Application Protocol: This system uses the TCP protocol and a simple send/receive loop where the client sends a message to the server, the server interprets in and does the operation the message specifies, and then the server sends a response back to be displayed by the client. This is the initialization code:

client

```
int create_socket() {
    struct addrinfo hints, *res;
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    if(getaddrinfo("localhost", NULL, &hints, &res)) {
        printf("getaddrinfo error!\n");
        exit(-1);
    }
    int ret = -1;
    if((ret = socket(res->ai_family, res->ai_socktype, res->ai_protocol)) < 0) {
        printf("socket error!\n");
        exit(-1);
    }
    struct sockaddr_in* inaddr = (struct sockaddr_in*)(res->ai_addr);
    inaddr->sin_port = htons(PORT);
    if(connect(ret, res->ai_addr, res->ai_addrlen) < 0) {
        printf("connect error!\n");
        exit(-1);
    }
    return ret;
}
```

server init

```
int start_server() {
    struct sockaddr_in sin;
    int ssoc;
    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    sin.sin_port = htons((unsigned short)PORT);
    if((ssoc = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        printf("socket error!\n");
        exit(-1);
    }
    if(bind(ssoc, (struct sockaddr*)&sin, sizeof(sin)) < 0) {
        printf("bind error!\n");
        exit(-1);
    }
    if(listen(ssoc, 5) < 0) {
        printf("listen error!\n");
        exit(-1);
    }
    return ssoc;
}
```

server listen loop

```
void listen_loop(int ssoc) {
    struct sockaddr_in fsin;
    int size = sizeof(fsin);
    int commsoc;
    char buf[MESSAGE_SIZE];
    char* response = malloc(MESSAGE_SIZE);
    char* user, *pass;

    NEWCONNECTION:

    if((commsoc = accept(ssoc, (struct sockaddr*)&fsin, &size)) < 0) {
        printf("accept error!\n");
        exit(-1);
    }

    while(1) {
        response = "";
        recvLoop(commsoc, buf, MESSAGE_SIZE);

        switch(buf[0]) {
            case 'C':
                memmove(buf, buf+1, strlen(buf));
                response = cancel_payment_request(buf);
                break;
            case 'G':
                memmove(buf, buf+1, strlen(buf));
                response = request_refund(buf);
                break;
```

Messages: All messages passed between server and client are a fixed size of 500 bytes as defined in common.h. The messages that the client passes to the server are identified by type based on the first byte, so that the server knows what the message is requesting it to do. The table below is all the headers and their meanings.

Header Character (first byte)	Meaning (interpreted by server)
J	register new user
L	login existing user

B	get balance
R	request payment
P	pay current pending payment request
N	refuse current pending payment request
T	print relevant transactions in ledger
G	request a refund
C	cancel a payment request you made
X	logout

When the server receives a message, it switches based on the first character and then removes the first character using memmove() so that the rest of the information in the message can be used to perform the operation it needs to do. The rest of this message is not explicitly outlined in the message protocol, so it can be any information it needs to be for the operation, such as a ledger entry (“P: Alice has requested $42 from Bob”) or a username and password (“myles 123”). In the case of a login request, for example, the full message the server receives before it manipulates it would be “Lmyles 123”.

```
    while(1) {
        response = "";
        recvLoop(commsoc, buf, MESSAGE_SIZE);

        switch(buf[0]) {
            case 'C':
                memmove(buf, buf+1, strlen(buf));
                response = cancel_payment_request(buf);
                break;
            case 'G':
                memmove(buf, buf+1, strlen(buf));
                response = request_refund(buf);
                break;
            case 'T':
                memmove(buf, buf+1, strlen(buf));
                response = get_ledger(buf);
                break;
            case 'N':
                memmove(buf, buf+1, strlen(buf));
                update_ledger(buf, 'X');
                response = "You have denied a pending transaction. The ledger has been updated.\n";
                break;
            case 'J':
                memmove(buf, buf+1, strlen(buf));
                user = strtok(buf, " ");
                pass = strtok(NULL, " ");
                response = add_user(user, pass);
```