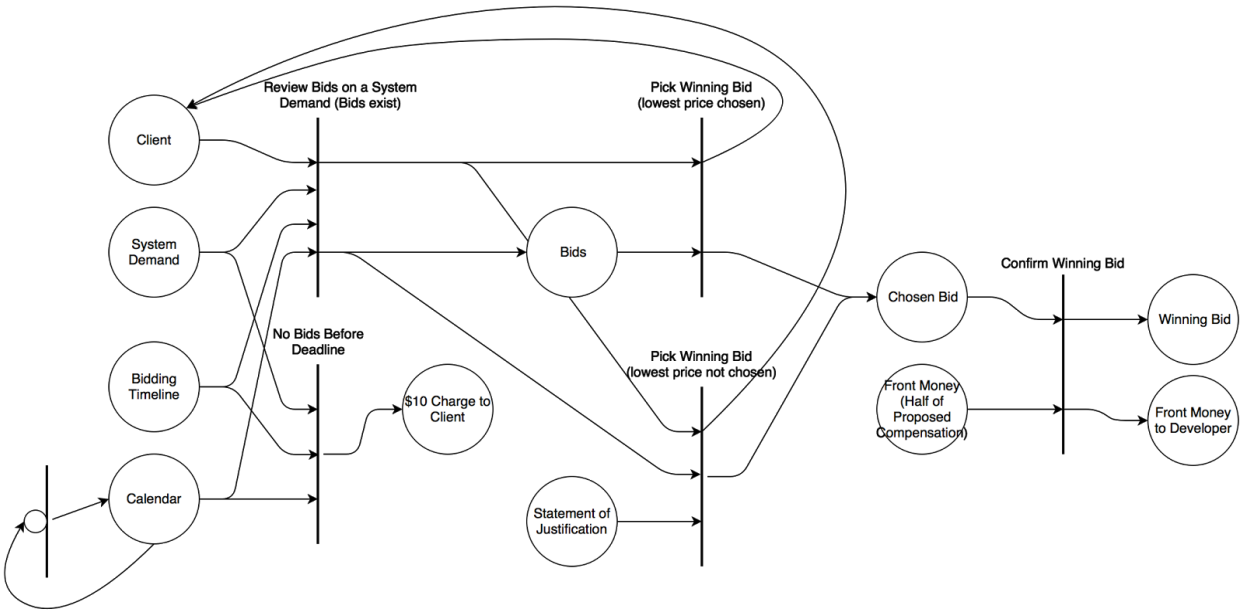




5. Choose Winning Bid

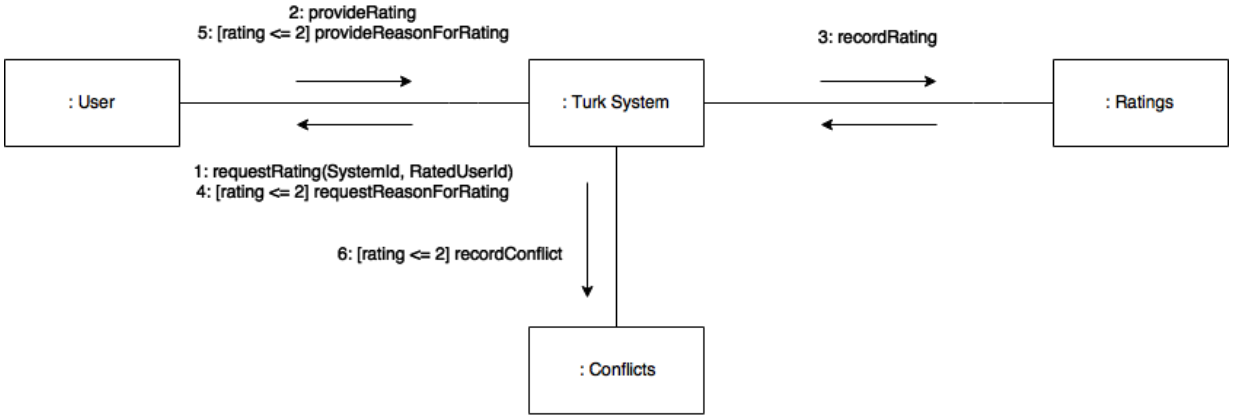
Clients are able to pick the winning bid for their system demands after the bidding timeline was expired. If there are no bids during this timeline, the demand is removed and the client is charged $10. Otherwise, the client can view and choose from the bids on the system demand with the condition that if they choose a bid that does not offer the lowest price, they must submit a statement describing why they chose that particular bid.

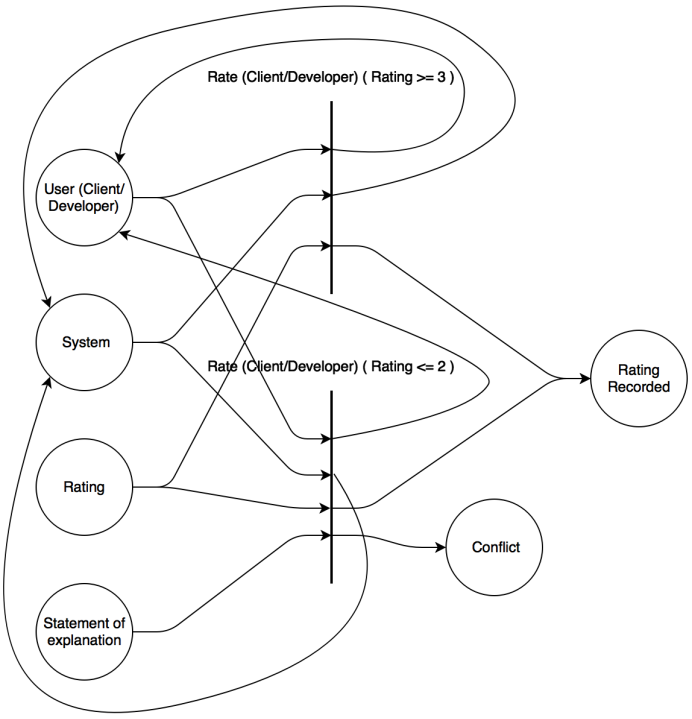




6. Rating Use Case

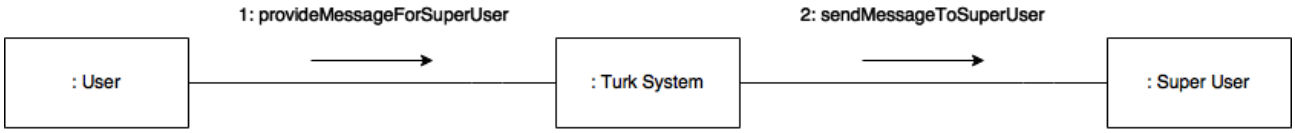
Clients are able to rate a system and the developer they hired upon completion of the contract. Developers may rate clients based on any potential issues or pleasant experiences working with them. Either user types must explain his or her rating if he or she provides a rating less than 3.

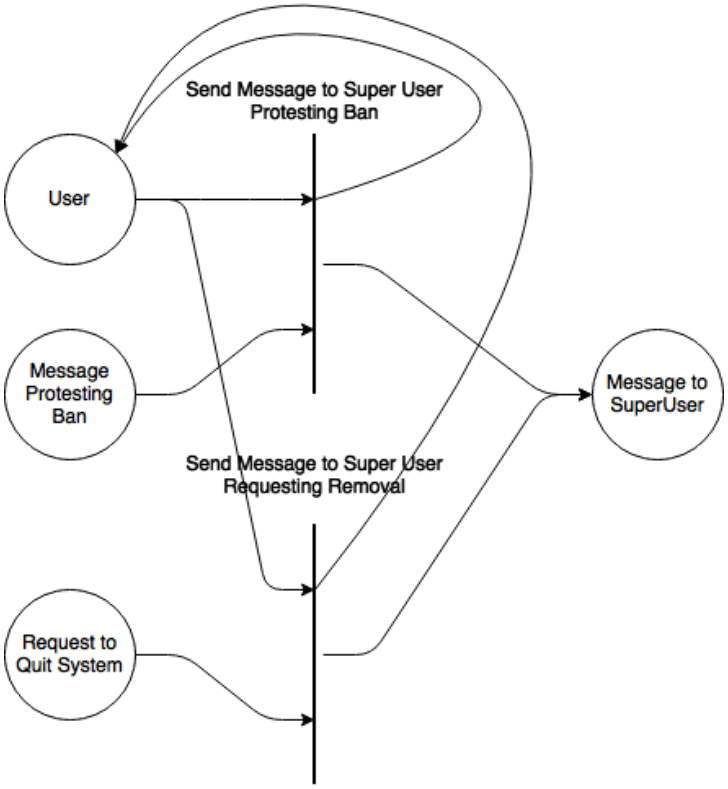




7. Contact Super User Use Case

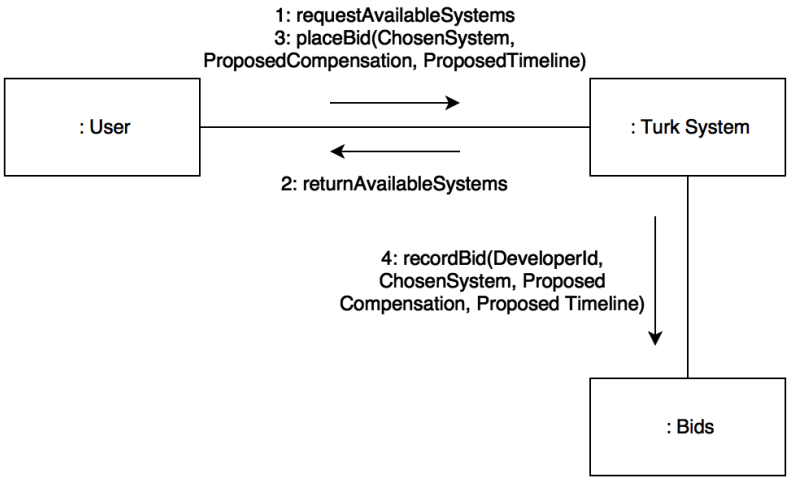
Registered users can protest to the super user about a warning or ban that they received. Registered users can also request to quit the system if they do not see themselves using the Turk System in the future.

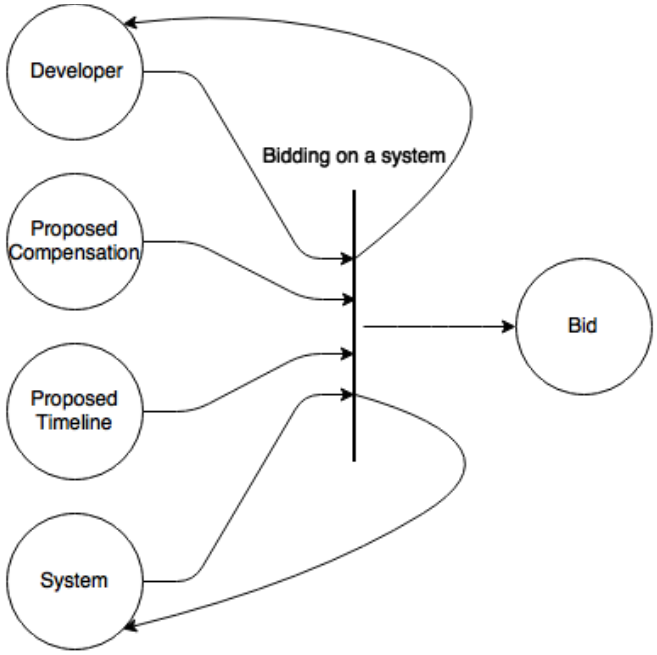




8. System Bidding Use Case

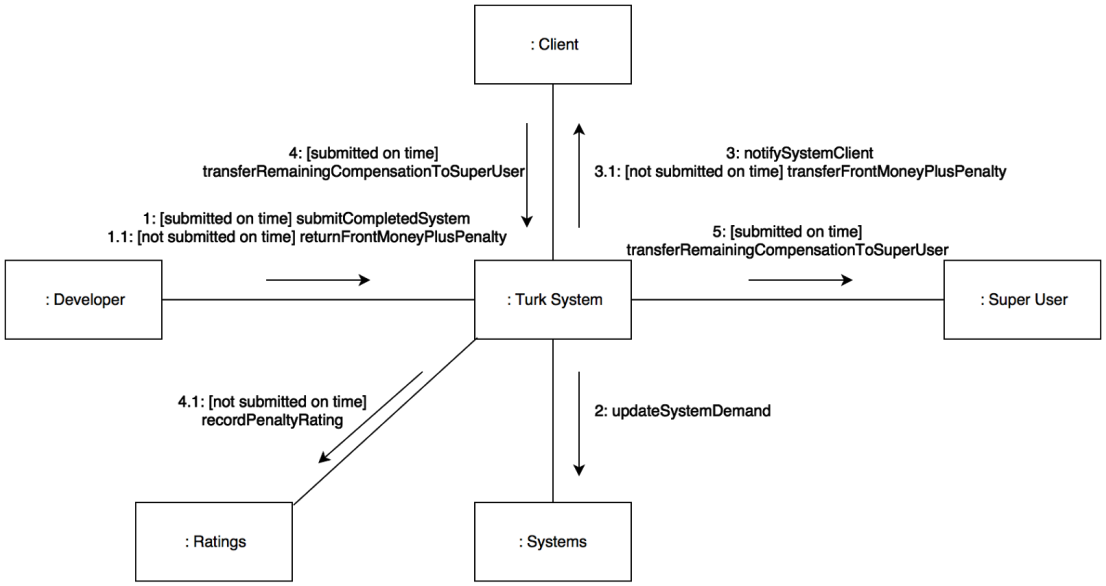
Developers can bid on a system with their proposed timeline and cost. Clients are able to pick the winning bid from the developers based on their own criteria.

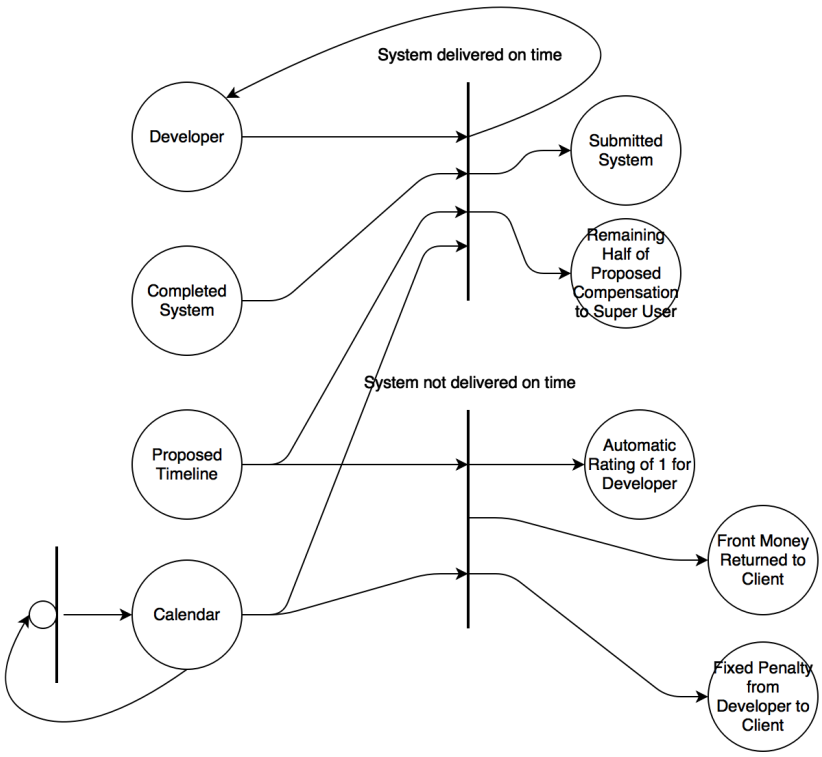




9. Submission Use Case

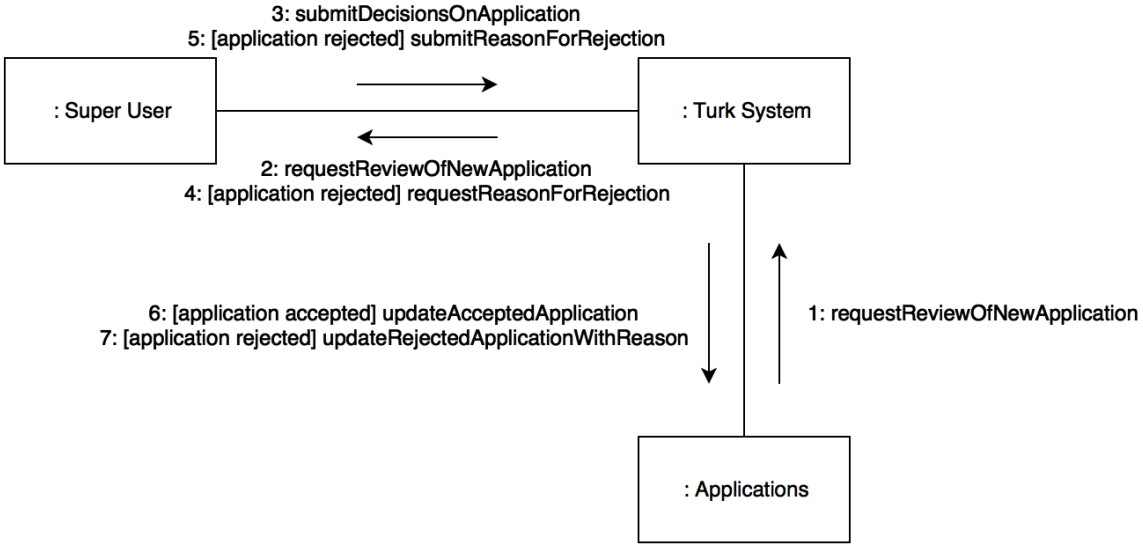
Developers can use a submission form to deliver their completed system by their proposed deadline. Otherwise they will not be able to and receive an automatic bad rating.

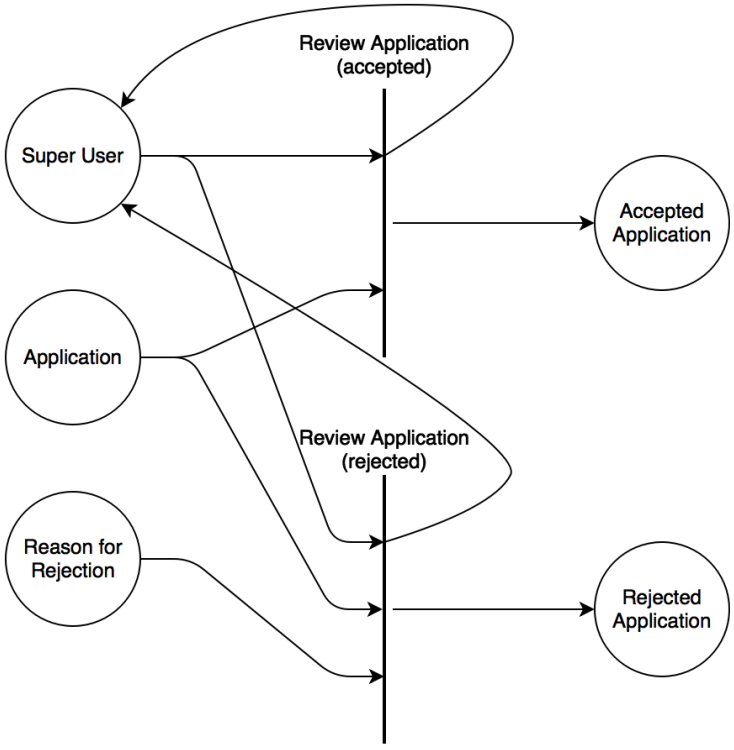




10. Review New User Use Case

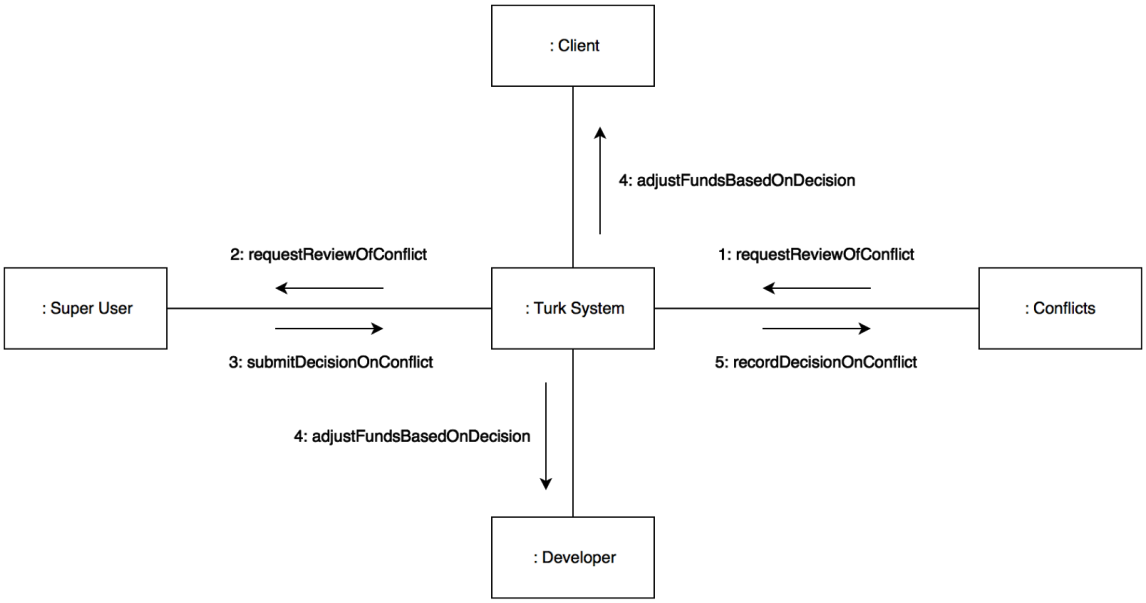
Super Users can use their administrative privileges to accept or reject a new user. They are required to explain a reject with a brief summary of their justifications. Both of these actions are done after reviewing a new user application.

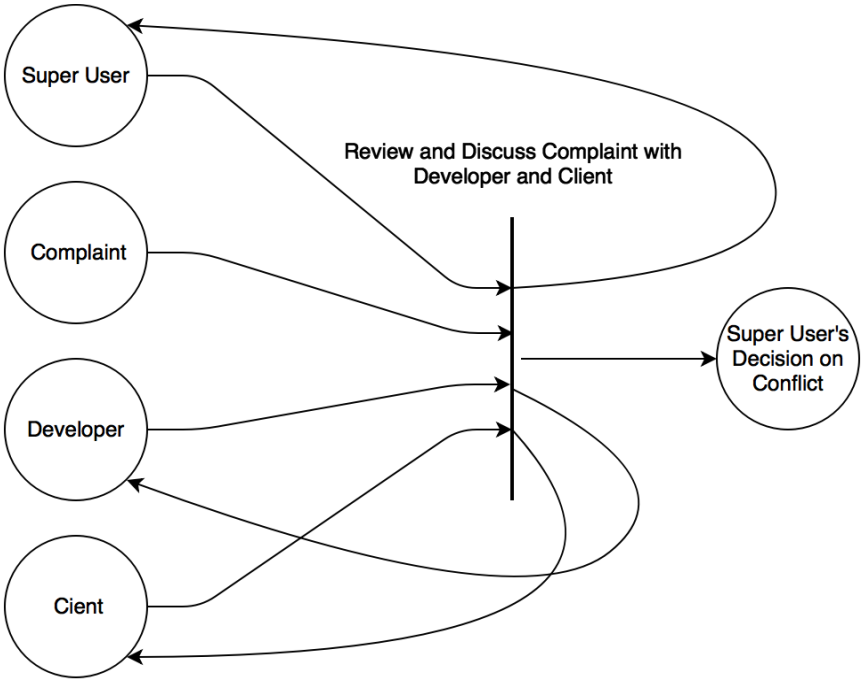




11. Evaluate Conflicts Use Case

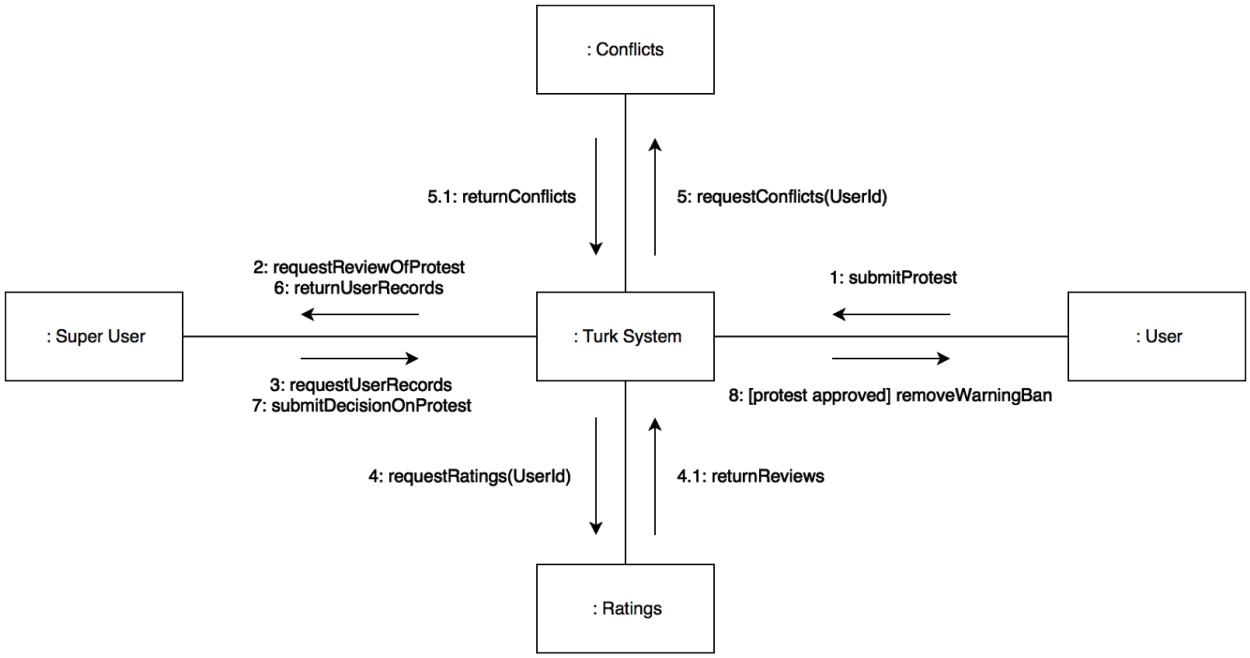
Super Users can resolve complaints that are submitted by clients about a submitted system. They will have the ability to evaluate the complaints, must discuss with the client offline, and determine how to distribute the remaining money from the contract.

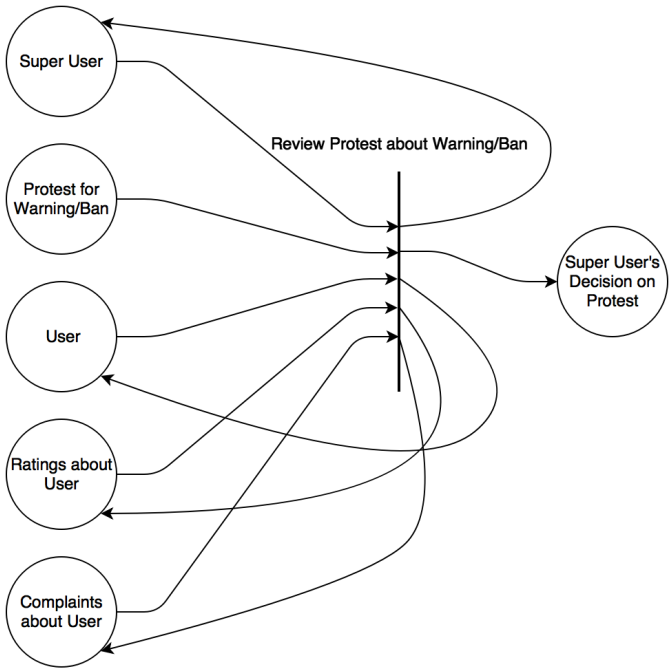




12. Review Warning/Ban Protest Use Case

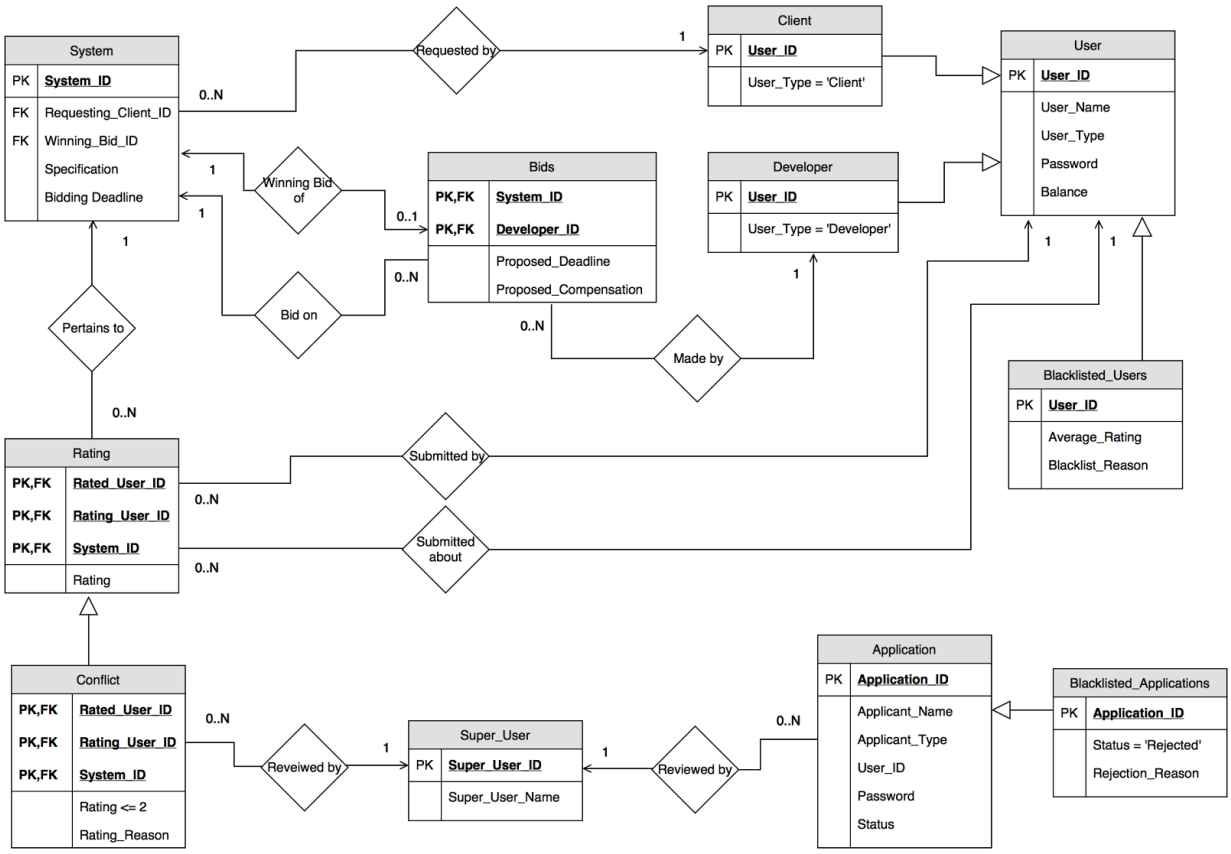
Super Users can review protests from warned/banned users and reverse the system's decision should they determine it was not justified.





3. Entity-Relation Diagram

Below is an Entity Relation Diagram describing the overall system and outlining our database.



4. Detailed design

Pseudocode is written below for each of the functions we planned for the project.

0. BrowsePublicInformation

Visitor -> Turk System

- Input: None
- Output: Public Information Page
- Function: Allow user to view public information of the system. This method should be available for all types of users.

```
def BrowsePublicInformation():
    return get(statistics, postings)
```

1. RequestNewAccount

Visitor -> Turk System

- Input: Username, Password, Type (client or developer)
- Output:
- Function: Allows guests (visitors) to make a new account.

```
def RequestAccount(User_Info, User_Id, Password, Type)
        If UserID not taken:
                If user not in blacklist:
                        TempAccount := CreateTemporaryAccount(userID, password, type)
                        CreateApplication(User_Info, Temp_Account)
                        Return TempAccount
                else:
                        DenyApplication()
        else:
                PromptToChooseNewUser_Id()
```

2. RequestApplicationInformation

Turk System -> Visitor

- Input: None
- Output: Application Information
- Function: Returns the applicant's information to the visitor.

```
# Turk System needs application information from Visitor
# Turk System sends out request to fill out application
def RequestApplicationInformation():
        return ApplicationForm
```

3. SubmitApplication

Visitor -> Turk System

- Input: Application
- Output: Submission Success?
- Function: Submits the visitor’s application to the Turk System.

```
# Visitor has the application
# Visitor fills out application with credentials
# Visitor submits application to the Turk System
# Application information entered into DB
def SubmitApplication(application):
    EnterApplicationToDB(application)
```

4. SubmitApplicationForReview

Turk System -> Super User

- Input: None
- Output: Application
- Function: Submits the new visitor’s application to the super user.

```
# Visitor application available in Turk System
```

```
# Turk System sends out application and its fields to super user
def SubmitApplicationForReview(form):
        InsertFormIntoDB(form)
        Redirect(home)
```

5. ApproveOrRejectSystem

Super User -> Turk System

- Input: Approval/Rejection
- Output: None
- Function: Allows the super user to approve or reject a system request.

```
# Super User reviews the system
# Super User either accepts or rejects the system
# If accepted, put system into DB, update application status to accepted.
# Else (rejected), tell user it was rejected, update application status to rejected.
def ApproveSystem(system_id):
        SystemStatusToDB(accepted)
        CreateSystemInDB(system_id)

def RejectSystem(system_id):
        SystemStatusToDB(rejected)
        DisplayRejectionToUser(system_id))
```

5.1. [rejected] RequestRejectionExplanation

Turk System -> Super User

- Input: None
- Output: Request
- Function: Requests a rejection explanation to the super user.

```
# Turk System may request an explanation for the system rejection
# Send a request for rejection explanation to the Super User
def RequestRejectionExplanation():
        SendRejectionExplanationRequest(super_user)
```

5.2 SubmitRejectionExplanation

Super User -> Turk System

- Input: None
- Output: Request
- Function: Submits a request for a rejection explanation to the Turk System.

```
# Super User submits rejection explanation
# Submit the explanation for rejection to the Turk System.
# Put explanation in DB.
def SubmitRejectionExplanation():
        SendRejectionExplanation()
```

6. RequestApplicationStatus

Visitor -> Turk System

- Input: User_Id
- Output: Request
- Function: Submits a request for application status to the Turk System.

```
# Visitor can request application status
# Visitor submits a request for application status
def RequestApplicationStatus(user_id):
        SendApplicationStatusRequest(user_id)
```

7. ReturnApplicationStatus

Turk System -> Visitor

- Input: User_Id
- Output: Application Status
- Function: Returns the application status to the visitor.

```
# Turk System fetches application status from DB
# Turk System returns the application status to the user
def ReturnApplicationStatus(user_id):
        GetApplicationStatus(user_id)
```

8. RequestNewSystem

Client -> Turk System

- Input: None
- Output: Request
- Function: Submits a new system request to the Turk System.

```
# Client submits a request for new system
# Client gives new system a name
# Assign the new system a system_id, place in DB
def requestNewSystem(system_name):
        System_id = convertToInt(system_name)
        createSystemInDb(system_id)
```

9. RequestNewSystemSpec

Turk System -> Client

- Input: System specs
- Output: Request
- Function: Submits the new system request to the super user.

```
# Turk System gets the system_id in question
# Turk System submits a request for system specifications for this system to the Client
def RequestNewSystemSpec(system_id):
         user _id = GetUserClientID()
        SendSystemSpecRequest(user_id)
```

10. SubmitNewSystemSpec

Client -> Turk System

- Input: System specs
- Output: System specs
- Function: Submits the specs for a new system to the Turk System.

```
# Client submits system specification
# Put system specification in DB

def submitNewSystemSpec(system_id, spec):
        createSpecInDB(system_id, spec)
```

11. RequestAvailableSystems

Developer -> Turk System

- Input: None
- Output: Request
- Function: Submits a request to view all available systems data from the Turk System.

```
# Developer submits a request to view all available systems
def RequestAvailableSystems():
        SendAvailableSystemsRequest()
```

12. ReturnAvailableSystems

Turk System -> Developer

- Input: None
- Output: Available Systems
- Function: Returns all available systems data from the Turk System, to the developer.

```
# Turk System fetches list of all available systems from DB
# Returns the list of available systems to Developer
def ReturnAvailableSystems():
        systems = getAllSystems()
        return render(systems)
```

13. SubmitBidForSystem

Developer -> Turk System

- Input: Bid
- Output: None
- Function: Submits a bid for a system in the Turk System.

```
# Developer can submit bid for a system
# Developer inputs their bid amount to the system
# Turk System places bid into DB into the table for corresponding Developer
def SubmitBidForSystem(bid):
        LookUpInDB(system_id)['bid'].append(bid)
```

14. ProvideBidsForReview

Turk System -> Client

- Input: None
- Output: All Bids
- Function: Returns all bids available for a system to the client.

```
# Turk System fetches all bids made on the system
# Returns all bids to the Client
def ProvideBidsForReview():
        return getFromDB(bids)
```

15. ChooseWinningBid

Client -> Turk System

- Input: Winning Bid
- Output: None
- Function: Submits the winning bid chosen by the client to the Turk System.

```
# Client chooses winning bid from all bids made on the system
# Place winning bid into DB
def ChooseWinningBid(bid):
        If bid in getAllBidsFromDB():
        putInDB(bid)
```

15.1. [didn't pick lowest bid] RequestWinningBidExplanation

Turk System -> Client

- Input: None
- Output: Request
- Function: Submit a request for an explanation of why the winning bid won from the Turk System.

```
# Submit a request for a winning bid explanation
def RequestWinningBidExplanation():
        SendWinningBidRequest()
```

16. InformWinningBid

Turk System -> Developer

- Input: None
- Output: Inform
- Function: Returns a notification informing the developer with the winning bid that they won.

```
# Fetch the winning bid from the DB
# Returns the winning bid (system, amount) to the Developer.
def InformWinningBid():
        WinningBid = getWinningBidFromDB()
        InformWinningBid(getDeveloper(WinningBid)
```

17. DeliverCompletedSystem

Developer -> Turk System

- Input: Completed System

- Output: None
- Function: Submits a completed system to the Turk System.

```
# Developer submits completed system to the Turk System.
# Developer submits finished system to the DB
#Turk System marks the system as completed (remove from list of available systems)
def DeliverCompletedSystem(system_id):
        SendSystemInDB(system_id)
        MarkSystemFinished(system_id)
        SendNotificationFinishedToClient(user_id)
```

18. RateClient

Developer -> Turk System

- Input: Rating
- Output: None
- Function: Submits a rating of a developer's clients to the Turk System.

```
# Developer submits a rating for the client
# Enter rating into DB
def RateClient(client, rating):
        TheClient = getFromDB(client)
        PlaceRatingInDB(rating)
```

18.1. [rating < 3] RequestRatingExplanation

Turk System -> Developer

- Input: None
- Output: Request
- Function: Submits a request for a rating explanation from the Turk System.

```
# If rating < 3, client can submit a request for rating explanation
# Client submits a request for rating explanation
# Fetch rating of the client from DB, verify it is < 3
# Turk System sends request to Developer
def RequestRatingExplanation():
        If rating < 3:
                SendRatingExplanationRequest(user_id)
        Else:
        flash('Rating too high')
```

18.2. SubmitRatingExplanation

Developer -> Turk System

- Input: explanation
- Output: Output
- Function: Submit the rating explanation to the Turk System.

```
# Developer submits rating explanation
# Input rating explanation into DB
def SubmitRatingExplanation():
```

SendRatingExplanation(explanation)

19. ProvideDeliveredSystemForEvaluation

Turk System -> Client

- Input: system_id
- Output: Delivered System
- Function: Returns the delivered system to the client for evaluation.

```
# Fetch completed system from DB
# Return completed system to Client for evaluation
def ProvideDeliveredSystemForEvaluation(system_id):
        return GetSystemFromDB(system_id)
```

20. RateSystemAndDeveloper

Client -> Turk System

- Input: Rating
- Output: None
- Function: Submits a rating from the client for the system and developer.

```
# Client submits a rating of the System
# Client submits a rating of the Developer
# Save rating into DB
def RateSystemAndDeveloper(rating):
        SendRatingSystem(system_id)
        SendRatingDeveloper([user_id,...])
```

20.1. [rating < 3] RequestRatingExplanation

Turk System -> Client

- Input: system_id
- Output: Request
- Function: Submits a request for a rating explanation from the client (if the rating < 3).

```
# Check if the rating the client submitted (either for the developer or the system) is < 3
# Turk System submits a request for rating explanation to the client
def RequestRatingExplanation(system_id):
        If rating < 3:
                user_id = FindUserInDB(system_id)
                SendRequestRatingExplanation(user_id)
        Else:
                flash(“Rating too high to explain)
```

Client -> Turk System

- Input: Rating Explanation
- Output: None
- Function: Submits the rating explanation to the Turk System from the client.

```
# Client submits rating explanation to the Turk System
```

```
# Store explanation in DB
def SubmitRatingExplanation(explanation):
        FindUserInDB(user_id)['rating'].append(explanation)
```

21. [rating < 3] RequestConflictReview(explanation)

Turk System -> Super User

- Input: None
- Output: Request
- Function: Submits a request for a conflict review if the rating < 3.

```
# Check if rating < 3
# Submit a request for conflict review to the Super User
def RequestConflictReview():
        SendConflictReviewRequest(super_user)
```

21.1. MakeDecisionOnConflict

Super User -> Turk System

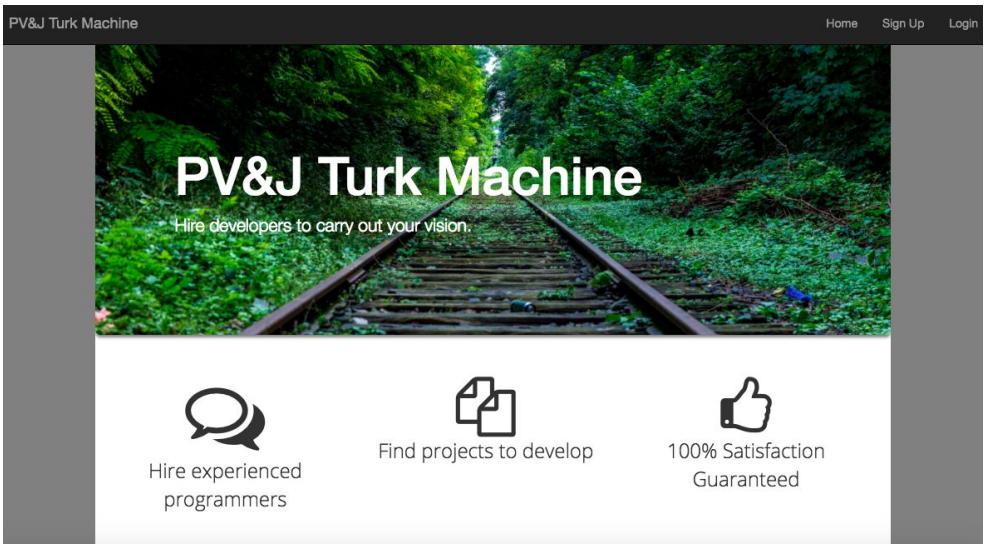
- Input: Conflict
- Output: Decision
- Function: Super user submits a decision on the conflict between client and developers.

```
# Super User inputs their decision on the conflict
# Store decision in DB
# Notify client and developer of the conflict decision
def MakeDecisionOnConflict(conflict_id):
        StoreDecisionInDB(conflict_id)
        SendDecisionToUsers(conflict_id)
```

5. System screens:

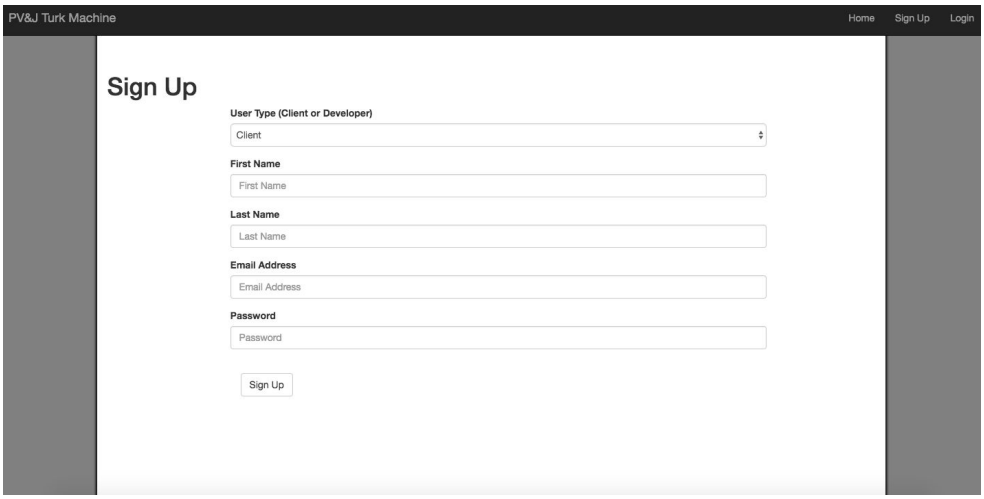
Here, we demonstrate major GUI screens of the system and showcase a prototype of the bidding functionality in the Turk Machine.

Landing Page



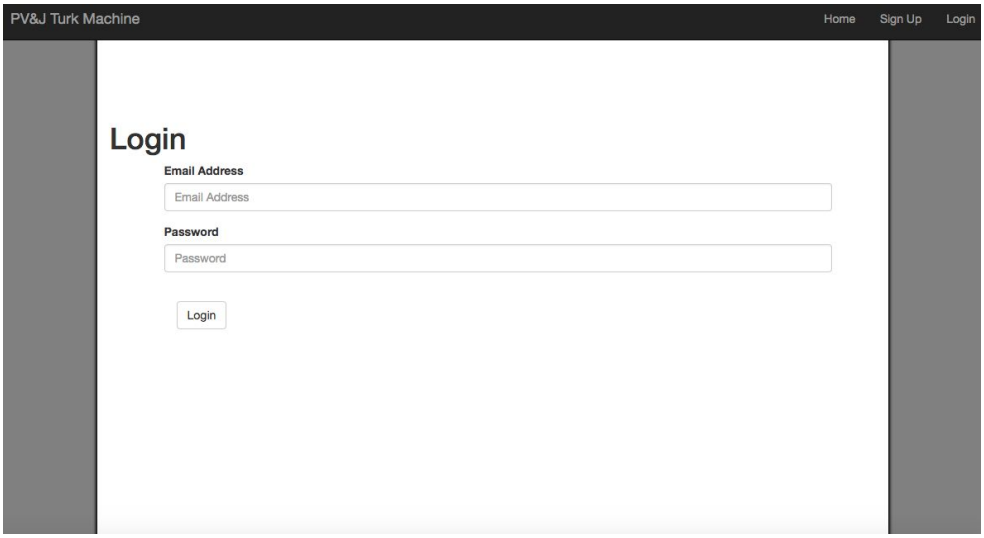
This is the landing page that the user will see. There will be buttons to signup and login. There is a brief glimpse of the value proposition for this Turk Machine product.

Sign up



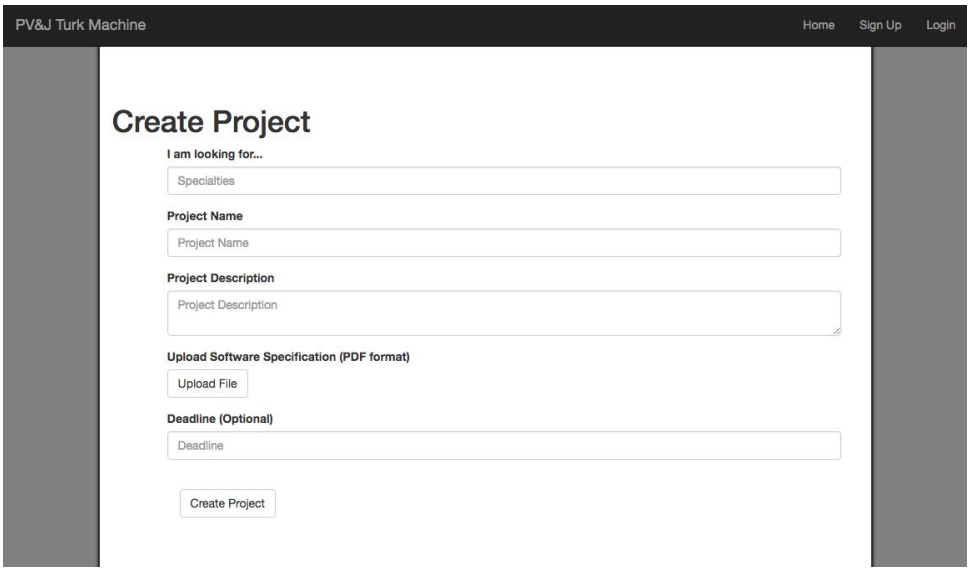
This is a standard sign up page that allows the user to specify what type of account they want to create. They can choose to become a client or developer.

Login



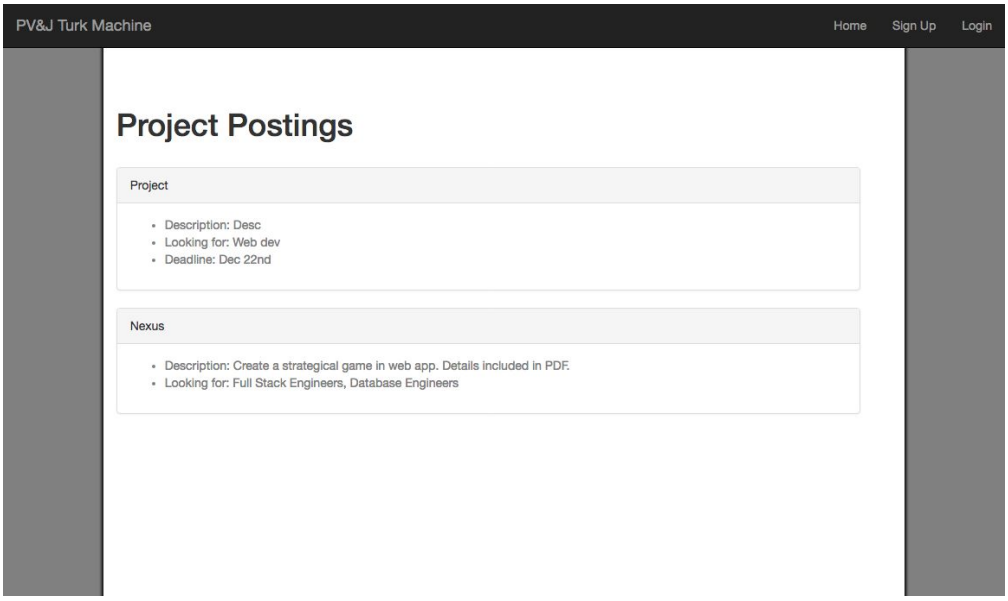
The login page only asks for email address and password credentials. This will create a session when we implement authentication, which we prioritized for last.

Create Posting



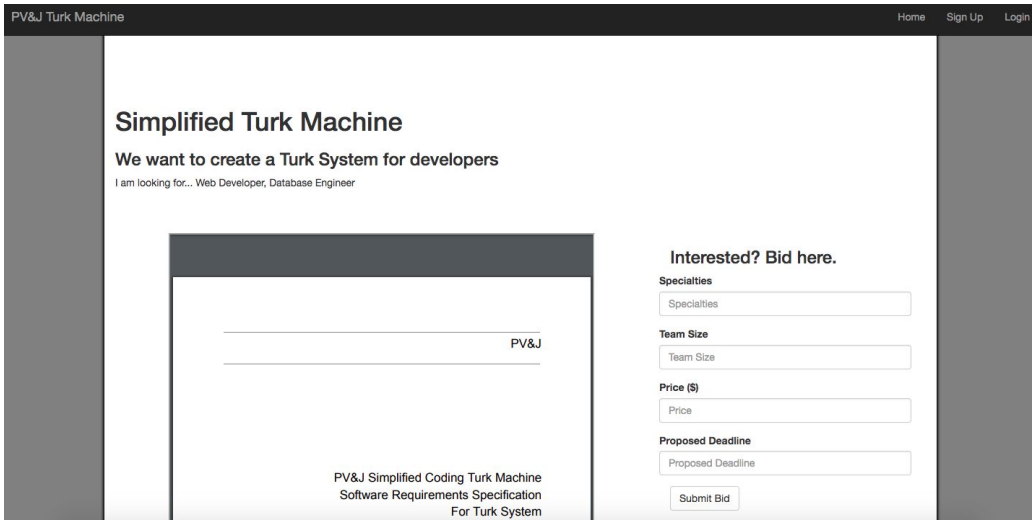
To create a project at the '/create' route, a form will be provided to fill out what type of developers the client is looking for. They will be asked to upload a PDF Software Specs so that the developers will know what to create. Deadlines are always floating and are listed as optional here unless there is a hard deadline. The Flask backend will process this data and store it into a JSON file for future retrieval.

View current postings



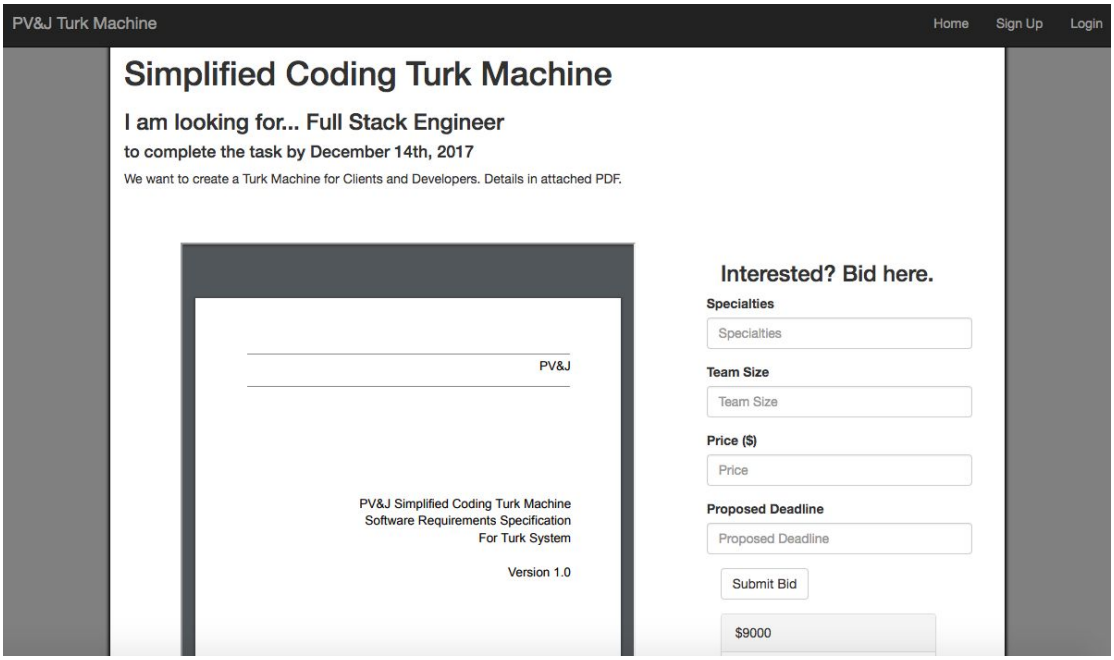
All project postings can be viewed at the route '/posts.' This will allow developers to view all of the possible projects that they can bid on. All of this data is rendered using actual data stored in the assets folder, which stores all of the project data that was created in '/create.'

View post and make bid

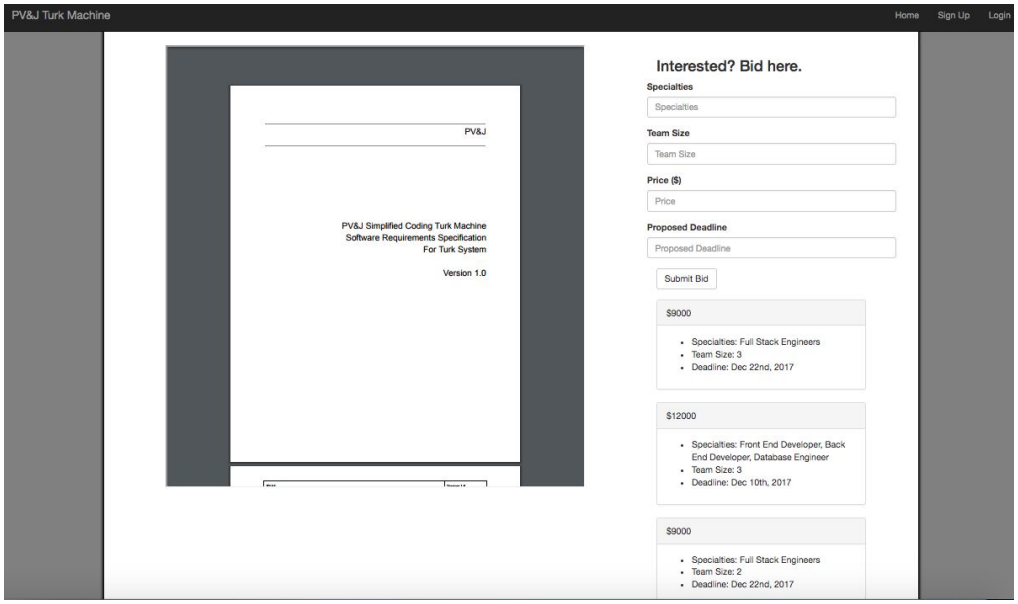


By clicking on any of the project listings in '/posts,' anyone can see the Software Specs and everything else they need to know about the project they may want to work on. If interested, a developer can bid on the right column with the developer roles, team size, prize, and proposed deadline that they can achieve. After the bid is submitted, it is added to a list of bids displayed at the bottom of this page. This will be demonstrated in the next section, 'Prototype.'

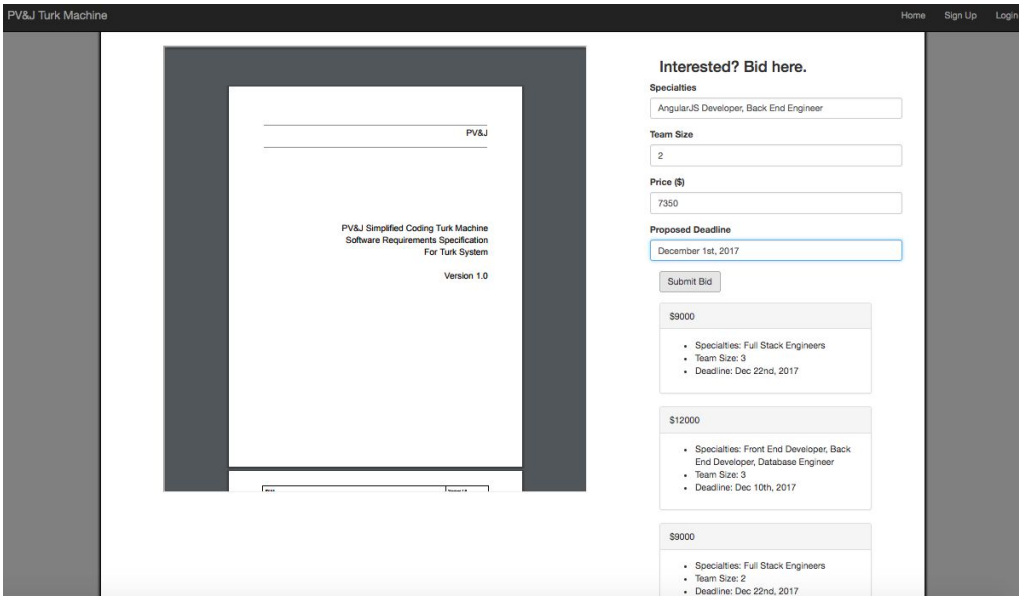
Prototype (Bidding)



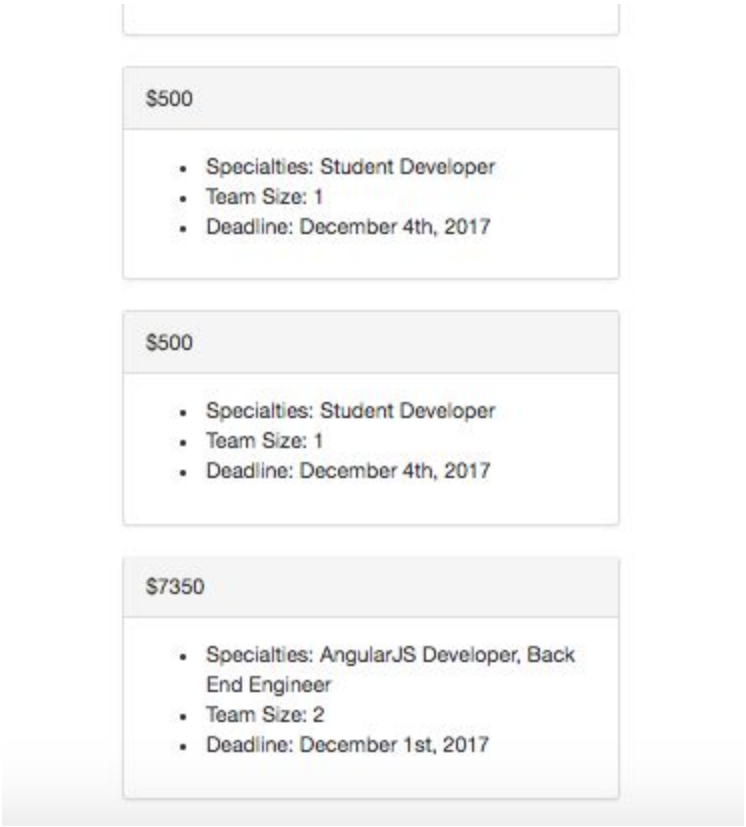
The form allows for bidding by developers on different projects.



All of the bids displayed are rendered straight from stored JSON files. Jinja2 engine takes all of the data sent from the backend and displays it.



Here, we input a new bid to demonstrate how the bids are added.



When scrolled down, the bottom of the bids show the newest bid that was added. Currently, all bids are displayed in chronological order by when they were added. We'll soon find a new configuration for displaying all of the bids effectively in a manner that is useful to the client that created the project post.

6. Meeting Minutes for all Group Meetings

October 16th, 2017

- Decide which use cases to prioritize
    - What cases were required before others
- What constitutes Minimal Viable Product for this project
    - Needs to have two types of users (Client and Developer)
    - Project listing system
    - Project bidding system
- Front end technology
    - Jinja2 rendering
- Back end technology
    - Flask web server (chosen)
    - Discussion around using Node.js (not chosen)
- Database?
    - Use normal files to store all data for now until MVP done

October 23rd, 2017

- Defined use cases in depth
    - How are we going to target these use cases
    - How to develop each use case
    - Importance and order of implementing use cases
- Assigned tasks to get started on the work
    - Writing pseudocode and thinking about functionality
    - Starting to think about relationships and entities
- Running through the program flow
    - What is the typical use case for a user on this system
    - What is the flow of each use case

October 30th, 2017

- Worked on the general Entity Relationships Diagram
    - Show the relationships between the objects
    - Easier to design database schemas later on
    - How to make it efficient and thinking about normal forms
- Defined methods and wrote pseudocode for relations
    - Easier to create using pseudocode

November 1st, 2017

- Drew Petrinets for the Turk System
    - How resources will be allocated in the system
    - Refresh on how to properly draw petrinets effectively

- Split more work on Flask web server
    - Based on routes and views

November 8th, 2017

- Drew general Collaborations Diagram for entire Turk System and its entities
    - Better demonstrate flow of the data
    - Easier to design file system in lieu of database for now to test core functionalities
- Discussed about all the views for the web app
    - Created low fidelity wireframes using HTML/CSS

November 15th, 2017

- Created a file system that will enable storing of data
- Data is propagated straight from JSON files in folders specific to the project ID
    - Made sure that this can easily be translated to a database schema
    - Discussed methods of doing so and it seemed feasible
- Worked on Collaboration Diagrams for use cases
- Used collaboration diagrams to build working prototype of Turk System
- Styled the views and created the rest of the views (posts, create post, bids)
    - Edited templates to pass on data from the backend.

7. The graded 1st phase report (attached)